2023

Novel neural architectures & algorithms for efficient inference

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

NOVEL NEURAL ARCHITECTURES & ALGORITHMS FOR EFFICIENT INFERENCE

by

ANIL KAG

B.Tech., Indian Institute of Technology Guwahati, 2014
M.S., Boston University, 2022

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2023

Approved by

First Reader

Venkatesh Saligrama, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering
Professor of Computer Science

Second Reader

Brian Kulis, PhD
Associate Professor of Electrical and Computer Engineering
Associate Professor of Systems Engineering

Third Reader

Alexander Olshevsky, PhD
Associate Professor of Electrical and Computer Engineering
Associate Professor of Systems Engineering

Fourth Reader

Kilian Quirin Weinberger, PhD
Professor of Computer Science
Cornell University

Fifth Reader

Prateek Jain, PhD
Senior Staff Research Scientist, Google AI
Adjunct Professor of Computer Science and Engineering
Indian Institute of Technology Kanpur

In loving memory of my Dada ji and Nana ji, one from whom I gleaned invaluable wisdom and the other whom I yearn to have known, but whose untimely departure cut short that opportunity.

Acknowledgments

Looking back on the unique trajectory of this thesis, I can see the flashbacks of many incredible people without whom this outcome would have been impossible.

First and foremost, I would like to thank my thesis advisor Prof. Venkatesh Saligrama for immense freedom and support during this journey. He enabled me to explore many crazy ideas and taught me the discipline of fleshing out these ideas to the general audience, including the dreaded "Reviewer 2". I have been very fortunate to be part of those long discussion sessions where we started with one problem and ended up generating many more problems to solve. This ensured I always had some problems to think about as a potential research direction. Finally, I am thankful for all the life skills I learned from him beyond academic life.

I want to thank my thesis committee members Prof. Brian Kulis, Prof. Alexander Olshevsky, Prof. Kilian Weinberger, and Dr. Prateek Jain. Their constructive feedback improved this thesis and my research directions in general. I am also grateful for the online learning lectures by Prof. Francesco Orabona, as they inspired some of the crazy ideas in this work. Finally, I am fortunate for my research collaborations with Dr. Prateek Jain. He has been a great role model. He helped me with many decisions during this journey, including the decision to pursue a Ph.D. at BU.

I am immensely grateful to Microsoft Research Bangalore for its exceptional Research Fellow program and to Dr. Manik Varma, my mentor during the fellowship. I would not have pursued this career if not for my MSR experience. I got my neck for empirical research from Manik. I learned the importance of application-driven research work from him.

Shout-out to amazing BU administrative folks. It is not easy to navigate the murky waters of official administrative activities. I want to thank Prof. Anna Swan for the first-year seminar course, as it helped in acclimatizing to the Ph.D. life at BU.

Next, I would acknowledge the support of ECE Ph.D. program managers Christine Ritzwoski and Nanna Syed. These were my go-to people for any admin issues. They were always supportive and quick to resolve issues. Finally, I want to thank Christina Polyzos for all the reimbursements she filed for many conference travels. I would also like to thank the BU Hariri Institute for the Research Fellowship in procuring valuable equipment during the pandemic as I adjusted to the work-from-home environment.

This journey would have been dull if not for my labmates Aditya Gangrade, Durmus Alp Emre Acar, Tianrui Chen, Samarth Mishra, Ruizhao Zhu, Param Budhraja, Pengkai Zhu, Ali Siahkamari, and Sheila W. Seidel. I owe my sanity to my office mates, Alp, Sheila, Tianrui, and Aditya. They ensured that I took regular breaks with interesting conversations during work hours. In addition, I would thank both Aditya and Alp for all the work and random life sessions that usually lasted hours. I would also like to thank my flatmate Ajay Brahmakshatriya for all the Bollywood movies we watched together that kept the gloomy days at bay and the support during various apartment hunts that drained my life. Finally, I want to thank my IIT-G friends, Harsh Gupta, Viresh Gehlawat, and Deepak Jain, for helping me with various hurdles during this journey. I am grateful to Deepak for the support during some dark hours.

Finally, I am grateful beyond words for the support of my family. I appreciate and dearly miss the warmth my paternal (Chandu & Maluji Kag) and maternal (Nandu & Badriji Devda) grandparents provided throughout my life. My sisters Ruchi and Sonam were a constant source of love and gossip back home. My parents, Anita and Motilal Kag, were my constant source of inspiration. Without their love and support, I would not have been where I am now. No words would be enough to thank them, so I would stop here and hope to be there for my family as they were for me all through these years.

NOVEL NEURAL ARCHITECTURES & ALGORITHMS FOR EFFICIENT INFERENCE

ANIL KAG

Boston University, College of Engineering, 2023

Major Professor: Venkatesh Saligrama, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering
Professor of Computer Science

ABSTRACT

In the last decade, the machine learning universe embraced deep neural networks (DNNs) wholeheartedly with the advent of neural architectures such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), transformers, etc. These models have empowered many applications, such as ChatGPT, Imagen, etc., and have achieved state-of-the-art (SOTA) performance on many vision, speech, and language modeling tasks. However, SOTA performance comes with various issues, such as large model size, compute-intensive training, increased inference latency, higher working memory, etc. This thesis aims at improving the resource efficiency of neural architectures, i.e., significantly reducing the computational, storage, and energy consumption of a DNN without any significant loss in performance.

Towards this goal, we explore novel neural architectures as well as training algorithms that allow low-capacity models to achieve near SOTA performance. We divide this thesis into two dimensions: *Efficient Low Complexity Models*, and *Input Hardness Adaptive Models*.

Along the first dimension, i.e., *Efficient Low Complexity Models*, we improve DNN

performance by addressing instabilities in the existing architectures and training methods. We propose novel neural architectures inspired by ordinary differential equations (ODEs) to reinforce input signals and attend to salient feature regions. In addition, we show that carefully designed training schemes improve the performance of existing neural networks. We divide this exploration into two parts:

(A) Efficient Low Complexity RNNs. We improve RNN resource efficiency by addressing poor gradients, noise amplifications, and BPTT training issues. First, we improve RNNs by solving ODEs that eliminate vanishing and exploding gradients during the training. To do so, we present Incremental Recurrent Neural Networks (iRNNs) that keep track of increments in the equilibrium surface. Next, we propose Time Adaptive RNNs that mitigate the noise propagation issue in RNNs by modulating the time constants in the ODE-based transition function. We empirically demonstrate the superiority of ODE-based neural architectures over existing RNNs. Finally, we propose Forward Propagation Through Time (FPTT) algorithm for training RNNs. We show that FPTT yields significant gains compared to the more conventional Backward Propagation Through Time (BPTT) scheme.

(B) Efficient Low Complexity CNNs. Next, we improve CNN architectures by reducing their resource usage. They require greater depth to generate high-level features, resulting in computationally expensive models. We design a novel residual block, the Global layer, that constrains the input and output features by approximately solving partial differential equations (PDEs). It yields better receptive fields than traditional convolutional blocks and thus results in shallower networks. Further, we reduce the model footprint by enforcing a novel inductive bias that formulates the output of a residual block as a spatial interpolation between high-compute anchor pixels and low-compute cheaper pixels. This results in spatially interpolated convolutional blocks (SI-CNNs) that have better compute and performance trade-offs.

Finally, we propose an algorithm that enforces various distributional constraints during training in order to achieve better generalization. We refer to this scheme as distributionally constrained learning (DCL).

In the second dimension, i.e., *Input Hardness Adaptive Models*, we introduce the notion of the hardness of any input relative to any architecture. In the first dimension, a neural network allocates the same resources, such as compute, storage, and working memory, for all the inputs. It inherently assumes that all examples are equally hard for a model. In this dimension, we challenge this assumption using input hardness as our reasoning that some inputs are relatively easy for a network to predict compared to others. Input hardness enables us to create selective classifiers wherein a low-capacity network handles simple inputs while abstaining from a prediction on the complex inputs. Next, we create hybrid models that route the hard inputs from the low-capacity abstaining network to a high-capacity expert model. We design various architectures that adhere to this hybrid inference style. Further, input hardness enables us to selectively distill the knowledge of a high-capacity model into a low-capacity model by cleverly discarding hard inputs during the distillation procedure.

Finally, we conclude this thesis by sketching out various interesting future research directions that emerge as an extension of different ideas explored in this work.

Contents

List of Tables

List of Figures

List of Abbreviations

AI	.....	Artificial Intelligence
CNN	.....	Convolutional Neural Network
DCL	.....	Distributionally Constrained Learning
DiSK	.....	Distilling Scaffolded Knowledge
DNN	.....	Deep Neural Network
FPTT	.....	Forward Propagation Through Time
GPU	.....	Graphics Processing Unit
iRNN	.....	Incremental Recurrent Neural Network
ML	.....	Machine Learning
ODE	.....	Ordinary Differential Equation
OSP	.....	One Sided Prediction
PDE	.....	Partial Differential Equation
RNN	.....	Recurrent Neural Network
SC	.....	Selective Classification
SI-CNN	.....	Spatially Interpolated Convolutional Neural Network
SOTA	.....	State-of-the-art
TARNN	.....	Time Adaptive Recurrent Neural Network
TPU	.....	Tensor Processing Unit

Chapter 1

Introduction

1.1 Motivation

In the last decade, the machine learning (ML) universe embraced deep neural networks (DNNs) wholeheartedly. More so in the past few years, with the advent of large-scale models such as DALLE-2(Ramesh et al., 2022), Imagen(Saharia et al., 2022), ChatGPT(OpenAI, 2023), Codex(Chen et al., 2021b), Whisper(Radford et al., 2022), etc. These models have remarkable capabilities, including some understanding of the language, vision, and speech domains. It was possible due to advances in training algorithms, hardware infrastructure, and neural constructs. For instance, neural accelerators like GPUs/TPUs, auto differentiation algorithms offered by libraries such as PyTorch (Paszke et al., 2019) / Tensorflow(Abadi et al., 2015), convolutional models(Tan and Le, 2021) for image classification, transformers(Brown et al., 2020) for natural language processing, diffusion models(Saharia et al., 2022) for mixed modalities, etc. While these architectures achieve state-of-the-art (SOTA) performance on many tasks, their storage, energy, and computational requirements restrict their deployment to large cloud clusters. More specifically, the following issues become bottlenecks in terms of the resource-efficiency during the training and inference phase.

- *Storage Complexity & Working Memory.* High-capacity models have large storage requirements. For instance, GPT-3 (Brown et al., 2020) model has nearly

175 billion parameters that occupy 800 GB storage on a hard disk. It is prohibitively large for deployment in resource-constrained setups such as mobile phones. Even if somehow we can store the model, the more pressing issue comes down to utilizing this model while training or inference, as that would require loading each layer in the fast memory like RAM or GPU memory during the forward pass(Dreuning et al., 2022). It would indeed slow down the entire training and inference stage(Banbury et al., 2021). It would also be a severe design flaw in a DNN architecture to have asymmetric memory utilization of different DNN layers as a costly layer would become the bottleneck even if other layers are lightweight.

- *Computational Complexity & Carbon Footprint.* Large models require high-end graphics processing units — such as GPUs/TPUs — for training. A typical training cycle involves experimentation, including hyper-parameter tuning, architecture selection, distributed pipelines, etc. This entire process requires a considerable amount of energy(Desislavov et al., 2023), leads to greenhouse gas emissions, and contributes significantly higher carbon footprint(Patterson et al., 2021). While it would seem that one can write the training cost as a one-time investment, it is not true simply because in a practical industrial use-case, a model goes through many training cycles, and due to the inherent nature of the data collection and distribution drift, it is likely that the models need to be retrained, with a significant cost. Even if we can write the training as a one-time investment, this does not reduce the carbon footprint (Cai et al., 2020) as the inference stage — where users interact with this system — still requires lots of computational resources. Thus, large models have significant computational resources resulting in a high carbon footprint during the training and inference stage.

- *Privacy.* Given the current status quo of inference on the cloud setup due to storage and compute requirements, any user would inevitably have to share their data with cloud service providers. It leads to severe privacy concerns, and users would prefer on-device inference rather than on-cloud deployment. For instance, it would be beneficial from a consumer standpoint that conversations with personal assistants (like Alexa or Siri) be private. Similarly, artists would prefer privacy during art development with diffusion models like DALLE-2(Ramesh et al., 2022) and Imagen(Saharia et al., 2022).

- *Real Time Inference.* Many ML applications require immediate responses to the input data. For example, decision-making in self-driving cars, detection in surveillance cameras, responsiveness in conversations with personal assistants, etc., require real-time inference. Storage and computational requirements of large DNNs mean we can only deploy tiny variants of such systems in constrained setups and take a hit in the performance. Or, we rely on a cloud service provider and trade off our privacy for better performance.

- *Open Access.* Research on Large DNNs – due to their resource requirements and huge monetary cost – would be limited to industrial labs. This would force academic researchers with limited resources to treat these systems as black-box. This resource barrier would limit innovations in the architectures and algorithmic design required to improve these systems. In addition, it would be extremely hard to place safeguards such as trustworthiness, transparency, fairness, and ethics in such ML models.

Thus, it is imperative to focus on improving the resource efficiency of DNNs to overcome the abovementioned issues.

1.2 Problem Definition: Efficient Inference

For an ML model, efficient inference refers to the problem of making accurate predictions quickly with minimal resource usage. In this problem, there are two key metrics, (a) resource usage (such as model size/inference latency) and (b) predictive performance (such as prediction accuracy). Mathematically, to maximize predictive performance $\mathcal{A}$ given a constraint ϱ on the resource usage $\mathcal{R}$, we can write it as the following optimization problem,

$$\max \ \mathcal{A} \text{ s.t. } \mathcal{R} \leq \varrho. \tag{1.1}$$

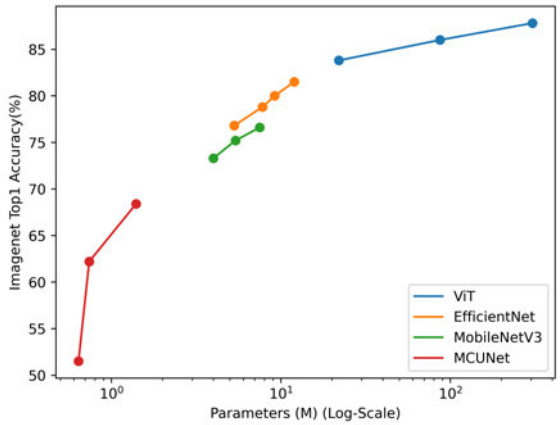


Figure 1·1: ImageNet Classification. We evaluate different architecture families in this plot, namely MCUNet(Lin et al., 2020a), MobileNetV3(Howard et al., 2019), EfficientNet(Tan and Le, 2019), and Vision Transformers(Dosovitskiy et al., 2021). We plot the Top-1 accuracy (in %) achieved by these models on the ImageNet-1K validation dataset against the number of parameters (in millions).

This coupling enables any reasonable ML system design to trade off resource usage for performance, i.e., improve performance by consuming more resources or reduce resource usage with a decrease in performance. As an example, consider the ImageNet(Russakovsky et al., 2015) object recognition task in Figure 1·1, where we have very tiny models such as MCUNets(Lin et al., 2020a) achieving close to 51% accu-

racy, and on the other hand, we have large vision models such as vision transformers (Dosovitskiy et al., 2021) achieving close to 89% accuracy. Typically, practitioners operate with some resource constraints, say deploying a model on mobile phones. It puts a hard limit on resource usage, such as model size, inference latency, etc. Given a resource budget, a practitioner deploys the best-performing model that satisfies the resource constraints. Hence, models that are resource efficient would be desirable. In this specific example, although one would desire to deploy vision transformers to achieve the best performance, but the mobile phone resource constraints such as real-time inference latency and model storage would restrict us to models like MobileNetV3(Howard et al., 2019) or MCUNets(Lin et al., 2020a). Thus, at the heart of efficient inference lies the task of improving the performance of a model within given resource constraints. It could either amount to developing new neural architectures that improve upon the current SOTA models or designing new algorithms to improve the performance of existing architectures.

1.3 Existing Solutions

There has been significant progress in improving the resource efficiency of DNNs. We will briefly summarize various strategies used in the literature. We will carefully re-visit various closely related works in the review chapters when we go through our proposed solutions. Broadly, we can divide the literature into the following key areas. It is worth noting that a practitioner would typically create a recipe that combines many of these techniques to get the best performance within resource constraints.

1.3.1 Architecture Design

DNNs have evolved significantly from early attempts like Multi-Layer Perceptron, LeNet(Lecun et al., 1998), and vanilla RNN(Hochreiter, 1991) architectures. These

attempts had only a few layers and could not scale to their deeper variants due to architectural issues, hardware limitations, data scarcity, and poor algorithms. They were largely supplanted by simpler models such as support vector machines (SVMs)(Cortes and Vapnik, 1995) and boosting/bagging(Dietterich, 2000) models. AlexNet(Krizhevsky et al., 2012) demonstrated that deeper models could be trained using GPUs and could significantly surpass previous state-of-the-art image recognition performance. It led to even deeper architectures like VGGNet(Simonyan and Zisserman, 2015). These new developments came with issues such as vanishing and exploding gradients (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013b; He et al., 2016), which meant that any deeper models would yield diminishing gains and might result in unstable training trajectory. This was followed by the introduction of skip-connections (ResNet(He et al., 2016)/ Highway networks(Srivastava et al., 2015)) that enabled deep models with as many as a thousand layers. While these architectures enabled training deeper networks, their resource usage increased substantially with diminishing performance returns. It started a race for the design of resource-efficient residual blocks(Howard et al., 2017; Howard et al., 2019; Gholami et al., 2018; Zagoruyko and Komodakis, 2016).

Many subsequent works proposed design changes that improved resource efficiency through improved non-linearities (ReLU(Nair and Hinton, 2010), SeLU(Klambauer et al., 2017), GeLU(Hendrycks and Gimpel, 2016), etc.), normalization layers (Batch-Norm(Ioffe and Szegedy, 2015), Layer-Norm(Ba et al., 2016), Group-Norm(Wu and He, 2018)), overfitting measures (dropout(Srivastava et al., 2014)/drop-path(Larsson et al., 2017)), efficient layers (squeeze-and-excite(Hu et al., 2018), separable convolutions, low-rank decomposition, etc.), etc. This process was further accelerated through an architecture search process that enabled efficient search for new architectures as a combination of various building blocks(White et al., 2021; Liu et al., 2019a;

Wu et al., 2019; Zoph et al., 2018).

In parallel, there have been improvements in the sequential models. Since RNNs unroll their processing over a large time horizon in the sequential input, they suffer significantly from the vanishing and exploding gradient issues, more so than the feed-forward layers like CNNs / MLPs. LSTMs(Hochreiter and Schmidhuber, 1997b) were introduced to avoid the vanishing gradient problem by introducing a gating mechanism to control the flow of information during state transition. This led to the development of GRU(Cho et al., 2014b) and other gated variants(Kusupati et al., 2018) that address the vanishing and exploding gradients while improving resource efficiency. Researchers have also focused on RNNs that evolve with well-conditioned state transition matrices such as Unitary or Orthogonal transitions (Unitary RNNs(Mhammedi et al., 2017; Arjovsky et al., 2016; Kerg et al., 2019; Lezcano-Casado and Martínez-Rubio, 2019), Spectral RNNs(Zhang et al., 2018a), etc.) to improve RNN training and achieve better generalization. In addition, some works(Chang et al., 2019; Rusch and Mishra, 2021a; Niu et al., 2019; Chen et al., 2018; Rubanova et al., 2019) have tried to address these issues through a transition function that evolves via an ordinary differential equation (ODEs).

One of the primary issues with RNNs is their sequential processing of the inputs, as it requires careful design of the transition matrices to result in models that can capture long-range dependencies for lengthy input sequences, typical in tasks like text summarization, question answering, etc. This process led to the Transformer(Vaswani et al., 2017) architectures where instead of sequential processing on the input as RNNs, they look at the entire sequence and compute pairwise dependencies between each input token. Thus, avoiding issues that exist in RNNs such as long-range dependencies, vanishing or exploding gradients, etc. But, these transformers come with huge computational and resource costs. There have been many efforts

to address these issues(Tay et al., 2022). The success of these transformer architectures in the NLP domain led to vision transformers(Dosovitskiy et al., 2021) with the eventual development of transformer architecture capable of handling multi-modal data(Saharia et al., 2022).

Many popular neural networks are static w.r.t. an input, i.e., a network does the forward pass for every example irrespective of their hardness. They utilize all network resources without considering into account input characteristics. This viewpoint is in contrast to the human brain, where simple inputs are handled without much processing while we ponder longer on the complex inputs to get the correct prediction. There have been some efforts in the literature to incorporate these dynamics in DNNs, namely, efforts like adaptive computation(Graves, 2016; Campos et al., 2018; Chung et al., 2016; Yu et al., 2017; Jernite et al., 2017; Hansen et al., 2019) in RNNs, and adaptive inference(Han et al., 2022; Bolukbasi et al., 2017; Nan and Saligrama, 2017a; Li et al., 2021) in the CNNs.

We point out that we will cover some of these architectures in the upcoming chapters. We refer readers interested in gaining a more nuanced understanding of these efficiency aspects to the survey works(Han et al., 2022; Li et al., 2022; Lipton et al., 2015; Tay et al., 2022).

1.3.2 Training Algorithms

While architecture is an important aspect of achieving good performance, an equally important piece is the training algorithm. Poorly trained architectures yield subpar performance.

One of the primary reasons deep learning took off the runway is the access to libraries such as PyTorch(Paszke et al., 2019), Tensorflow(Abadi et al., 2015), Caffe(Jia et al., 2014), Theano(The Theano Development Team et al., 2016), etc. It enabled

computing gradients through auto-differentiation in the backward pass and made training even large models super easy using back-propagation. In contrast, earlier attempts at training neural networks required computing gradients by hand and was a tiresome and error-prone process. But, back-propagation results in auto gradients for a training loss function. This can be further fed into various gradient-based optimization schemes to yield better performance, such as SGD+Momentum(Ruder, 2016), Adam(Kingma and Ba, 2015), AdaGrad(Duchi et al., 2011), RMSProp(Tieleman et al., 2012), LARS(You et al., 2017), LAMB(You et al., 2020), etc.

Since these optimizers aim to minimize the training loss function, it is necessary to use loss functions that result in (a) a smooth surface for easier optimization and (b) a solution with better generalization, i.e., nearly approximate the test time behavior. Below, we list various methods used in the literature to achieve these objectives.

- *Promote Generalizing Properties.* Many works have associated various loss landscape properties to a low generalization error, namely properties such as flat minima (Foret et al., 2021; Keskar et al., 2017), consistency(Xie et al., 2020), robustness(Yang et al., 2022), adversarial noise(Volpi et al., 2018), deep mutual learning(Zhang et al., 2018c), etc. In parallel, (Jiang et al., 2020) empirically examined the diverse set of complexity measures and tabulated the correlation between these measures and the generalization gap observed in DNNs trained with CE loss.

- *Additional Data.* It is always desirable to have more training data to generalize better to the test time behavior. It helps to approximate well the unknown probability distribution behind the task. Note that having access to good quality labelled data would help improve the performance of even somewhat mediocre architectures with even naive optimization schemes. But, acquiring high-quality labelled data is an expensive task, both in terms of the financial cost and the

human hours required. Thus, the community has started exploring various mechanisms to augment the small but high-quality labelled data.

One way to achieve this is data augmentation, i.e., to incorporate multiple views of the labelled data point. For the vision domain, in the past, the field stuck to simpler augmentations like cropping, resizing, translating, horizontal flips, etc. More recently, the field has transitioned into more complex augmentation strategies such as AutoAugment(Cubuk et al., 2019), Cutout(DeVries and Taylor, 2017), CutMix(Yun et al., 2019), RandAugment(Cubuk et al., 2020), etc. Similarly in the language domain, we have Back-Translation(Sennrich et al., 2016), Word-Replacement in TF-IDF(Xie et al., 2020), Cross-View Training(Clark et al., 2018), etc. While these augmentations reduce the chance of overfitting with large models, they may not be natural augmentations from the human viewpoint. It may not be helping to approximate the unknown data distribution.

Another popular strategy is to utilize the unlabelled data, leading to unsupervised or semi-supervised(Yang et al., 2022) learning paradigms. Note that unsupervised data is abundant in the wild, and with some precaution, it can yield good quality feature representation for solving downstream tasks. One key aspect of utilizing the unsupervised data is the masking, or what I refer to as the fill-in-the-blank strategy(He et al., 2022; Devlin et al., 2019), where a neural network looks at the masked /blanked-out inputs and is asked to predict the masked-out part of the input. This has been a widely adapted strategy in many learning algorithms like self-supervised learning(Jaiswal et al., 2021), masked training(He et al., 2022), next token prediction in learning language models(Merity et al., 2018; Brown et al., 2020), and so on.

- *Additional Supervision.* In terms of supervision, many works have attempted to

incorporate feedback other than just labelled supervision. For instance, knowledge distillation(Gou et al., 2021) uses the supervision from a large teacher model into a student network. It enables the student to learn some additional problem structures not available through the labelled data. Another paradigm is self-supervision(Jaiswal et al., 2021; Tarvainen and Valpola, 2017; Grill et al., 2020; Caron et al., 2021; Hochreiter and Schmidhuber, 1997a; Foret et al., 2021; Cha et al., 2021; Chen et al., 2020b; He et al., 2020) where a model uses an exponential moving average of its weights during training as the reference teacher model and uses this as a distillation objective. Other works, such as label smoothing(Müller et al., 2019), have recommended adding some systematic noise to the labelled data to avoid overfitting.

- *Scheduling Concept Learning.* Researchers have also emphasized concept learning in relation to the way humans learn. In particular, progressive learning(Tan and Le, 2021) refers to learning simple concepts first and progressively learning complex concepts during the training trajectory. Curriculum learning(Bengio et al., 2009; Hacohen and Weinshall, 2019; Graves et al., 2017) refers to the fixed strategy of prioritizing examples during batched training such that a model looks at the harder examples, followed by easier examples later in the epoch. Similarly, earlier works have argued the importance of hard instances(Zhou et al., 2020) having more weight in shaping the decision boundary to focus more often on showing hard instances for training.

1.3.3 Miscellaneous

Researchers have used various strategies in a post-hoc manner to compress an architecture during the inference phase and many times even during the training stage. Quantization(Gholami et al., 2021; Jacob et al., 2018) schemes have been developed

to reduce the model arithmetic precision from floating point FP32 to low-precision numerics like INT4. It yields significant computational speed-ups at only slight degradation in performance. Similarly, network pruning(Liang et al., 2021; Han et al., 2016) is a strategy to convert a dense DNN into a sparse DNN by removing unnecessary connections in a network. Similarly, low-rank(Idelbayev and Carreira-Perpinan, 2020; Howard et al., 2017; Zhang et al., 2018a) factorization and CUDA kernel fusion(Filipovic et al., 2015) have been applied in the literature to further reduce the computational and storage footprint of the neural networks.

1.4 Approach in this thesis

This thesis aims at improving the resource efficiency of neural architectures, i.e., significantly reducing the computational, storage, and energy consumption of a DNN without any significant loss in performance. Towards this goal, we explore novel neural architectures as well as training methods that allow low-capacity models to achieve near SOTA performance. While our proposed ideas are generic in nature, we ground their utility in applications by focusing on learning problems in sequential processing (time-series and natural language) (Devlin et al., 2019; Wu et al., 2016; Anguita et al., 2013; Shahroudy et al., 2016) and vision domains(Tan and Le, 2019; Howard et al., 2019). Specifically, our work follows the following two themes:

1.4.1 Efficient Low Complexity Models

In this theme, we improve the performance of existing DNNs by either addressing various design issues or improving training algorithms to yield better performance. Broadly, we focus on *Recurrent Neural Networks (RNNs)* and *Convolutional Neural Networks (CNNs)* and discuss extensions of these ideas to transformer architectures.

First, we address various fundamental challenges posed by RNNs: (a) during

training, the gradient of loss back-propagated in time could suffer from exponential decay/explosion (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013b) resulting in poor generalization for processes exhibiting long-term dependencies (LTD), (b) sequential data such as time-series has a low signal to noise ratio, thus improperly designed RNNs could propagate noise in the hidden states, and (c) training with unrolling in time requires storing all the intermediate hidden states, resulting in significant memory cost for very large sequences. These instabilities lead to sub-optimal performance. We address these issues by proposing novel architectures (Kag et al., 2020; Kag and Saligrama, 2021a) inspired by constructs such as ordinary differential equations (ODEs). In addition, we propose novel a training methodology (Kag and Saligrama, 2021b) to achieve performance gains.

Next, we improve CNN architectures by reducing their resource usage. They require greater depth to generate high-level features, resulting in computationally expensive models. We design a novel residual block, Global layer (Kag and Saligrama, 2022), that constrains the input and output features by approximately solving partial differential equations (PDEs). It yields better receptive fields than traditional convolutional blocks and thus results in shallower networks. Further, we reduce the model footprint by enforcing a novel inductive bias that formulates the output of a residual block as a spatial interpolation(Kag et al., 2023d) between high-compute anchor pixels and low-compute cheaper pixels. This results in spatially interpolated convolutional blocks (SI-CNNs) that have better compute and performance trade-offs. Finally, we propose distributionally constrained learning (DCL)(Kag et al., 2023b), an algorithm that enforces various distributional constraints during training in order to achieve better generalization.

1.4.2 Input Hardness Adaptive Models

Although efficient low-complexity models provide non-trivial gains over the existing networks, we further improve these leveraging per input hardness, namely the notion that not all input instances are equally hard to predict for a DNN. We use input hardness to specialize DNNs in various input regions. First, we allow models to abstain from a prediction on a few inputs, resulting in abstaining models (Gangrade et al., 2021). Secondly, we create hybrid models using a few such specialized networks. In its simplest form(Kag et al., 2023c), we introduce the notion of input hardness to specialize low-capacity networks for easy examples and routing difficult ones to a high-capacity network. Finally, the low-capacity model deployed on the edge device handles easy examples and routes hard ones to the high-capacity network living in the cloud. This hybrid design achieves SOTA accuracy and significantly outperforms the existing edge-deployable efficient low-complexity models. Finally, we incorporate this input hardness notion in training itself. We develop a novel distillation scheme, DiSK (Kag et al., 2023a), where the teacher scaffolds the student's prediction on hard-to-learn examples. It smoothens the student's loss landscape so that the student encounters fewer local minima. As a result, it has good generalization properties.

1.5 Contributions

Our contributions are three-folds as listed below.

1.5.1 Efficient Low Complexity RNNs

We focus on improving the trainability of recurrent neural networks, namely issues such as poor gradients during training, noise amplification in the latent space, and significant memory cost in unrolling during backpropagation through time.

First, we propose *Incremental Recurrent Neural Network (iRNN)* (Kag et al.,

2020) wherein the state transition function is defined by an ordinary differential equation such that the hidden state vectors keep track of incremental changes. iRNN exhibits identity gradients and is able to account for long-term dependencies (LTD). We also present a simple and cost effective discretization scheme to solve this ODE. Finally, we provide empirical evidence that iRNNs do not face vanishing and exploding gradients during training. As a result, iRNNs achieve better performance at similar cost when compared with existing architectures.

Next, we extend iRNNs by developing *Time Adaptive Recurrent Neural Network (TARNN)* (Kag and Saligrama, 2021a) which dynamically adapts the time constants of the state transition ODE, thus enabling the suppression of noise in the inputs. By varying the time constants, TARNN can ponder longer over informative input segments and strengthen their contributions in the hidden state. At any time step, TARNN skips processing the input whenever it predicts that the input at this time step is noise. We provide the details of this time adaptive logic. In addition, TARNN's parameterization enables it to maintain identity gradients where desired during training. Through various synthetic as well as real-world benchmark datasets, we show that TARNN outperforms both iRNN as well other RNNs.

Finally, we develop a novel algorithm, *Forward Propagation Through Time (FPTT)* (Kag and Saligrama, 2021b), where at each time, for an instance, we update RNN parameters by optimizing an instantaneous risk function. Our proposed risk is a regularization penalty at time t that evolves dynamically based on previously observed losses and allows for RNN parameter updates to converge to a stationary solution of the empirical RNN objective. FPTT mitigates the vanishing/exploding gradient issues even in poorly designed architectures. In addition, FPTT reduces memory consumption as it does not unroll the RNN transition for the entire sequence length as typically required by BPTT.

1.5.2 Efficient Low Complexity CNNs

Convolutional neural networks typically require a lot of repetitions of the residual blocks in order to capture a large receptive field. This results in a large storage and computational footprint. We improve their resource efficiency by designing novel feature layers and training methods.

First, we show how to utilize differential equations to reduce the CNN computational/storage footprints. We develop a new feature layer, called the *Global Layer* (Kag and Saligrama, 2022), that enforces partial differential equation (PDE) as a constraint on the feature maps. This layer increases the receptive field without the additional overhead of large kernel convolutions. These constraints are solved by embedding iterative schemes in the network. Thus, the proposed layer can be embedded in any deep CNN to transform it into a shallower network.

Next, we design a novel interpolation scheme, named *Spatial Interpolation*(Kag et al., 2023d), which decomposes a residual block as an interpolation between features processed at a low-resolution sampling of the input features and cheaper features processed at the input resolution. This interpolation scheme reduces the computational cost of any generic convolutional residual block. We instantiate spatially interpolated variants of the popular Inverted Residual (Howard et al., 2019) and Fused-MBConv (Tan and Le, 2021) blocks.

Finally, we develop *Distributionally Constrained Learning (DCL)* (Kag et al., 2023b), a novel training method to improve generalization. It penalizes variation in *model-predictions* during training, which arises from the randomness of inputs and stochasticity of model updates. Additionally, in contrast to prior works, where constraints are handled by means of a regularization penalty and choice of hyperparameters, we propose a novel *cosine-scheduling* scheme coupled with a *multi-phase budget update* for gradually hardening distribution penalty that allows for sufficient

exploration during the training process.

1.5.3 Input Hardness Adaptive Models

We integrate the notion of input hardness in neural network design and learning strategies.

First, we develop an *abstaining classifier* (Gangrade et al., 2021) that learns to reject a few hard input instances and selectively predicts the remaining input space. We learn this abstaining model using a novel one-sided prediction objective that builds confidence region per class label.

Next, we design *hybrid models* (Kag et al., 2023c) that combines a low-capacity abstaining classifier with a high-capacity model such that most of the inference happens with the low-capacity model and the rejected samples from this abstaining classifier are routed to the high capacity model. Thus, allowing a much better resource trade-off in terms of performance and compute usage during inference.

Finally, we integrate input hardness in training low-capacity models. More specifically, we develop *Distilling Scaffolded Knowledge (DiSK)* (Kag et al., 2023a) wherein we use a large capacity teacher to selectively distill knowledge into a tiny student such that it focuses its model capacity on easier input region space. In doing so, the student's training loss smoothens, and it is able to learn a better generalizing solution than traditional methods such as training with cross-entropy loss or knowledge-distillation objective.

1.6 Thesis Overview

The rest of the thesis is in the same order as the three contributions discussed earlier. In the first part (Chapter 2-5), we design low complexity efficient RNN architectures. In the second part (Chapter 6-9), we propose new convolutional residual layers and

training algorithms to improve CNN architectures. Finally, in the third part (Chapter 10-13), we explore the notion of input hardness and design input adaptive architectures and algorithms. We leave the supplementary details, such as hyper-parameters, dataset descriptions, architecture configurations, etc., to the Appendices (Chapter A-J). Note that all the chapters (except for Chapter 8 and Chapter 9) are based on peer-reviewed conference publications published during my doctoral studies and have been edited in this thesis for coherency. We highlight the chapter organization below.

We begin our journey with the design of efficient low-complexity RNNs.

In Chapter 2, we review RNN architectures and various challenges that arise during training and inference. We discuss various solutions proposed in the literature to address the same. We discuss the vanishing and exploding gradients problems in RNN training as well as the noise amplification issue that arises while dealing with sequential inputs. In addition, we review the back-propagation through time (BPTT) algorithm and enumerate various issues it brings to the table.

In Chapter 3, we discuss *Incremental Recurrent Neural Network (iRNN)*(Kag et al., 2020) wherein hidden state transitions are represented by an ODE that eliminates vanishing and exploding gradients during RNN training. It enables identity gradients in the back-propagation. As a result, iRNNs achieve better performance at a similar cost compared to existing architectures.

In Chapter 4, we present *Time Adaptive Recurrent Neural Network (TARNN)*(Kag and Saligrama, 2021a) aimed at solving the noise amplification issue in RNN transitions. We modify the time constants in the ODE-based hidden state transitions such that the ODE solver is invoked only when a signal is present at a given time step.

So far, our focus has been on introducing new RNN architectures. In Chapter 5, we show that an improved training scheme yields significant improvements in the performance of existing neural architectures. We present an alternative to BPTT, i.e.,

Forward Propagation Through Time (FPTT)(Kag and Saligrama, 2021b), wherein instead of propagating the hidden states from the initial timestep to the final timestep, and finally back-propagating the gradients, we keep on moving the hidden states forward as well as keep updating the parameters in a forward manner. Thus, avoid the back-propagation throughout the sequence length. It helps us eliminate the vanishing and exploding gradients during the training phase and, as such, helps in reaching better generalizing parameter space.

Next, we move on to designing efficient low-complexity CNN architectures.

In Chapter 6, we review CNN architectures and previous attempts at improving their resource efficiency. We discuss some key issues underlying many of these efforts, namely low receptive field, residual block repetitions, and lack of exploitation of spatial redundancy. In addition, we discuss various training strategies adopted in the literature to improve generalization.

In Chapter 7, we propose a new feature layer, called the *Global Layer*(Kag and Saligrama, 2022), which enforces PDE constraints on the feature maps, resulting in rich features. This proposed layer can be embedded in any deep CNN to transform it into a shallower network.

In Chapter 8, we design a novel *Spatial Interpolation*(Kag et al., 2023d) scheme to exploit the redundancies in the spatial structures in the convolutional residual blocks. It decomposes the residual block as an interpolation between features processed at a low-resolution sampling of the input features and cheaper features processed at the input resolution. This interpolation strategy is applicable to any generic convolutional residual block.

In Chapter 9, we propose *Distributionally Constrained Learning (DCL)*(Kag et al., 2023b), a novel algorithm that incorporates distributional constraints for training deep neural networks to improve generalization.

In the final part, we develop architectures and algorithms that incorporate input hardness for adaptively trading off resources and model prediction accuracy.

In Chapter 10, we analyze the static nature of the most DNNs and review various attempts at incorporating input-hardness adaptive computation in the neural networks. We discuss the literature on selective classification, a.k.a. learning with a reject option, other strategies in which an abstaining classifier is coupled with other expert models to cover the entire input space, and various training strategies in this context.

In Chapter 11, we propose *One-Sided Prediction (OSP)*(Gangrade et al., 2021), a novel method for selective classification, a problem which allows a classifier to abstain from predicting some instances. In contrast to prior gating or confidence-set-based work, OSP optimizes a collection of class-wise decoupled one-sided empirical risks and is, in essence, a method for explicitly finding the largest decision sets for each class with few false positives.

In Chapter 12, we couple an abstaining classifier with a high-capacity expert model such that abstained inputs from the selective classifier are routed to the expert model. This results in *Hybrid Models*(Kag et al., 2023c) where a small abstaining classifier handles most of the easy input space, and few difficult instances are handled by a high-capacity expert model.

In Chapter 13, we integrate the input-hardness notion into a training algorithm. We propose a novel knowledge distillation method, DiSK(Kag et al., 2023a), to selectively instill teacher knowledge into a student model motivated by situations where the student's capacity is significantly smaller than that of the teachers. The teacher model helps the student by censoring hard-to-learn examples.

Finally, we conclude this thesis in Chapter 14. We discuss various future research directions that emerge out of this work.

Part I

Efficient Low Complexity RNNs

Chapter 2

Recurrent Neural Network: Background

Recurrent Neural Networks (RNNs) are deep neural architectures designed to process sequential data such as time series or natural language. RNNs have been widely used in language modelling(Merity et al., 2018), speech recognition(Warden, 2017), time-series analysis(Shahroudy et al., 2016), etc. At any time step in the input sequence, RNNs keep track of a hidden state that summarizes the input sequence seen so far. This hidden state is updated upon observing the current input token via a coupled transformation between the previous hidden state and the input token, followed by some non-linear operation. Thus, the hidden state represents the input sequence in a latent space. Improving RNN performance is important since they are deployed in a number of IoT applications (Dennis et al., 2019) due to their light memory footprint.

2.1 Definition

Let us formally define a vanilla RNN. We will fix a notation before proceeding further. Let $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, $i \in [N]$ denote the N i.i.d. training data points drawn from an unknown probability distribution $\mathbb{P}$. Each $\mathbf{x}^{(i)}$ is a $T-$length $d-$dimensional sequential input. For classification problems, $\mathbf{y}^{(i)}$ is a terminal label $y_T^{(i)}$, taking values in a discrete set of C classes. For language modeling tasks, we let the true label be a process, $(y_1^{(i)}, \ldots, y_T^{(i)})$. The predictions $(\hat{y}_1^{(i)}, \ldots, \hat{y}_T^{(i)})$ for each input $\mathbf{x}^{(i)}$ can be computed from the $D-$dimensional hidden states $(\mathbf{h}_1^{(i)}, \ldots, \mathbf{h}_T^{(i)})$ obtained by apply-

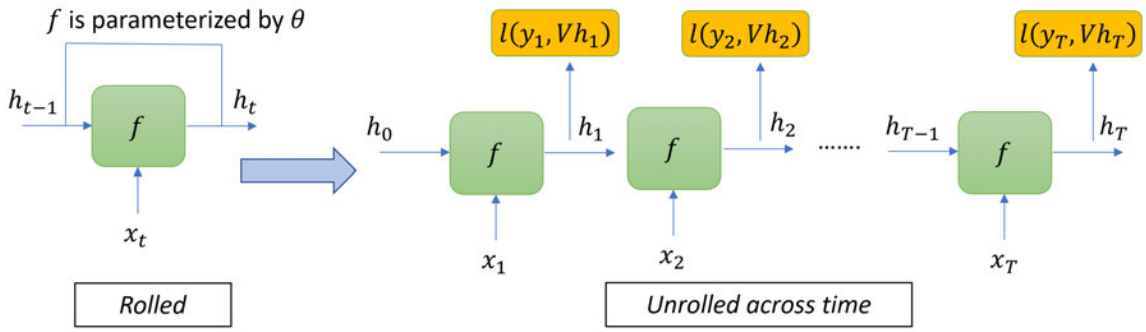


Figure 2·1: RNN State Transition: Unrolling RNN transition f across T time-steps, starting with initial hidden state h_0 till the final hidden state h_T.

ing RNN transition function described below. We omit superscripts when it is clear from the context. Unless stated otherwise, $\sigma(\cdot)$ denotes the sigmoid activation; $\phi(\cdot)$ refers to any non-linear activation such as a ReLU. At a time-step t, a vanilla RNN has access to hidden state $\mathbf{h}_{t-1}$ that summarizes the past $(\{\mathbf{h}_i\}_{i=0}^{t-1}, \{\mathbf{x}_j\}_{j=1}^{t-1})$. Upon receiving the new input $\mathbf{x}_t$, RNN updates the hidden state to $\mathbf{h}_t$ using the following transformation,

$$\mathbf{h}_t = f(\theta, \mathbf{h}_{t-1}, \mathbf{x}_t) \triangleq \phi(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \tag{2.1}$$

where $\mathbf{U} \in \mathbb{R}^{D\times D}, \mathbf{W} \in \mathbb{R}^{D\times d}, \mathbf{b} \in \mathbb{R}^D$ and $\theta = \{\mathbf{U}, \mathbf{W}, \mathbf{b}\}$. We obtain the RNN prediction at time t as the projection onto the output space through a linear classifier $\mathbf{V} \in \mathbb{R}^{C\times D}$ such that $\hat{y}_t = \mathbf{V}\mathbf{h}_t$. Figure 2·1 shows RNN transition for T time-steps using the state-transition denoted in Eq. 2.1. Note that the RNN model is defined by the learnable parameters $\{\theta, \mathbf{V}\}$ and the transition function f. We can learn these parameters by minimizing the empirical risk as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \ell(y_t^i, \hat{y}_t^i) \qquad \triangleq \arg\min_{\theta} \mathcal{L} \tag{2.2}$$

$$\forall i, t;\ \ \hat{y}_t^i = \mathbf{V}\mathbf{h}_t^i;\ \ \mathbf{h}_t^i = f(\theta, \mathbf{h}_{t-1}^i, x_t^i); \qquad \mathbf{h}_0^i = 0$$

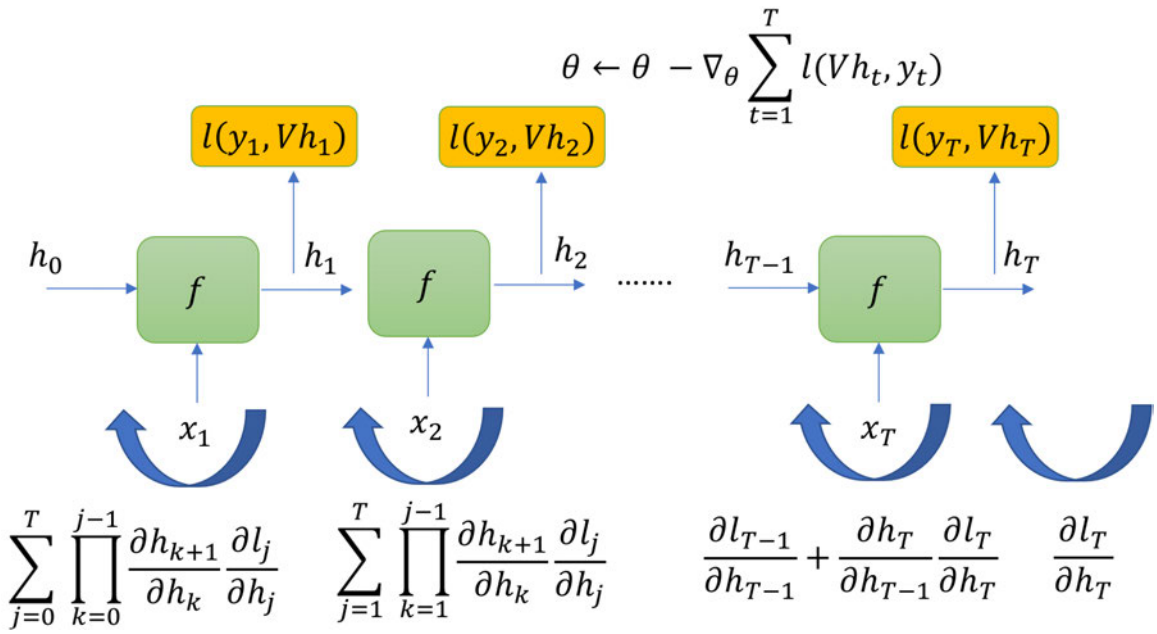


Figure 2·2: Back-Propagation Through Time (BPTT) Algorithm for training RNNs

where $\ell(y, \hat{y})$ is the loss function penalizing the prediction $\hat{y}$ for the ground truth label y. Many popular choices for this loss function exist, such as cross-entropy, mean-squared error, etc. We can use standard off-the-shelf optimizers to minimize the training loss in the setup mentioned earlier. Since this objective requires unrolling the RNN transitions for all the time steps in the input ($0 \rightarrow T$), the standard back-propagation algorithm used in this context is referred to as Back-Propagation-Through-Time (BPTT) (see Figure 2·2).

2.2 Trainability Challenges

In principle, RNNs are powerful enough to model many complex dynamics and should be able to retain information across various lengthy input sequences. But, in practice, they are very hard to train and suffer from many challenges, many of which arise from their transition function design as well as the BPTT training algorithm. Below, we

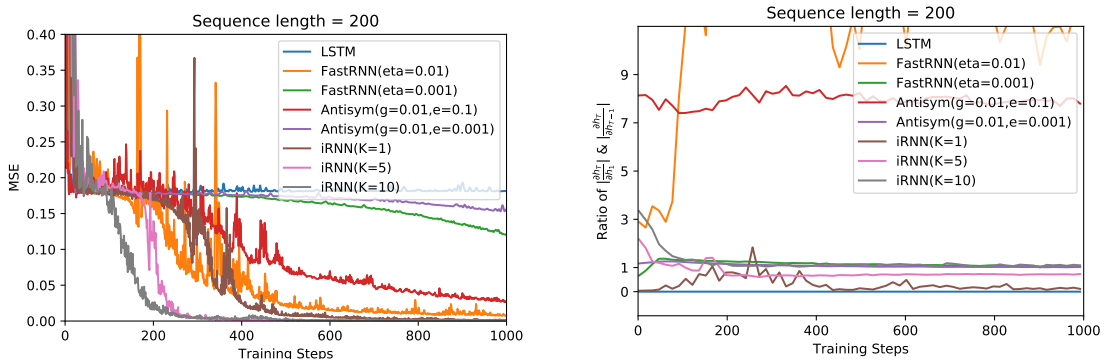
enumerate these issues, and in the next section discuss various existing solutions to these problems.

1. **Vanishing/Exploding Gradients.** During training using BPTT, the error gradient back-propagated (with one parameter say $\mathbf{U}$) can be written as follows,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\partial \ell(y_t^i, \hat{y}_i^t)}{\partial \mathbf{U}} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\partial \ell(y_t^i, \hat{y}_i^t)}{\partial \hat{y}_i^t} \frac{\partial \hat{y}_i^t}{\partial \mathbf{h}_t^i} \sum_{j=1}^{t} \Big(\prod_{s=j}^{t} \frac{\partial \mathbf{h}_s^i}{\partial \mathbf{h}_{s-1}^i} \Big) \frac{\partial \mathbf{h}_{j-1}^i}{\partial \mathbf{U}} \tag{2.3}$$

   It implies that the gradient back-propagated from time-step T to time-step $m < T$, is dominated by the product of the partials of the hidden-state vectors $\frac{\partial \mathbf{h}_s}{\partial \mathbf{h}_{s-1}}$. Many researchers (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013b) observed that such a product could be unstable. Specifically, the product term may not stay close to the identity matrix and could explode or vanish. In the vanishing gradients case, hidden states corresponding to $t \ll T$ will have little contribution to the overall error. If the gradients explode, later states may have little contribution. This phenomenon is referred to as the *Vanishing/Exploding Gradients*. It would result in poor performance on tasks that require long-term dependency (LTD). RNN will not be able to learn correct credit assignments during training and hence, will be unable to learn the correlation between the task and input time steps that are important for solving the task. For instance, Figure 2·3 shows that many popular RNN architectures suffer from vanishing/exploding gradients. As a result, they either fail to minimize the training objective or take forever to find the minimum of the objective function (i.e., have a poor convergence rate).

2. **Noise Amplification.** Although clever transition function f could help mitigate the poor gradient issues, an RNN could still suffer from accumulating spurious information from the noise present in the sequential input if it does not have a mechanism to weed out the noisy time steps and focus more on the informative

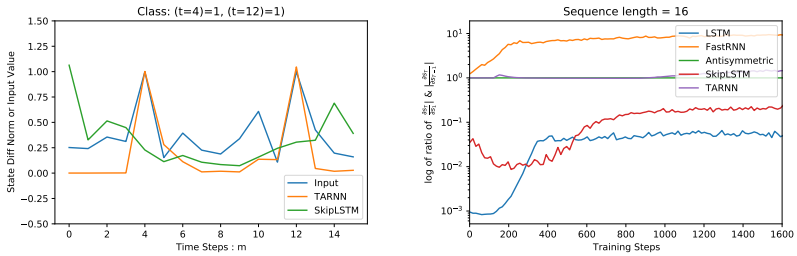


(a) **(b)**

Figure 2·3: Vanishing/Exploding Gradients in RNNs. Experiments on the addition task A.2.10 : (a) Convergence Rate Plot (loss at each training iteration); (b) Ratio $\|\frac{\partial h_T}{\partial h_1}\|/\|\frac{\partial h_T}{\partial h_{T-1}}\|$ illustrates Vanishing/Exploding gradient.

segments. We refer to this issue as the noise amplification problem. It could be very severe for very long sequences exhibiting long-term dependencies. We show this in an illustrative example in Figure 2·4. The input sequence contains signals only at two timesteps $t = 4$ and $t = 12$. Everywhere else, it only contains noise drawn uniformly from $[0, 1]$. Consider the AntisymmetricRNN in Figure 2·4b. It shows that even if an architecture achieves near identity gradients during back-propagation, it may suffer from the noise amplification issue and achieve poor performance. In contrast, our proposed architecture ((Kag and Saligrama, 2021a)) learns to effectively ponder only on the informative signals and ignore noisy input segments. Thus, it is very effective in solving this task.

3. **BPTT Complexity.** RNN training via BPTT is notoriously hard for RNNs with a poor transition function design, as illustrated earlier in vanishing/exploding gradients. In addition, the gradient computation is expensive for large T. In addition, this unrolling in time requires the algorithm to store all the intermediate hidden states as well as the partials for back-propagating the gradients. Thus, BPTT incurs heavy computational and working memory overhead.



Algorithm	Accuracy
Random Guess	25
FastRNN	45
LSTM	45
Antisymmetric	37
SkipLSTM	60
TARNN (Ours)	**100**

(a) **(b)** **(c)**

Figure 2·4: Noise Amplification Example. Example illustrates importance of mitigating gradient explosion/decay as well as ignoring noisy observations. Table lists test performance of baselines focused on improving RNN training. Fig. (a) plots the noisy input, and sequential changes in hidden state norms for SkipLSTM(Campos et al., 2018) and proposed TARNN(Kag and Saligrama, 2021a). Only ours responds to informative locations. Fig. (b) plots the norm of partials of hidden states. Only AntisymmetricRNN(Chang et al., 2019) and ours TARNN exhibit near identity gradients. However, only ours is effective as seen from the table. As such we infer TARNN (a) realizes near identity gradients for partials of hidden states, thus mitigating gradient explosion/decay, (b) zooms in on informative inputs and ignores noisy observations, and (c) By jointly ensuring (a) and (b), it improves RNN trainability, providing good generalization.

2.3 Related Works

2.3.1 RNN Architectures

Gated Architectures. Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997b) is widely used in RNNs to model long-term dependency in sequential data. Gated recurrent unit (GRU) (Cho et al., 2014a) is another gating mechanism designed to achieve a similar performance of LSTM architecture with fewer parameters. Some recent gated RNNs include UGRNN (Collins et al., 2017) and FastGRNN (Kusupati et al., 2018). While these architectures mitigate the vanishing/exploding gradient issues, they do not eliminate them. Often, these models incur increased inference, training costs, and model size.

Unitary RNNs. (Arjovsky et al., 2016; Jing et al., 2017; Zhang et al., 2018a; Mhammedi et al., 2017) focus on designing well-conditioned state transition matrices.

They attempt to enforce unitary property during training. Note that the unitary property does not generally circumvent vanishing gradient ((Pennington et al., 2017)). Also, it limits expressive power and prediction accuracy while also increasing training time. (Lezcano-Casado and Martínez-Rubio, 2019) alleviates some of these issues by leveraging Riemannian geometry and Lie group theory. (Kerg et al., 2019) uses Schur decomposition of the hidden-to-hidden transition matrix and maintains two separate components (normal and non-normal components) during training.

Deep RNNs. These are nonlinear transition functions incorporated into RNNs for performance improvement. For instance, (Pascanu et al., 2013a) empirically analyzed the problem of how to construct deep RNNs. (Zilly et al., 2017) proposed extending the LSTM architecture to allow step-to-step transition depths larger than one. (Mujika et al., 2017) proposed incorporating the strengths of both multiscale RNNs and deep transition RNNs to learn complex transition functions. While Deep RNNs offer richer representations relative to single-layered models, they are complementary to our proposed architectures.

Residual/Skip Connections. (Jaeger et al., 2007; Bengio et al., 2013; Chang et al., 2017; Campos et al., 2018; Kusupati et al., 2018) feed-forward state vectors to induce skip or residual connections, to serve as a middle ground between feed-forward and recurrent models, and to mitigate gradient decay. Nevertheless, these connections cannot entirely eliminate gradient explosion/decay. For instance, (Kusupati et al., 2018) suggest $\mathbf{h}_t = \alpha_t \mathbf{h}_{t-1} + \beta_t \phi(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$, and learn parameters so that $\alpha_t \approx 1$ and $\beta_t \approx 0$. Evidently, this setting can lead to identity gradients. But observe that setting $\beta_t \approx 0$ implies little contribution from the inputs and can conflict with good accuracy, as also observed in our experiments.

Linear RNNs. Many attempts improve the computational cost of the sequential models by introducing lighter recurrent connections. (Lei et al., 2018; Balduzzi and

Ghifary, 2016) replace the hidden-to-hidden interactions in the LSTMs with linear connections in the hope of parallelism. (Bradbury et al., 2016) performs similar linear interactions along with the increased receptive field from the inputs, i.e., instead of just using the current observation, it uses the previous few inputs to compensate for the lost non-linear interaction between the hidden states. Similar to these works, (Li et al., 2018b; Li et al., 2019b) introduce linear connection in vanilla RNNs and compensate for the loss of performance by allowing various architectures on top of this light-recurrent unit. These architectures include stacked encoders, residual, and dense connections between multiple layers. It should be noted that although light, the loss of non-linear interaction does result in a significant setback, and as a result, these works have to rely on more than one RNN layer to gain anything reasonable in comparison to traditional variants. It reduces training time but results in significantly increased model size. For example, (Lei et al., 2018) requires twice the number of cells for LSTM-level performance. These multi-layered models will be prohibitive for IoT devices as inference time would be larger than vanilla RNNs. Besides, we can extend our work by allowing only linear connections and applying their orthogonal ideas for better parallelism and computational speed.

ODE/Dynamical Perspective. (Rosenblatt, 1962; Funahashi and Nakamura, 1993) inspired many RNNs with transition functions leveraging ordinary differential equations (ODEs). Few ODE-inspired architectures attempt to address stability but do not end up eliminating vanishing/exploding gradients. (Talathi and Vartak, 2015) proposed a modified weight initialization strategy based on a dynamical system perspective on the weight initialization process to successfully train RNNs composed of ReLUs. (Niu et al., 2019) analyzed RNN architectures using numerical methods of ODE and propose a family of ODE-RNNs. (Chang et al., 2019), propose Antisymmetric-RNN. Their key idea is to express the transition matrix in Eq. 3.1,

for the special case $\alpha = 0, \tau = 1$, as a difference: $\mathbf{U} = V - V^T$ and note that the eigenspectrum is imaginary. Nevertheless, Euler discretization, in this context, leads to instability, necessitating damping of the system. As such, vanishing gradients cannot be completely eliminated. Its behavior is analogous to FastRNN (Kusupati et al., 2018), where the identity gradients conflict with high accuracy. In summary, we are the first to propose evolution over the equilibrium manifold and demonstrate identity gradients.

Neural ODEs (Chen et al., 2018; Rubanova et al., 2019) have also been proposed for time-series prediction to deal with irregularly sampled inputs. They parameterize the derivative of the hidden state in terms of an *autonomous* differential equation and let the ODE evolve in continuous time until the next input arrives. As such, this is not our goal, our ODE explicitly depends on the input and evolves until equilibrium for that input is reached. We introduce incremental updates to bypass vanishing/exploding gradient issues, which is not of specific concern for these works.

(Erichson et al., 2021) proposed Lipschitz RNN where the transition function is written as a combination of a linear transformation of the hidden state and a Lipschitz non-linear function. They study global stability properties of such a dynamical system and propose continuous-time Lipschitz recurrent units as a discretization using Runge-Kutta scheme (RK2) [1]. (Rusch and Mishra, 2021a) developed coRNN that discretizes a system of second-order ODE, modeling controlled non-linear coupled damped oscillators. Similarly, (Rusch and Mishra, 2021b) designed UnICORNN, a recurrent unit that uses a Hamiltonian system of second-order ODEs.

Conditional Computation and Attention. Our pondering perspective can be viewed as a form of conditional computation in time. Nevertheless, much of the conditional computation work aims to gradually scale model capacity without suffering proportional increases in computational (inference) cost (see (Graves, 2016; Chung

[1] https://en.wikipedia.org/wiki/Runge-Kutta_methods

et al., 2016; Yu et al., 2017; Jernite et al., 2017; Hansen et al., 2019)). Different from these works, our focus is on improving RNN trainability by suppressing noisy observations, so that long-term dependencies can be handled by ignoring uninformative input segments. Within this context, only (Campos et al., 2018) is closely related to our viewpoint. Like us, (Campos et al., 2018) also proposes to skip input segments to improve RNN training, but unlike us, since their state-transition designs are conventional, they still suffer vanishing and exploding gradients on segments that are not skipped, and as a result suffer performance degradation on benchmark datasets. Also, as (Campos et al., 2018) points out, our work can also be viewed as a temporal version of hard attention mechanisms for selecting image regions. These works (see (Campos et al., 2018)) that deal with visually-based sequential tasks, have high model complexity and are difficult to train on long input sequences. Others (Vaswani et al., 2017) leverage attention to bypass RNNs. In contrast, we offer an approach that is lightweight and improves RNN trainability on long sequences.

Structured State Spaces. Recently, researchers have explored structured state spaces to model long-range dependencies in sequential data. (Gu et al., 2021) investigated linear state-space layers that transform the input sequence using a linear continuous time state-space representation. Unfortunately, their memory and computational requirements render them infeasible for many problems of practical importance. (Gu et al., 2022) improve the linear state-space model and propose new parameterization of the transition function (sum of low-rank and skew-symmetric matrices).

2.3.2 RNN Training Algorithms

While a number of RNN training methods have been proposed, BPTT remains the single most dominant method (Rumelhart et al., 1986; Werbos, 1990). BPTT unrolls

the recurrent logic for the entire time horizon and computes gradient through this horizon. It has been observed to be computationally expensive (storing the hidden states and computing gradient for entire time horizon) and unless the RNN architecture is carefully designed, this leads to vanishing / exploding gradients (Hochreiter, 1991; Bengio et al., 2013). Truncated BPTT (Williams and Peng, 1990) is the variant of BPTT where gradient flow is truncated after a fixed number of timesteps. This fails to learn dependencies present beyond the fixed window.

Real time recurrent learning (RTRL) (Williams and Zipser, 1989), a BPTT alternative, proposes to propagate the partials $\frac{\partial h_t}{\partial h_{t-1}}$ and $\frac{\partial h_t}{\partial W}$ from timestep t to $t+1$, by noting that there is significant overlap in the product term (see Eq. 2.3) from time t to $t+1$, allowing for recursive computation. Early attempts suffered large memory overhead limiting its usage, and while recent attempts (Mujika et al., 2018; Menick et al., 2021; Tallec and Ollivier, 2018; Ollivier and Charpiat, 2015) have been more successful, these methods still fall short of BPTT performance, and so trainability of RNNs is still a significant issue.

This allows the computation of the gradient at the final timestep recursively. The initial attempts at maintaining the dynamics to propagate gradients introduced large overheads. Thus limiting their usage to only small networks. There have been many recent attempts to revive RTRL (Mujika et al., 2018; Menick et al., 2021; Tallec and Ollivier, 2018; Ollivier and Charpiat, 2015). While being low on memory footprint and fast, these attempts have still been lagging behind the performance of BPTT.

2.3.3 Miscellaneous

(Miller and Hardt, 2019) analyzes LSTMs from the stability perspective and show that in an unconstrained form LSTMs are unstable. They show that under some conditions on the non-linearity in transition, stable recurrent neural networks can be

represented by a feed-forward network. While stability reasons about the exploding gradients, it does not eliminate the vanishing gradients issue during training. (Hardt et al., 2018) analyze a linear time-invariant dynamical system using a sequence of noisy system generated observations and prove that stochastic gradient descent efficiently converges to the global optimizer of the maximum likelihood objective. (Linsley et al., 2020) addresses the large memory cost of BPTT which scales linearly with the number of time steps. They modify the training process by replacing the BPTT algorithm with Recurrent BackProp (RBP) algorithm which optimizes the parameters to achieve steady state dynamics.

(Kuznetsov and Mohri, 2015) study non-stationary non-mixing stochastic process, specifically a time series forecasting problem. They present data-dependent measure of sequential complexity to be estimated from data under mild assumptions. (Kuznetsov and Mohri, 2016) study time-series forecasting from online learning perspective. They prove generalization bounds for a hypothesis derived by online-to-batch conversion in a non-stationary non-mixing stochastic processes. These works only analyze this connection from the theoretical perspective and do not provide any algorithmic insights into training RNNs, which has been the focus in this chapter.

Chapter 3

Incremental Recurrent Neural Networks (iRNNs)

3.1 Introduction

(**Rosenblatt, 1962**), on whose work we draw inspiration from, introduced continuous-time RNN (CTRNN) to mimic activation propagation in neural circuitry. CTRNN dynamics evolves as follows:

$$\tau \dot{g}(t) = -\alpha g(t) + \phi(Ug(t) + Wx(t) + b),\ t \geq t_0. \tag{3.1}$$

Here, $x(t) \in \mathbb{R}^d$ is the input signal, $g(t) \in \mathbb{R}^D$ is the hidden state vector of D neurons, $\dot{g}_i(t)$ is the rate of change of the i-th state component; $\tau, \alpha \in \mathbb{R}^+$, referred to as the post-synaptic time-constant, impacts the rate of a neuron's response to the instantaneous activation $\phi(Ug(t) + Wx(t) + b)$; and $U \in \mathbb{R}^{D\times D}$, $W \in \mathbb{R}^{D\times d}$, $b \in \mathbb{R}^D$ are model parameters. In passing, note that recent RNN works that draw inspiration from ODE's (Chang et al., 2019) are special cases of CTRNN ($\tau = 1$, $\alpha = 0$).

Vanishing Gradients. The qualitative aspects of the CTRNN dynamics is transparent in its integral form:

$$g(t) = e^{-\alpha\frac{t-t_0}{\tau}} g(t_0) + \frac{1}{\tau}\int_{t_0}^{t} e^{-\alpha\frac{t-s}{\tau}} \phi(Ug(s) + Wx(s) + b)ds \tag{3.2}$$

This integral form reveals that the partials of hidden-state vector with respect to the initial condition, $\frac{\partial g(t)}{\partial g(t_0)}$, gets attenuated rapidly (first term in RHS), and so we face a vanishing gradient problem. We will address this issue later but we note that this is not an artifact of CTRNN but is exhibited by ODEs that have motivated other RNNs (see Sec. 2.3.1).

Shannon-Nyquist Sampling. A key property of CTRNN is that the time-constant τ together with the first term $-g(t)$, is in effect a low-pass filter with bandwidth $\alpha\tau^{-1}$ suppressing high frequency components of the activation signal, $\phi((Ug(s)) + (Wx(s)) + b)$. This is good, because, by virtue of the Shannon-Nyquist sampling theorem, we can now *maintain fidelity* of discrete samples with respect to continuous time dynamics, in contrast to conventional ODEs ($\alpha = 0$). Additionally, since high-frequencies are already suppressed, in effect we may assume that the input signal $x(t)$ is slowly varying relative to the post-synaptic time constant τ.

Equilibrium. The combination of low pass filtering and slowly time varying input has a significant bearing. The state vector as well as the discrete samples evolve close to the equilibrium state, i.e., $g(t) \approx \phi(Ug(t) + Wx(t) + b)$ under general conditions (Sec. 3.2).

Incremental Updates. Whether or not system is in equilibrium, the integral form in Eq. 3.2 points to gradient attenuation as a fundamental issue. To overcome this situation, we store and process increments rather than the cumulative values $g(t)$ and propose dynamic evolution in terms of increments. Let us denote hidden state sequence as $h_m \in \mathbb{R}^D$ and input sequence $x_m \in \mathbb{R}^d$. For $m = 1, 2, \ldots, T$, and a

suitable $\beta > 0$

$$\tau \dot{g}(t) = -\alpha(g(t) \pm h_{m-1}) + \phi(U(g(t) \pm h_{m-1}) + Wx_m + b),\ g(0) = 0,\ t \geq 0 \tag{3.3}$$

$$h_m \triangleq h_m^{\beta \cdot \tau} \triangleq g(\beta \cdot \tau)$$

Intuitively, say system is in equilibrium and $-\alpha(\mu(x_m, h_{m-1})) + \phi(U\mu(x_m, h_{m-1}) + Wx_m + b) = 0$. We note state transitions are marginal changes from previous states, namely, $h_m = \mu(x_m, h_{m-1}) - h_{m-1}$. Now for a fixed input x_m, as to which equilibrium is reached depends on h_{m-1}, but are nevertheless finitely many. So encoding marginal changes as states leads to "identity" gradient.

Incremental RNN (iRNN) achieves Identity Gradient. We propose to discretize Eq. 3.3 to realize iRNN (see Sec. 3.2). At time m, it takes the previous state $h_{m-1} \in \mathbb{R}^D$ and input $x_m \in \mathbb{R}^d$ and outputs $h_m \in \mathbb{R}^D$ after simulating the CTRNN evolution in discrete-time, for a suitable number of discrete steps. We show that the proposed RNN approximates the continuous dynamics and solves the vanishing/exploding gradient issue by ensuring identity gradient. In general, we consider two options, SiRNN, whose state is updated with a single CTRNN sample, similar to vanilla RNNs, and, iRNN, with many intermediate samples. SiRNN is well-suited for slowly varying inputs.

Contributions. To summarize, we list our main contributions:

(A) iRNN converges to equilibrium for typical activation functions. The partial gradients of hidden-state vectors for iRNNs converge to identity, thus solving vanishing/exploding gradient problem!

(B) iRNN converges rapidly, at an exponential rate in the number of discrete samplings of Eq. 3.1. SiRNN, the single-step iRNN, is efficient and can be leveraged for slowly varying input sequences. It exhibits fast training time, has fewer parameters

and better accuracy relative to standard LSTMs.

(C) Extensive experiments on LTD datasets show that we improve upon standard LSTM accuracy as well as other recent proposals that are based on designing transition matrices and/or skip connections. iRNNs/SiRNNs are robust to time-series distortions such as noise paddings

(D) While our method extends directly (see Appendix B.1) to Deep RNNs, we deem these extensions complementary, and focus on single-layer to highlight our incremental perspective.

3.2 Method

We use Euler's method to discretize Eq. 3.3 in steps $\delta = \eta\tau$. Denoting the kth step as $g_k = g(k\delta)$

$$\tau\frac{g_k - g_{k-1}}{\delta} = -\alpha(g_{k-1} + h_{m-1}) + \phi(U(g_{k-1} + h_{m-1}) + Wx_m + b),\ k \in [K] \tag{3.4}$$

Rearranging terms we get a compact form for iRNN (see Fig. 3·1). In addition we introduce a learnable parameter η_m^k and let it be a function of time m and the recursion-step k.

$$\begin{aligned} g_k &= g_{k-1} + \eta_m^k(\phi(U(g_{k-1} + h_{m-1}) + Wx_m + b) - \alpha(g_{k-1} + h_{m-1})),\ k \in [K] \\ h_m^K &= g_K \end{aligned} \tag{3.5}$$

We run the recursion for $k \in [K]$ with some suitable initial condition. This could be $g_0 = 0$ or initialized to the previous state, i.e., $g_0 = h_{m-1}$ at time m.

In many of our examples, we find the input sequence is slowly varying, and $K = 1$ can also realize good empirical performance. We refer to this as single-

step-incremental-RNN (SiRNN).

$$h_m^1 = g_0 + \eta_m(\phi(U(g_0 + h_{m-1}) + Wx_m + b) - \alpha(h_{m-1} + g_0)) \tag{3.6}$$

For both iRNN and SiRNN we drop the superscript whenever it is clear from the context.

Root Finding and Transitions. The two indices k and m should not be confused. The index $m \in [T]$ refers to the time index, and indexes input, x_m and hidden state h_m over time horizon T. The index $k \in [K]$ is a fixed-point recursion for converging to the equilibrium solution at each time m, given input x_m and the hidden state h_{m-1}. We iterate over k so that at $k = K$, g_K satisfies,

$$\phi(U(g_K + h_{m-1}) + Wx_m + b) - \alpha(g_K + h_{m-1}) \approx 0$$

The recursion (Eq. 3.5) at time m runs for K rounds, terminates, and recursion is reset for the new input, x_{m+1}. Indeed, Eq. 3.5 is a standard root-finding recursion, with g_{k-1} serving as the previous solution, plus a correction term, which is the error, $\phi(U(g_{k-1} + h_{m-1}) + Wx_m + b) - \alpha(g_{k-1} + h_{m-1})$. If the sequence converges, the resulting solution is the equilibrium point. Proposition 2 guarantees a geometric rate of convergence.

Identity Gradient. We will informally (see Theorem 1) show here that partial gradients are identity. Say we have for sufficiently large K, $h_m = g_K$ is the equilibrium solution. It follows that,

$$\phi(U(h_m + h_{m-1}) + Wx_m + b) - \alpha(h_m + h_{m-1})) = 0$$

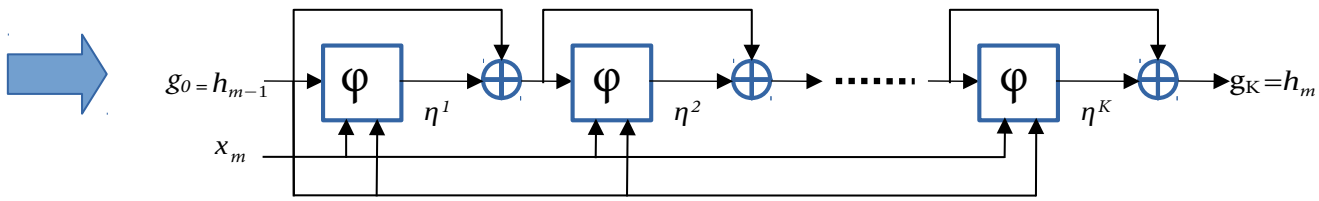


Figure 3·1: iRNN depicted by unfolding into K recursions for one transition from $g_0 = h_{m-1}$ to $h_m = g_K$. Here, $\varphi(x, g, h) = \phi(U(g+h)+Wx+b)-\alpha(g+h)$. See Sec. B.2 for implementation and pseudo-code. This resembles (Graves, 2016), who propose to vary K with m as a way to attend to important input transitions. However, the transition functions used are gated units, unlike our conventional ungated functions. As such, while this is not their concern, equilibrium may not even exist and identity gradients are not guaranteed in their setup.

Taking derivatives, we have,

$$\nabla\phi(\cdot)U\left(\frac{\partial h_m}{\partial h_{m-1}}+I\right)-\alpha\left(\frac{\partial h_m}{\partial h_{m-1}}+I\right)=0 \implies (\nabla\phi(\cdot)U-\alpha I)\left(\frac{\partial h_m}{\partial h_{m-1}}+I\right)=0. \tag{3.7}$$

Thus if the matrix $(\nabla\phi(\cdot)U - \alpha I)$ is not singular, it follows that $(\frac{\partial h_m}{\partial h_{m-1}} + I) = 0$.

SiRNN vs. iRNN. SiRNN approximates iRNN. In particular, say x_m is a constant in the segment, $m \in [m_0, m_0 + K]$, then SiRNN trajectory of hidden states, denoted as $h^1_{m_0+K}$ is equal to the iRNN hidden state $h^K_{m_0}$, when both SiRNN and iRNN are initialized with $g_0 = h_{m-1}$. Thus, for slowly time-varying inputs we can expect SiRNN to closely approximate iRNN.

Residual Connections vs. iRNN/SiRNN. As such, our architecture is a special case of skip/residual connections. Nevertheless, unlike skip connections, our connections are *structured*, and the dynamics driven by the error term ensures that the hidden state is associated with equilibrium and leads to identity gradient. No such guarantees are possible with unstructured skip connections. Note that for slowly varying inputs, after a certain transition-time period, we should expect SiRNN to be close to equilibrium as well. Without this imposed structure, general residual architectures can learn patterns that can be dramatically different (see Fig. 3·2).

3.2.1 Identity Gradient Property and Convergence Guarantees.

Let us now collect a few properties of Eq. 3.3 and Eq. 3.5. First, denote the equilibrium solutions for an arbirary input $x \in \mathbb{R}^d$, arbitrary state-vector $\nu \in \mathbb{R}^D$, in an arbitrary round:

$$\mathcal{M}_{eq}(x, \nu) = \{\mu \in \mathbb{R}^D \mid \alpha(\mu + \nu) = \phi(U(\mu + \nu) + Wx + b)\}$$

Whenever the equilibrium set is a singleton, we denote it as a function $h_{eq}(x, \nu)$. For simplicity, we assume below that η_k^i is a positive constant independent of k and i.

Proposition 1. *Suppose, $\phi(\cdot)$ is a 1-Lipshitz function in the norm induced by $\|\cdot\|$, and $\|U\| < \alpha$, then for any $x_m \in \mathbb{R}^d$ and $h_{m-1} \in \mathbb{R}^D$, it follows that $\mathcal{M}_{eq}(x, \nu)$ is a singleton and as $K \to \infty$, the iRNN recursions converge to this solution, namely, $h_m = \lim_{K\to\infty} g_K = h_{eq}(x_m, h_{m-1})$*

Proof. Define $T : \mathbb{R}^D \to \mathbb{R}^d$, with $T(g) = (1 - \eta\alpha)g + \eta(\phi(U(g + h_{m-1}) + Wx_m + b) - h_{m-1})$. It follows that $T(\cdot)$ is a contraction:

$$\begin{aligned}
\|T(g) - T(g')\| \leq & (1 - \eta\alpha)\|g - g'\| \\
& + \eta\|\phi(U(g + h_{m-1}) + Wx_m + b) - \phi(U(g' + h_{m-1}) + Wx_m + b)\| \\
\leq & (1 - \eta\alpha + \|U\|\eta)\|g - g'\| < \|g - g'\|.
\end{aligned}$$

We now invoke the Banach fixed point theorem, which asserts that a contractive operator on a complete metric space converges to a unique fixed point, namely, $T^K(g) \to g_*$. Upon substitution, we see that this point g_* must be such that, $\phi(U(g_* + h_{m-1}) + Wx_m + b) - (g_* + h_{m-1}) = 0$. Thus equilibrium point exists and is unique. Result follows by setting $h_m \triangleq h_{eq}(x_m, h_{m-1})$. □

Handling $\|U\| \leq \alpha$. In experiments, we set $\alpha = 1$, and do not enforce $\|U\| \leq \alpha$ constraint. Instead, we initialize U as a Gaussian matrix with IID mean zero, small

variance components. As such, the matrix norm is smaller than 1. Evidently, the resulting learnt U matrix does not violate this condition.

Next we show for $\eta > 0$, iRNN converges at a linear rate, which follows directly from Proposition 1.

Proposition 2. *Under the setup in Proposition 1, it follows that,*

$$\|h_m^K - h_{eq}(x_m, h_{m-1})\| \triangleq \|g_K - h_{eq}(x_m, h_{m-1})\| \leq (1-\alpha\eta+\eta\|U\|)^K \|g_1 - h_{eq}(x_m, h_{m-1})\|$$

Remark. Proposition 1 accounts for typical activation functions ReLU, tanh, sigmoids as well as deep RNNs (appendix B.1).

In passing we point out that, in our experiments, we learn parameters η_m^k, and a result that accounts for this case is desirable. We describe this case in Appendix B.3. A fundamental result we describe below is that partials of hidden-state vectors, on

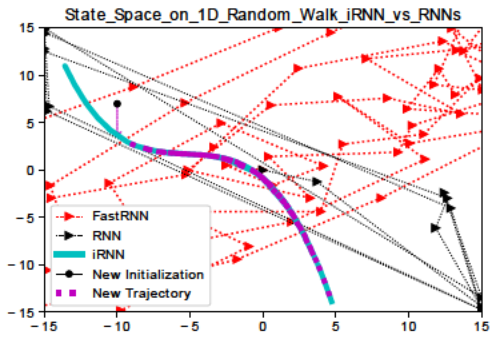


Figure 3·2: Phase-space trajectory with tanh activation of RNN, FastRNN, iRNN. X-axis denotes 1st dimension, and Y-axis 2nd dimension of 2D hidden state subject to random walk input with variance 10 for 1000 time-steps. Parameters U, W, b are randomly initialized. RNN states are scaled to fit plot since FastRNN is not required to be in the cube.

the equilibrium surface is unity. For technical simplicity, we assume a continuously differentiable activation, which appears to exclude ReLU activations. Nevertheless, we can overcome this issue, but requires more technical arguments. The main difficulty

stems from ensuring that derivatives along the equilibrium surface exist, and this can be realized by invoking the implicit function theorem (IFT). IFT requires continuous differentiability, which ReLUs violate. Nevertheless, recent results [1] suggests that one can state implicit function theorem for everywhere differentiable functions, which includes ReLUs.

Theorem 1. *Suppose $\phi(\cdot)$ is a continuously differentiable, 1-Lipshitz function, with $\|U\| < \alpha$. Then as $K \to \infty$, $\frac{\partial h_m}{\partial h_{m-1}} \to \frac{\partial h_{eq}(x_m, h_{m-1})}{\partial h_{m-1}} = -I$. Furthermore, as $K \to \infty$ the partial gradients over arbitrary number of rounds for iRNN is identity.*

$$\frac{\partial h_r}{\partial h_s} = \prod_{r \geq m > s} \frac{\partial h_m}{\partial h_{m-1}} = (-1)^{r-s}\mathbf{I} \Rightarrow \left\| \frac{\partial h_r}{\partial h_s} \right\| = 1. \tag{3.8}$$

Proof. Define, $\psi(g, h_{m-1}) = \phi(U(g + h_{m-1}) + Wx_m + b) - \alpha(g + h_{m-1})$. We overload notation and view the equilibrium point as a function of h_{m-1}, i.e., $g_*(h_{m-1}) = h_{eq}(x_m, h_{m-1})$. Invoking standard results[2] in ODE's, it follows that $g_*(h_{m-1})$ is a smooth function, so long as the Jacobian, $\nabla_g \psi(g_*, h_{m-1})$ with respect to the first coordinate, g_*, is non-singular. Upon computation, we see that, $\nabla_g \psi(g_*, h_{m-1}) = \nabla\phi(g_*, h_{m-1})U - \alpha I$, is non-singular, since $\|\nabla\phi(g_*, h_{m-1})U\| \leq \|U\|$. It follows that we can take partials of the state-vectors. By taking the partial derivatives *w.r.t.* h_{m-1} in Eq. 3.5, at the equilibrium points we have $[\nabla\phi(g_*, h_{m-1})U - \alpha\mathbf{I}][\frac{\partial g_*}{\partial h_{m-1}} + \mathbf{I}] = \mathbf{0}$ (see Eq. 3.7). The rest of the proof follows by observing that the first term is non-singular. □

Remark. We notice that replacing h_{m-1} with $-h_{m-1}$ in Eq. B.3 will lead to $\frac{\partial h_{eq}}{\partial h_{m-1}} = \mathbf{I}$, which also has no impact on magnitudes of gradients. As a result, both

[1] https://terrytao.wordpress.com/2011/09/12/the-inverse-function-theorem-for-everywhere-differentiable-maps/

[2] http://cosweb1.fau.edu/~jmirelesjames/ODE_course/lectureNotes_shortVersion_day1.pdf

choices are suitable for circumventing vanishing or exploding gradients during training, but still may converge to different local minima and thus result in different test-time performance. Furthermore, notice that the norm preserving property is somewhat insensitive to choices of α, so long as the non-singular condition is satisfied.

3.2.2 iRNN Design Implications: Low-Rank Model Parametrization

Fig. 3·2 depicts phase portrait and illustrates salient differences between RNN, FastRNN (RNN with skip connection), and iRNN (K=5). RNN and FastRNN exhibit complex trajectories, while iRNN trajectory is smooth, projecting initial point (black circle) onto the equilibrium surface (blue) and moving within it (green). This suggests that iRNN trajectory belongs to a low-dimensional manifold.

Variation of Equilibrium *w.r.t.* Input. As before, h_{eq} be an equilibrium solution for some tuple (h_{m-1}, x_m). It follows that,

$$(\alpha\mathbf{I} - \nabla\phi(U(h_{eq} + \mathbf{h}_{m-1}) + Wx_m + b)U)\partial h_{eq} = \nabla\phi(U(h_{eq} + h_{m-1}) + Wx_m + b)W\partial x_m$$

This suggests that, whenever the input undergoes a slow variation, we expect that the equilibrium point moves in such a way that $U\partial h_{eq}$ must lie in a transformed span of W. Now $W \in \mathbb{R}^{D\times d}$ with $d \ll D$, which implies that $(\alpha\mathbf{I} - \nabla\phi(U(h_{eq} + h_{m-1}) + Wx_m + b)U$ is rank-deficient.

Low Rank Matrix Parameterization. For typical activation functions, note that whenever the argument is in the unsaturated regime, $\nabla\phi(\cdot) \approx \mathbf{I}$. We then approximately get $\text{span}(\alpha\mathbf{I} - U) \approx \text{span}(W)$. We can express these constraints as $U = \alpha\mathbf{I} + VH$ with low-rank matrices $V \in \mathbb{R}^{D\times d_1}, H \in \mathbb{R}^{d_1\times D}$, and further map both Uh_m and Wx_m onto a shared space. Since in our experiments the signal vectors we encounter are low-dimensional, and sequential inputs vary slowly over time, we

enforce this restriction in all our experiments. In particular, we consider,

$$\phi\left(P[U(h_m + h_{m-1}) + Wx_m + b]\right) - (h_m + h_{m-1}) = \mathbf{0}. \tag{3.9}$$

The parameter matrix $P \in \mathbb{R}^{D\times D}$ maps the contributions from input and hidden states onto the same space. To decrease model-size we let $P = U = (\mathbf{I} + VH)$ learn these parameters.

3.3 Experiments

We organize this section as follows. First, the experimental setup, competing algorithms will be described. Then we present an ablative analysis to highlight salient aspects of iRNN and justify some of our experimental choices. We then plot and tabulate experimental results on benchmark datasets.

3.3.1 Experimental Setup and Baselines

Choice of Competing Methods. We choose competing methods based on the following criteria: (a) methods that are devoid of additional application or dataset-specific heuristics, (b) methods that leverage only single cell/block/layer, and (c) methods without the benefit of complementary add-ons (such as gating, advanced regularization, model compression, etc.). Requiring (a) is not controversial since our goal is methodological. Conditions (b),(c) are justifiable since we could also leverage these add-ons and are not germane to any particular method[3]. We benchmark iRNN against standard RNN, LSTM (Hochreiter and Schmidhuber, 1997b), (ungated) AntisymmetricRNN (Chang et al., 2019), (ungated) FastRNN (Kusupati et al., 2018).

Unitary RNN Variants. Results for methods based on unitary transitions (such

[3]These conditions eliminate some potential baselines. We provide specific justifications in the appendix B.4.

as (Arjovsky et al., 2016; Wisdom et al., 2016; Vorontsov et al., 2017; Zhang et al., 2018a)) are not reported in the main paper (when available reported in appendix) for the following reasons: (a) They are substantially more expensive, and requiring large model sizes; (b) Apart from the benchmark copy and add tasks, results tabulated by FastRNN and Antisymmetric authors (see (Zhang et al., 2018a; Chang et al., 2019)) show that they are well below SOTA; (c) iRNN dominates unitary-RNN variants on add-task (see Sec. 3.3.3); (d) On copy task, while unitary invariants are superior, (Vorontsov et al., 2017) attributes it to modReLU or leaky ReLU activations. Leaky ReLUs allow for linear transitions, and copy task being a memory task benefits from it. With hard non-linear activation, unitary RNN variants can take up to 1000's of epochs for even 100-length sequences ((Vorontsov et al., 2017)).

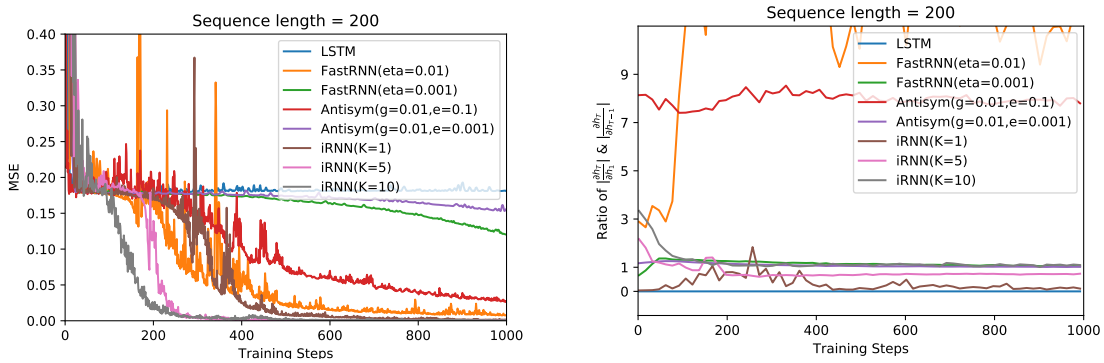
Implementation. For all our experiments, we used the parametrized update formulation in Eq. 3.9 for iRNN . We used tensorflow framework for our experiments. For most competing methods apart from AntisymmetricRNN, which we implemented, code is publicly available. All the experiments were run on an Nvidia GTX 1080 GPU with CUDA 9 and cuDNN 7.0 on a machine with Intel Xeon 2.60 GHz CPU with 20 cores.

Datasets. Pre-processing and feature extraction details for all publicly available datasets are in the appendix A.2. We replicate benchmark test/train split with 20% of training data for validation to tune hyperparameters. Reported results are based on the full training set, and performance achieved on the publicly available test set. Table A.1 (Appendix) and A.2 describes details for all the data sets.

Hyper Parameters. We used grid search and fine-grained validation wherever possible to set the hyper-parameters of each algorithm, or according to the settings published in (Kusupati et al., 2018; Arjovsky et al., 2016) (*e.g.* number of hidden states). Both the learning rate and η's were initialized to 10^{-2}. The batch size of 128

seems to work well across all the data sets. We used ReLU as the non-linearity and Adam ((Kingma and Ba, 2015)) as the optimizer for all the experiments.

3.3.2 Ablative Analysis



(a) **(b)**

Figure 3·3: Exploratory experiments for the Add task (a) Convergence with varying K; (b) Ratio $\|\frac{\partial h_T}{\partial h_1}\|/\|\frac{\partial h_T}{\partial h_{T-1}}\|$ illustrates Vanishing/Exploding gradient ($\|\frac{\partial h_T}{\partial h_{T-1}}\|$ and loss gradients are omitted but displayed in B.6.8. For iRNN (a) and (b) together show strong correlation of gradient with accuracy in contrast to other methods.

We perform ablative analysis on the benchmark add-task (Sec 3.3.3) for sequence length 200 for 1000 iterations and explore mean-squared error as a metric. Fig. 3·3 depicts salient results.

(a) Identity Gradients & Accuracy. iRNN accuracy is correlated with identity gradients. Increasing K improves gradients, and correlates with increased accuracy (Fig. 3·3). While other models $h_t = \alpha h_{t-1} + \beta\phi((U - \gamma I)h_{t-1} + Wx_t)$, can realize identity gradients for suitable choices; linear ($\alpha = 1, \beta = 1, \gamma = 0, U = 0$), FastRNN ($\alpha \approx 1, \beta \approx 0, \gamma = 0$) and Antisymmetric ($\alpha = 1, \beta = 1, U = V - V^T$, $\|U\| \leq \gamma$), this goal may not be correlated with improved test accuracy. FastRNN($\eta = 0.001$), Antisymmetric ($\gamma = 0.01, \epsilon = 0.001$) have good gradients but poorer test accuracy relative to FastRNN($\eta = 0.01$), Antisymmetric($\gamma = 0.01, \epsilon = 0.1$), with poorer gradients.

(b) Identity gradient implies faster convergence. Identity gradient, whenever effective, must be capable of assigning credit to the informative parts, which in turn results in larger loss gradients, and significantly faster convergence with number of iterations. This is borne out in figure 3·3(a). iRNN for larger K is closer to identity gradient with fewer (unstable) spikes ($K = 1$, 5, 10). With $K = 10$, iRNN converges within 300 iterations while competing methods take about twice this time (other baselines not included here exhibited poorer performance than the once plotted).

(c) SiRNN (iRNN with $K = 1$) delivers good performane in some cases. Fig. 3·3(a) illustrates that iRNN $K = \{5, 10\}$ achieves faster convergence than SiRNN, but the computational overhead per iteration roughly doubles or triples in comparison. SiRNN is faster relative to competitors. For this reason, we sometimes tabulate only SiRNN, whenever it is SOTA in benchmark experiments, since accuracy improves with K but requires higher overhead.

3.3.3 Long-term Dependency and Other Tasks

We list five types of datasets, all of which in some way require effective gradient propagation: (1) Conventional Benchmark LTD tasks (Add & Copy tasks) that illustrate that iRNN can rapidly learn long-term dependence; (2) Benchmark vision tasks (pixel MNIST, perm-MNIST) that may not require long-term, but nevertheless, demonstrates that iRNN achieves SOTA for short term dependencies but with less resources. (3) Noise Padded (LTD) Vision tasks (Noisy MNIST, Noisy CIFAR), where a large noise time segment separates information segments and the terminal state, and so the learner must extract information parts while rejecting the noisy parts; (4) short duration activity embedded in a larger time-window (HAR-2, Google-30 in Appendix Table A.1 and many others B.6), that usually arise in the context of smart IoT applications and require a small model-size footprint. (Chang et al., 2019)

further justify (3) and (4) as LTD, because for these datasets where only a smaller unknown segment(s) of a longer sequence is informative. (5) Sequence-sequence prediction tasks (PTB language modeling) that are different from terminal prediction (reported in appendix B.6).

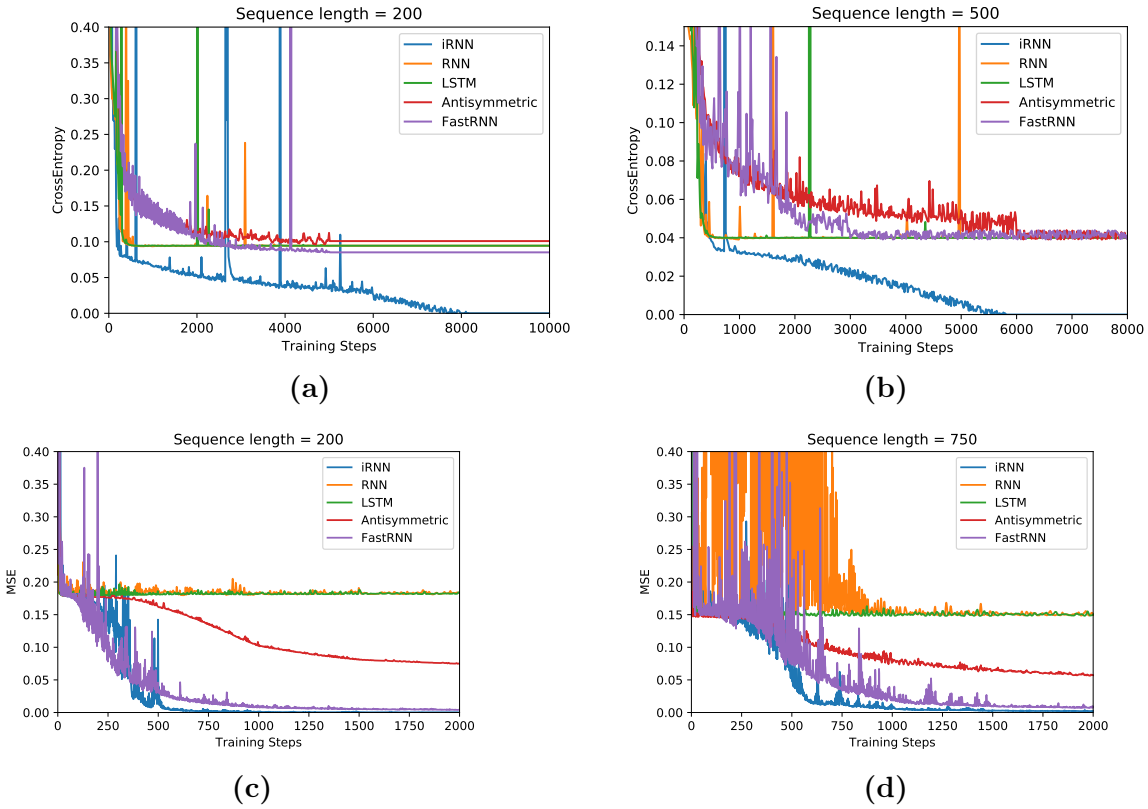


Figure 3·4: Following (Arjovsky et al., 2016) we display average Cross Entropy for the Copy Task (Sequence Length (with baseline memoryless strategy)): (a) 200 (0.09) (b) 500 (0.039). Mean Squared Error for the Add Task, baseline performance is 0.167 (Sequence Length) : (c) 200 (d) 750. For both tasks, iRNN runs $K = 5$.

Standard Benchmark LTD Tasks : Addition & Copy Memory

Addition and Copy tasks (Hochreiter and Schmidhuber, 1997b) have long been used as benchmarks in the literature to evaluate LTD (Hori et al., 2017; Zhang et al., 2018a; Arjovsky et al., 2016; Martens and Sutskever, 2011). We follow the setup described in

(Arjovsky et al., 2016) to create the adding and copying tasks. See appendix A.2.10 and A.2.11 for detailed description. For both tasks we run iRNN with $K = 5$.

Figure 4·2 show the average performance of various methods on these tasks. For the copying task we observe that iRNN converges rapidly to the naive baseline and is the only method to achieve zero average cross entropy. For the addition task, both FastRNN and iRNN solves the addition task but FastRNN takes twice the number of iterations to reach desired 0 MSE. [4] In both the tasks, iRNN performance is much more stable across number of online training samples. In contrast, other methods either takes a lot of samples to match iRNN 's performance or depict high variance in the evaluation metric. This shows that iRNN converges faster than the baselines (to the desired error). *These results demonstrate that iRNN easily and quickly learns the long term dependencies* . We omitted reporting unitary RNN variants for Add and Copy task. See Sec. 3.3.1 for copy task. On Add-task we point out that our performance is superior. In particular, for the longer $T = 750$ length, (Arjovsky et al., 2016), points out that MSE does not reach zero, and uRNN is noisy. Others either (Wisdom et al., 2016) do not report add-task or report only for shorter lengths (Zhang et al., 2018a).

Non LTD Vision Tasks: Pixel MNIST, Permute MNIST

Next, we perform experiments on the sequential vision tasks: (a) classification of MNIST images on a pixel-by-pixel sequence; (b) a fixed random permuted MNIST sequence (Lecun et al., 1998). These tasks typically do not fall in the LTD categories (Chang et al., 2019), but are useful to demonstrate faster training, which can be attributed to better gradients.

For the pixel-MNIST task, (Kusupati et al., 2018) reports that it takes significantly

[4] Note that LSTM solves the addition problem in (Arjovsky et al., 2016) only with more than $10k$ iterations. We only use $2k$ iterations in our experiments to demonstrate the effectiveness of our method.

Table 3.1: Results for Pixel-by-Pixel MNIST and Permuted MNIST datasets. K denotes pre-defined recursions embedded in graph to reach equilibrium.

Data set	Algorithm	Accuracy (%)	Train Time (hr)	#Params
Pixel-MNIST	FastRNN	96.44	15.10	33k
	RNN	94.10	45.56	14k
	LSTM	97.81	26.57	53k
	Antisymmetric	98.01	8.61	14k
	iRNN (K=1)	97.73	**2.83**	**4k**
	iRNN (K=3)	**98.13**	2.93	**4k**
Permute-MNIST	FastRNN	92.68	9.32	8.75k
	LSTM	92.61	19.31	35k
	Antisymmetric	93.59	4.75	14k
	iRNN (K=1)	**95.62**	**2.41**	**8k**

longer time for existing (LSTMs, Unitary, Gated, Spectral) RNNs to converge to reasonable performance. In contrast, FastRNN trains at least $2x$ faster than LSTMs. Our results (table 4.1) for iRNN shows a $9x$ speedup relative LSTMs, and $2x$ speedup in comparison to Antisymmetric. In terms of test accuracy, iRNN matches the performance of Antisymmetric, but with at least $3x$ fewer parameters. We did not gain much with increased K values[5]. For the permuted version of this task, we seem to outperform the existing baselines [6]. In both tasks, iRNN trained at least $2x$ faster than the strongest baselines. *These results demonstrate that iRNN converges much faster than the baselines with fewer parameters.*

Noise padding Tasks: Noisy-MNIST, Noisy-CIFAR

Additionally, as in (Chang et al., 2019), we *induce* LTD by padding CIFAR-10 with noise exactly replicating their setup, resulting in Noisy-CIFAR. We extend this setting to MNIST dataset resulting in Noisy-MNIST. Intuitively we expect our model to be

[5]For some existing comparisons LSTM have achieved roughly 98.9 with dataset specific heuristics (Cooijmans et al., 2017), but we could not achieve this performance in our comparison (and so have many others like (Kusupati et al., 2018; Zhang et al., 2018a; Arjovsky et al., 2016)).

[6]Note that there's no standard permutation in the literature. This may be the main reason we could not replicate (Chang et al., 2019) performance on the permute MNIST task.

Table 3.2: Results for Noise Padded CIFAR-10 and MNIST datasets. Since the equilibrium surface is smooth and resilient to small perturbations, iRNN achieves better performance than the baselines with faster convergence.

Data set	Algorithm	Accuracy (%)	Train Time (hr)	#Params
Noisy-MNIST	FastRNN	98.12	8.93	11k
	LSTM	10.31	19.43	44k
	Antisymmetric	97.76	5.21	10k
	iRNN (K=1)	**98.48**	**2.39**	**6k**
Noisy-CIFAR	FastRNN	45.76	11.61	16k
	LSTM	11.60	23.47	64k
	Antisymmetric	48.63	5.81	16k
	iRNN (K=1)	**54.50**	**2.47**	**11.5k**

resilient to such perturbations. We attribute iRNN's superior performance to the fact that it is capable of suppressing noise. For example, say noise is padded at $t > \tau$ and this results in Wx_t being zero on average. For iRNN the resulting states ceases to be updated. So iRNN recalls last informative state h_τ (modulo const) unlike RNNs/variants! Thus information from signal component is possibly better preserved.

Results for Noisy-MNIST and Noisy-CIFAR are shown in Table 3.2. Note that almost all timesteps contain noise in these datasets. LSTMs perform poorly on these tasks due to vanishing gradients. This is consistent with the earlier observations (Chang et al., 2019). iRNN outperforms the baselines very comprehensively on CIFAR-10, while on MNIST the gains are smaller, as it's a relatively easier task. *These results show that iRNN is more resilient to noise and can account for longer dependencies.*

Short Duration Embedded Activity Recognition Tasks: HAR-2, Google-30

We are interested in detecting activity embedded in a longer sequence with small footprint RNNs ((Kusupati et al., 2018)): (a) Google-30 (Warden, 2018), *i.e.* detec-

Table 3.3: Results for Activity Recognition Datasets. iRNN outperforms the baselines on all metrics even with $K = 1$. Its worth noticing that although $K = 5$ increases test time, it's well within LSTM's numbers, the overall train time and resulting performance are better than $K = 1$.

Data set	Algorithm	Accuracy (%)	Train Time (hr)	#Params	Test Time (ms)
HAR-2	FastRNN	94.50	0.063	7.5k	**0.01**
	RNN	91.31	0.114	7.5k	**0.01**
	LSTM	93.65	0.183	16k	0.04
	Antisymmetric	93.15	0.087	7.5k	**0.01**
	iRNN (K=1)	95.32	0.061	**4k**	**0.01**
	iRNN (K=5)	**96.30**	**0.018**	**4k**	0.03
Google-30	FastRNN	91.60	1.30	18k	**0.01**
	RNN	80.05	2.13	12k	**0.01**
	LSTM	90.31	2.63	41k	0.05
	Antisymmetric	90.91	0.54	12k	**0.01**
	iRNN (K=1)	93.77	**0.44**	**8.5k**	**0.01**
	iRNN (K=5)	**94.23**	**0.44**	**8.5k**	0.05

tion of utterances of 30 commands plus background noise and silence, and (b) HAR-2 (Anguita et al., 2012), *i.e.* Human Activity Recognition from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone.

Table 3.3 shows accuracy, training time, number of parameters and prediction time. Even with $K = 1$, we compare well against competing methods, and iRNN accuracy improves with larger K. Interestingly, higher K yields faster training as well as moderate prediction time, despite the overhead of additional recursions. *These results show that iRNN outperforms baselines on activity recognition tasks, and fits within IoT/edge-device budgets.*

3.4 Discussion

We eliminate the vanishing/exploding gradients by keeping track of the increments in the ODE-based state transition. While, in theory, solving the ODE up to equilibrium yields identity gradients during back-propagation, in practice, we utilize an iterative

solver with K number of discrete steps. In some sequential tasks, the equilibrium (i.e., high K values) may not be the optimal solution. We take this observation into account in the next chapter (with Time Adaptive RNN design), where we do not fix the parameterization to yield identity gradients. In contrast, we lift this problem with free parameters that can achieve loss-less information propagation whenever the task at hand desires. Additionally, iRNNs do not modify the time constants. As a result, they are prone to noise amplifications in the hidden state transitions.

Another point is that we use Euler discretization in this setup for simplicity. Later works(Erichson et al., 2021; Rusch and Mishra, 2021a) have resorted to advanced ODE solvers such as Range-Kutta solvers, LeapFrop integration, etc. These advanced solvers can be easily integrated into the ODE state transitions in iRNN hidden state updates.

Chapter 4

Time Adaptive Recurrent Neural Networks (TARNNs)

4.1 Introduction

While we also draw upon ODEs to propose solutions to improve vanilla RNN trainability, our proposal differs from existing works in fundamental ways. To build intuition, first consider the ODE, with $\lambda \in \mathbb{R}^+$, $\mathbf{U} \in \mathbb{R}^{D\times D}, \mathbf{W} \in \mathbb{R}^{D\times d}$, and $\mathbf{A} \in \mathbb{R}^{D\times D}$ Hurwitz stable (Khalil, 2002):

$$\lambda \dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m) \tag{4.1}$$

where, $\phi(\cdot)$ is the conventional non-linear RNN activation function such as a ReLU; This particular form, serving as an analogue[1] of vanilla RNNs, is quite old (Rosenblatt, 1962). In each round, m, we start from an initial state, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$, which corresponds to the current hidden state, and input, $\mathbf{x}_m$, and evolve the ODE for a unit period of time. Subsequently, the hidden state is updated by setting $\mathbf{s}_m = \mathbf{z}(t_0 + 1)$, and in this way, mapping inputs to the hidden state sequence.

What is new? We introduce two novel aspects within this context. First, we allow for λ to be time-varying, and in particular, a function of previous hidden state

[1] Vanilla RNNs and residual variants amount to a suitable Euler discretization (see Appendix).

and input. Our reasoning is that λ serves as a time-constant, and inherently accounts for how long we evolve the ODE in response to the current input. To see this, let us write the ODE in integral form for a fixed λ:

$$\mathcal{S}_m \triangleq \mathbf{z}(t_0+1) = \exp\left(A\frac{1}{\lambda}\right)\mathbf{s}_{m-1} + \frac{1}{\lambda}\int_0^1 \exp\left(A\frac{1-t}{\lambda}\right)\phi(\mathbf{U}\mathbf{z}(t)+\mathbf{W}\mathbf{x}_m)dt \quad (4.2)$$

Then, with $\lambda \to \infty$, we deduce that, $\mathbf{z}(t_0+1) \to \mathbf{s}_{m-1}$. Namely, when time constant is large relative to integration time, we barely process the new input, remaining essentially at our previous solution. Alternatively, if $\lambda \to 0$, namely, when the integration time is large relative to the time-constant, we reach equilibrium, and in this process strengthen influence of the current input. Moreover, by letting the time-constant be a function, of $\mathbf{s}_{m-1}, \mathbf{x}_m$, we selectively adapt the amount of "pondering" that we need on each new input. Finally, we let $\lambda(\cdot)$ take values in $\mathbf{R}^D$, and thus allow for element-wise dependence for each hidden state, leading to selective updates of hidden state components. These ideas result in a time-adaptive RNN (TARNN).

Next, we augment the current input with the hidden state, and consider $\mathbf{u}_m = [\mathbf{x}_m, \mathbf{s}_{m-1}]^\top$ as a composite input in our ODE with initial condition, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$:

$$\lambda(\mathbf{u}_m) \circ \dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) \quad (4.3)$$

where $\circ$ represents the element-wise (Hadamard) product. To build intuition into our ODE choice, observe from the first term in Eq. 4.2 that for $\mathbf{A}$ stable, the contribution of the hidden state, $\mathcal{S}_{m-1}$ decays exponentially in time, and as such, the discrete transition process, $\mathcal{S}_1, \ldots, \mathcal{S}_T$ rapidly de-correlates. We can overcome this effect by a persistent presence of the hidden state in the ODE. We also add the linear term, $\mathbf{B}\mathbf{u}_m$, as it turns out to be important for improving partial gradient properties for hidden state sequence. As such our choice does not significantly increase model complexity

of vanilla RNN.

Our proposed ODE is sufficiently rich admitting parameter settings that completely eliminate gradient decay and explosion, which is desirable for LTD tasks. In addition, our method is capable of enhancing contribution from informative inputs, while suppressing noisy segments through the pondering mechanism described above. This aspect is useful in IoT applications (Kusupati et al., 2018; Dennis et al., 2019) such as keyword detection and wearable sensing devices.

Discretization: For simplicity we discretize our ODEs with Euler discretization to realize vanilla RNNs. Methods that seek computational and memory efficiency in this context (Chen et al., 2018; Rubanova et al., 2019) are entirely complementary to our method. Our novelty is in the design of state-transition with the goal of realizing desirable ODE solutions[2].

Contributions: The main contributions of this work are

- TARNN learns to modulate time constants of transition function, allowing for selectively pondering on informative inputs to strengthen their contribution, and ignoring noisy inputs. This modification along with designing suitable transition matrices yield lossless information propagation.
- TARNN improves trainability leading to better handling of LTD tasks with a lighter memory footprint, and as such our proposed method can be leveraged for IoT tasks.
- Our pseudo code is an RNN cell that is readily deployable in any deep learning library. We provide a simple implementation at `https://github.com/anilkagak2/TARNN`.
- We conduct extensive experiments on benchmark datasets, and show that we improve upon standard LSTM performance as well as other recently proposed works. We also demonstrate robustness to time-series distortions such as noise paddings.

[2](Chen et al., 2018; Rubanova et al., 2019), also propose recurrent models to handle non-uniform input sampling. While this is interesting, their proposals are unrelated to our goal of improving RNN trainability.

4.2 Time Adaptive Recurrent Neural Network (TARNN)

In this section we further present our objective, ODE discretization and algorithmic details.

Notation. $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, $i \in [N]$ denotes training data. Each $\mathbf{x}^{(i)}$ is a $T-$length $d-$dimensional sequential input. For classification problems, $\mathbf{y}^{(i)}$ is a terminal label $y_T^{(i)}$, taking values in a discrete set of C classes. For language modeling tasks, we let the true label be a process, $(y_1^{(i)}, \ldots, y_T^{(i)})$. The predictions $(\hat{y}_1^{(i)}, \ldots, \hat{y}_T^{(i)})$ for each input $\mathbf{x}^{(i)}$ can be computed from the $D-$dimensional hidden states $(\mathbf{s}_1^{(i)}, \ldots, \mathbf{s}_T^{(i)})$ obtained by solving the ODE Eq. 4.3. When clear from the context we omit superscripts. Unless stated otherwise, $\sigma(\cdot)$ denotes the sigmoid activation; $\phi(\cdot)$ refers to any non-linear activation such as a ReLU. We collect all model parameters in θ.

Empirical Risk Minimization. Let $\ell(\hat{y}, y)$ be the function measuring loss incurred for predicting value $\hat{y}$ on the true value y. Our objective is to minimize the regularized empirical loss, through back-propagation in any deep learning framework. We specify the regularizer $\Omega(\theta)$ later.

$$L(\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}) = \frac{1}{N}\frac{1}{T}\sum\nolimits_{i=1}^{N}\sum\nolimits_{m=1}^{T} \ell(\hat{y}_m^i, y_m^i) + \Omega(\theta) \tag{4.4}$$

Time-constants. We re-write the ODE Eq. 4.3 in terms of $\boldsymbol{\beta}(\cdot)$, the inverse of $\boldsymbol{\lambda}(\cdot)$, since it is convenient for describing our discretization steps. We parameterize $\boldsymbol{\beta}(\mathbf{u}_m) = \sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x\mathbf{x}_m)$, where $\mathbf{U}_s \in \mathbb{R}^{D\times D}, \mathbf{W}_x \in \mathbb{R}^{D\times d}$ are parameters to be learnt. For a component j where $\boldsymbol{\beta}_j \approx 1$, then $(\dot{\mathbf{z}}(t))_j \approx (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))_j$, and the system responds to the input $\mathbf{u}_m$ and reaches equilibrium. On the other hand, when $\boldsymbol{\beta}_j \approx 0$, then $(\dot{\mathbf{z}}(t))_j \approx 0$, and the corresponding state is frozen, with the input at time m completely skipped. In this paper we limit ourselves to a binary behavior, i.e. whether to ponder over the input observation for a long time or

not ponder at all. For this reason, it suffices to limit the range in $[0, 1]$ with sigmoid activation. This also avoids numerical instabilities with unbounded non-linearities.

Setting up the ODE. To obtain a discrete implementation, first, we update the ODE Eq. 4.3 with the change of variables for time-constants, resulting in the ODE:

$$\begin{aligned}\dot{\mathbf{z}}(t) =& \boldsymbol{\beta} \odot (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m)) \\ \triangleq& F(\mathbf{z}(t), \mathbf{u}_m); \;\; \mathbf{z}(t_0) = \mathbf{s}_{m-1}\end{aligned} \tag{4.5}$$

where, $\odot$ represents the Hadamard product. Next, we instantiate the specific parameterization for transition matrices. Finally, an ODE solver is invoked, over a time-horizon $[t_0, t_1]$ to update the state:

$$\mathbf{s}_m = \mathbf{z}(t_1); \;\; \mathbf{z}(t_1) = \text{ODESolve}(\mathbf{s}_{m-1}, \mathbf{x}_m, F(\cdot), t_0, t_1)$$

We predict the output $\hat{y}_m = \sigma(\mathbf{w}^\top \mathbf{s}_m + b)$ using a sigmoid activation on top of a linear layer parameterized as $(\mathbf{w}, b)$. Since, we need $\mathbf{A}$ to be Hurwitz-stable, and we impose equilibrium, when a component is active, we a priori fix $\mathbf{A}$ as negative identity. Other TARNN model parameters $(\mathbf{B}, \mathbf{U}, \mathbf{W}, \mathbf{w}, b, \mathbf{U}_s, \mathbf{W}_x)$ are learnt during training by minimizing the empirical loss in Eq. 4.4.

The ODE solver. A number of methods exists to numerically solve the ODE of Eq. 4.5 including black-box solvers such as Neural ODEs(Chen et al., 2018) or advanced root-finding methods such as the Broyden's method (Broyden, 1965). While these methods could be further employed to improve computational efficiency, for exposition we limit ourselves to Euler-recursion with $K = 3$ steps, since computational efficiency as such is not the focus of our paper. We let η denote the step-size, with

$\mathbf{z}_m^k$ denoting the recursion steps:

$$\mathbf{z}_m^k = \begin{cases} \mathbf{s}_{m-1} & \text{if} \quad k = 1 \\ \mathbf{z}_m^{k-1} + \eta(F(\mathbf{z}_m^{k-1}, \mathbf{u}_m)) & \text{if}\, 1 < k < K \end{cases} \tag{4.6}$$

$$\mathbf{s}_m = \mathbf{z}_m^K$$

As shown in the Sec. 4.2.1, for suitable choice of the activation function, $\phi(\cdot)$, (includes popular activations such as ReLU, tanh, sigmoid, etc.), these recursions in the limit, for $(\boldsymbol{\beta})_j > 0$, $\mathbf{z}_m^* = \lim_{k\to\infty} \mathbf{z}_m^k$ is an equilibrium solution to the ODE of Eq. 4.5. We provide the pseudo code in Algorithm 1, which generates the hidden states for a sequential input $\{\mathbf{x}_m\}_{m=1}^T$.

Algorithm 1 TARNN hidden states computation

Input : Sequence $\{\mathbf{x}_m\}_{m=1}^T$
Model : $(\mathbf{A}, \mathbf{U}, \mathbf{W}, \mathbf{U}_s, \mathbf{W}_s, \mathbf{B})$
Initialize hidden state $\mathbf{s}_0 = 0$
for $m = 1$ **to** T **do**
  $\boldsymbol{\beta} = \sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x\mathbf{x}_m)$
  $F(\cdot) = \boldsymbol{\beta} \odot (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))$
  $\mathbf{z}(t_1) = \text{ODESolve}(\ \mathbf{s}_{m-1}, \mathbf{x}_m, F(\cdot), t_0, t_1\ )$
  $\mathcal{S}_m = \mathbf{z}(t_1)$

4.2.1 Analysis

In this section, we show that our setup benefits from several properties, and as a result, our proposed method leads to a theoretically sound approach for an adaptive recurrent system that is capable of focusing attention on informative inputs and rejecting uninformative inputs. The first few propositions establish properties of TARNN with the proposed parameterization. We then describe a result to assert that our adaptively recurrent system preserves information by showing that the partial gradients of hidden states have unit norm.

The following proposition shows that equilibrium points for the ODE of Eq. 4.5 exist and are unique. Although, we a priori fix $\mathbf{A}$ to be negative identity, we present a more general result for the sake of completion. We impose the following conditions, (i) there is a $\eta_0 > 0$ such that for all $\eta \in [0, \eta_0]$, there is some $\alpha \in (0, 1]$ such that $\sigma_{\max}(I + \eta A) \leq 1 - \alpha\eta$. (ii) $\lambda_{\max}(\mathbf{A} + \mathbf{A}^\top) < -1$. It is easily verified that these conditions are satisfied in a number of cases including $\mathbf{A}$ -identity, $\mathbf{A}$ block triangular with negative identity blocks.

Proposition 3. *Consider the ODE in Eq. 4.5 and assumptions on* $\mathbf{A}$ *described above. Suppose we have* $\|U\| < \alpha$, *and* $\phi(\cdot)$ *is 1-Lipshitz function, it follows that, for any given,* $\boldsymbol{\beta}$, $\mathbf{u}_m$, *an equilibrium point exists and is unique.*

Remark. Note that, we impose conditions on $\mathbf{U}$ to derive our result. In experiments we do not impose this condition, since for our choices for $\mathbf{A}$, $\alpha \approx 1$, and as such, initializing $\mathbf{U}$ to a Gaussian zero-mean, unit covariance often takes care of this requirement during training, since we generally operate with a small learning rate.

Proof Sketch. To show this we must find a solution to the non-linear equation $\mathbf{Az}+\mathbf{Bu}_m-\phi(\mathbf{Uz}+\mathbf{Wu}_m) = 0$ and show that it is unique. We do this by constructing a fixed-point iterate, and show that the iteration is contractive. The result then follows by invoking the Banach fixed point theorem (contraction-mapping theorem). The proof is presented in the appendix 6.

Proposition 4. *With the setup in Proposition 3, and regardless of* $\boldsymbol{\beta}$, *the equilibrium point is globally asymptotically stable, and the discrete Euler recursion converges to the equilibrium solution at a linear rate.*

We discuss the main idea and present the proof in the appendix. Let $\mathbf{z}^*$ be the equilibrium solution. We consider the Lyapunov function $V(\mathbf{z}(t)) = \|\mathbf{z}(t) - \mathbf{z}^*\|^2$ and show that it is monotonically decreasing along the ODE system trajectories. Observe

that, as per our setup, components where $(\boldsymbol{\beta})_j = 0$ does not pose a problem, because those states remain frozen, and serve as an additional exogenous input in our ODE.

Lossless Information Propagation. Our goal is to show that there exist parameter constraints in Eq. 4.5 that can result in identity partial gradients of the hidden states. This will in turn inform our regularization objective, $\Omega(\theta)$ later. With the constraint in place, for arbitrary values, m, $n \in \mathbb{Z}^+$, we will show that, $\frac{\partial \mathbf{s}_n(j)}{\partial \mathbf{s}_m(j)} = 1$. For ease of analysis we replace binary-valued $\boldsymbol{\beta}$ with a continuous function and let the output be a ReLU non-linearity. Partition $\mathbf{W} = [\mathbf{W}^1, \mathbf{W}^2]$, $\mathbf{B} = [\mathbf{B}^1, \mathbf{B}^2]$, where $\mathbf{W}^2, \mathbf{B}^2 \in \mathbf{R}^{D \times D}$ are associated with the hidden state components. To realize identity gradients for a specific component i we need to constrain the parameter space. While there are many possibilities, we consider following constraints, because they lead to concrete regularization objectives, and generalize the specific $\mathbf{A}$ matrices we have in mind (identity, and upper-triangular). We constrain $\|\mathbf{U}\| < 1 \leq \|A\|$, and consider the following case: $\mathbf{A} \pm \mathbf{B}^2 = 0$, $\mathbf{U} \pm \mathbf{W}^2 = 0$.

Theorem 2. *Under the above setup, as $K \to \infty$ in Eq. 4.6, for any $m, n \in \mathbb{Z}^+$, $|\partial \mathbf{s}_n(i) / \partial \mathbf{s}_m(i)| \to 1$.*

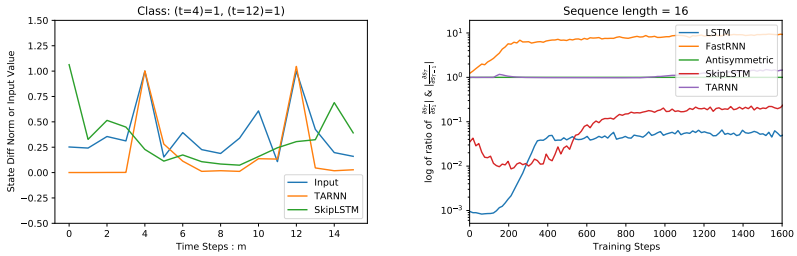
Proof Sketch (see Appendix for proof). Note that, when $\beta_j = 0$, the jth component $\mathbf{s}_m(j) = \mathbf{s}_{m-1}(j)$ and the result follows trivially. Suppose now the jth component $(\beta)_j > 0$, we will show that, $\partial \mathbf{s}_m(j) / \partial \mathbf{s}_{m-1}(j) = 1$, which then establishes the result through chain rule.

Theorem 2 shows that there is a configuration with lossless propagation. Thus, if it is necessary, the training algorithm will find a solution, that results in lossless propagation, even without imposing parameter constraints stated in the theorem. However, Theorem 2 suggests a natural regularizer, with γ_1 and γ_2 serving as hyperparameters. As a case in point, we could encourage parameters to subscribe to

constraints of theorem if we consider the following regularizer for Eq. 4.4:

$$\Omega(\theta) \triangleq \Omega([\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{W}]) = \gamma_1 \|\mathbf{A} + \mathbf{B}_2\|_2^2 + \gamma_2 \|\mathbf{U} + \mathbf{W}_2\|_2^2$$

An interesting case is when B^2 row-wise sparse. In this case, states corresponding to zero rows operate as standard RNN (no linear term). We can ensure identity gradient holds in this case with block-wise parametric constraints, leading to more structured regularization penalty.



Algorithm	Accuracy
Random Guess	25
FastRNN	45
LSTM	45
Antisymmetric	37
SkipLSTM	60
TARNN (Ours)	**100**

(a) **(b)** **(c)**

Figure 4·1: Example illustrates importance of mitigating gradient explosion/decay as well as ignoring noisy observations. Table lists test performance of baselines focused on improving RNN training. Fig. (a) plots the noisy input, and sequential changes in hidden state norms for SkipLSTM(Campos et al., 2018) and proposed TARNN. Only ours responds to informative locations. Fig. (b) plots the norm of partials of hidden states. Only AntisymmetricRNN(Chang et al., 2019) and ours TARNN exhibit near identity gradients. However, only ours is effective as seen from the table. As such we infer TARNN (a) realizes near identity gradients for partials of hidden states, thus mitigating gradient explosion/decay, (b) zooms in on informative inputs and ignores noisy observations, and (c) By jointly ensuring (a) and (b), it improves RNN trainability, providing good generalization.

4.3 Experiments

Toy Example. For a sneak preview of our results, we illustrate the importance of both time-constants and gradient mitigation on a toy example. We construct a 16-length input sequence with 4 class labels. Information is placed in the form of binary

$\{0, 1\}$ values at locations $4, 12$, corresponding to the four classes, and for all other locations we assign values from a uniform distribution in the unit interval. RNNs with a 2-dimensional state-space are trained on 50K time-traces. Due to low-dimension, the (terminal) state cannot replicate the entire trace, requiring generalization.

On one hand, techniques that mitigate gradient explosion/decay like Antisymmetric (Chang et al., 2019), do so across all input locations, but fail to output meaningful results as seen from Figure 4·1(c). Thus focusing solely on vanishing/exploding gradients is not sufficient, since noise gets amplified in latent state updates. On the other hand, SkipLSTM (Campos et al., 2018), which is capable of pondering at informative inputs and skipping uninformative inputs, is also ineffective. SkipLSTM (Campos et al., 2018) suffers severe gradient degradation, leading to poor control over which locations to ponder. In contrast, TARNN exhibits near identity gradients, skips all but locations $4, 12$, and achieves 100% accuracy. Similar trend holds for larger state space (see Appendix).

4.3.1 Experimental Setup and Baselines

Datasets

We follow earlier works (Chang et al., 2019; Kag et al., 2020; Kusupati et al., 2018) in order to setup experiments. Datasets used in this work are publicly available, except NTU RGB+d(Shahroudy et al., 2016) (skeleton modality is available for academic usage). We use 10% of the training data as validation set for tuning the hyper-parameters through grid search. The grid for each method is setup as per their experimental section. Finally, the entire training set is used to train the model. The performance is reported on the publicly available test set. We refer the reader to Appendix A.2 for detailed description.

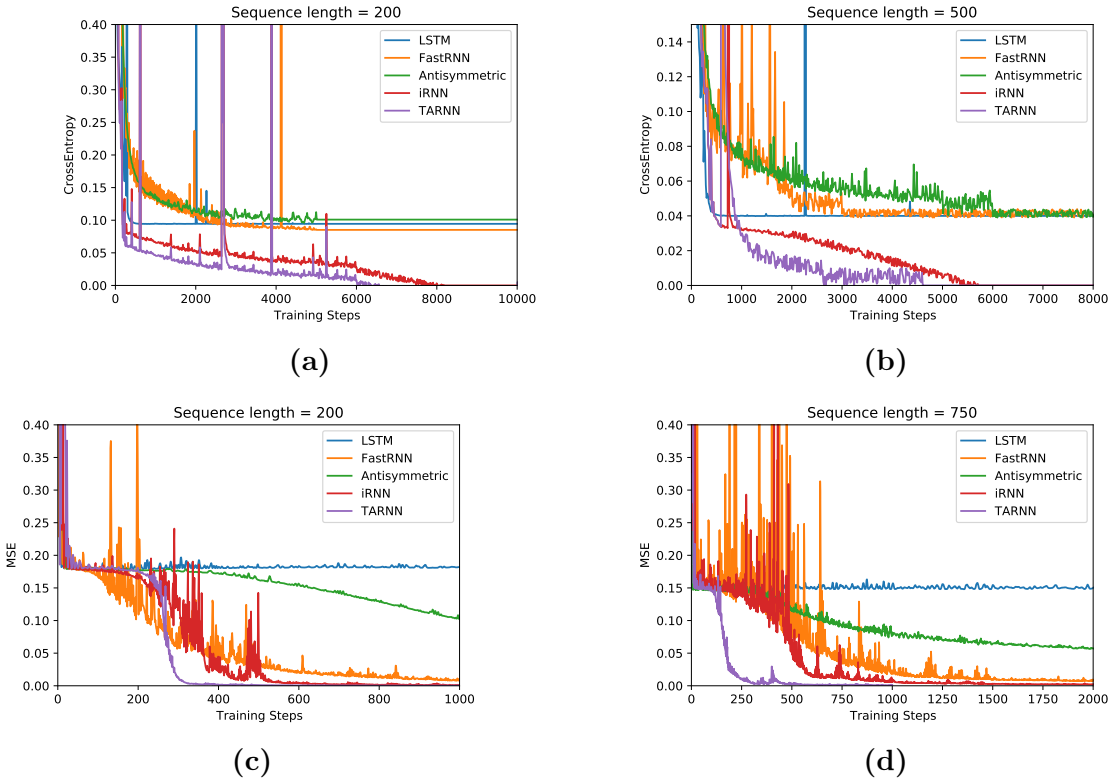


Figure 4·2: We evaluate TARNN on synthetic LTD tasks: Copy task with sequence lengths : (a) 200, (b) 500, and Add task with sequence lengths: (c) 200, (d) 750. Note that many methods perform similar to a simple fixed baselines described in (Kag et al., 2020)(Appendix:A.4), while TARNN achieves significantly better solution in fewer training steps.

Table 4.1: Results for Pixel MNIST, Permuted MNIST, Noise Padded CIFAR-10 and MNIST datasets. Since TARNN effectively focuses on informative segments, it achieves better performance with faster convergence. Note that we only keep baselines which report results with single RNN layer and no batch normalization (this excludes baselines such as (Li et al., 2018b), (Cooijmans et al., 2017)).

Dataset	**Pixel-MNIST**				**Permute-MNIST**			
	Hidden Dim.	**Accuracy (%)**	**Train Time (hr)**	**#Params**	**Hidden Dim.**	**Accuracy (%)**	**Train Time (hr)**	**#Params**
FastRNN	128	97.71	16.10	18K	128	92.68	9.32	18K
LSTM	128	97.81	26.57	68K	128	92.61	19.31	68K
SkipLSTM	128	97.31	-	68K	128	93.72	23.31	68K
Antisymmetric	128	98.81	10.34	18K	128	93.59	4.75	18K
expRNN	128	97.35	-	34K	128	94.01	-	34K
nnRNN	128	97.81	-	51K	128	94.29	-	51K
iRNN	128	98.13	2.93	**4K**	128	95.62	2.41	**8K**
TARNN	32	98.43	**2.13**	10K	32	96.21	**1.71**	10K
TARNN	128	**98.93**	3.42	68K	128	**97.13**	2.96	68K
Dataset	**Noisy-MNIST**				**Noisy-CIFAR**			
FastRNN	128	98.12	8.93	11K	128	45.76	11.61	16K
(Skip)LSTM	128	10.21	19.43	82K	128	10.41	13.31	114K
Antisymmetric	128	97.76	5.21	10K	128	54.70	7.48	41K
expRNN	128	97.92	-	37K	128	48.97	-	47K
nnRNN	128	98.06	-	54K	128	49.28	-	63K
iRNN	128	98.48	2.14	**6K**	128	54.50	2.47	**12K**
TARNN	32	98.78	**1.31**	8K	32	57.42	2.01	14K
TARNN	128	**99.03**	1.71	78K	128	**59.06**	**1.05**	100K

Baselines

We use various state-of-the-art methods for evaluating TARNN's performance, including popular RNNs such as: gating methods(LSTM(Hochreiter and Schmidhuber, 1997b), FastGRNN(Kusupati et al., 2018)), ODE inspired(iRNN (Kag et al., 2020), AntisymmetricRNN (Chang et al., 2019)), conditional computation (SkipRNN (Campos et al., 2018)), as well as recent Unitary/Orthogonal (nnRNN(Kerg et al., 2019) & expRNN(Lezcano-Casado and Martínez-Rubio, 2019)). For baselines with gated/ungated variants, we report results for the best of the two. We also tried to incorporate SkipRNNs (Campos et al., 2018) in our baselines, but for many of our tasks, its performance remained similar to the corresponding RNN variant. Hence, we do not list SkipRNNs on all our experiments. Furthermore, adaptive computation time (ACT) (Graves, 2016) is not tabulated as we found that performance of SkipLSTM is significantly better. This has also been observed in (Fojo et al., 2018), who shows repeat-RNNs, a variant of iRNN outperforms ACT. Note that we do not report baselines (Lei et al., 2018; Bradbury et al., 2016; Li et al., 2018b) which trade-off non-linear hidden-to-hidden connections with linear connections, since these interactions are complementary to our method and can be incorporated in TARNN for computational advantages. Note that the datasets 5 and 6 (PTB-w, PTB-c and Action recognition) are computationally expensive and take days for a single run with standard baselines, hence we do not run baselines on these datasets and simply cite the current best known results. These datasets demonstrate that, (a) TARNN can outperform baselines with smaller models, and (b) since these datasets require stacked or other complex architectures, our experiments show that multi-layered TARNN can be trained with similar ease.

Code & Evaluation Metrics

We implement TARNN in the tensorflow framework using the pseudo code. Most of the baselines are publicly available except Antisymmetric and Incremental RNNs which provide pseudo code for implementation. For all the methods, we report the accuracy, training time and the model parameters. Unfortunately, we do not report train times for nnRNN and expRNN as their code is written in PyTorch. We use Adam((Kingma and Ba, 2015)) for minimizing the loss function in Eq. 4.4. We provide the final hyper-parameters along with the grid values for our experiments in the appendix C.2. Our inference time is comparable to FastRNNs and iRNNs, in contrast LSTMs take 4x longer for inference (see Appendix C.9).

4.3.2 Results and Discussion

Figure 4·2 shows the results on Copy and Add tasks. Table 4.1 reports the performance on Pixel-MNIST, Permute-MNIST, Noisy-MNIST and Noisy-CIFAR datasets. These results show that TARNN outperforms various methods on many benchmark LTD tasks, which can be attributed to its near lossless gradient propagation between informative segments. Additionally, tables 4.2 and 4.3 report TARNN's performance on various PTB datasets, and table 4.4 lists accuracies of all the methods on CS and CV variants of the Skeleton based Action recognition task. These experiments demonstrate that TARNN outperforms many baselines in learning short-term dependencies on language modelling tasks and terminal short term dependency task. Below we present TARNN's useful properties backed by empirical evaluations.

(A) Fast convergence. Figure 4·2 shows the convergence plots for various methods on the Add & Copy tasks. It should be observed that TARNN solves both of these tasks significantly faster than the baselines. Due to poor gradient propagation, LSTMs only achieve the performance of fixed strategies. While iRNN solves these two

Table 4.2: PTB Language Modeling: 1 Layer (standard small config except the sequence length is 300 as per (Kusupati et al., 2018) as opposed to 70 in the conventional PTB). TARNN achieves significantly better performance than the baselines on this task (even with half the hidden dimensions than the baselines). Note that embedding size is same as hidden dimension in these experiments, thus smaller hidden dimensions result in smaller embedding storage as well.

Algorithm	Hidden Dimension	Test Perplexity	Train Time (min)	#Params
FastRNN	256	115.92	40.33	131K
LSTM	256	116.86	56.52	524K
SkipLSTM	256	114.23	63.52	524K
iRNN	256	113.38	**34.11**	**100K**
TARNN	128	**102.42**	40.23	114K
TARNN	256	**94.62**	53.16	524K

tasks, it requires more training steps to reach the desired target error. Note that we do not show Unitary RNNs on these tasks, as they take significantly longer number of training steps to solve the Addition task, and benefit from the modReLU activation on the copy tasks (Kag et al., 2020). Similarly, TARNN trains significantly faster on LTD tasks presented in the Table 4.1 (at least 8× faster than LSTMs and at least 1.3× faster than the best).

(B) Better generalization. Table 4.1 shows that TARNN outperforms the baselines resulting in better accuracies on all the terminal prediction tasks. On Noisy-CIFAR dataset, TARNN achieves more than four points increase in accuracy, while on the 300-length PTB language modelling task, we get nearly 20 points better in perplexity than the best method.

(C) Noise resiliency. In order to evaluate TARNN's noise resilience, we conduct experiments on the Noisy-MNIST and Noisy-CIFAR datasets (Chang et al., 2019; Kag et al., 2020) which introduces the informative segments in the first few timesteps and embeds every other segment with noise. These datasets requires both lossless gradient propagation along with the ability to suppress noisy segments and only focus on informative segments. Intuitively we expect to perform better on this task since TARNN selectively ponders on informative segments to strengthen

their contribution and allows the state transition to achieve near lossless gradient propagation. Table 4.1 shows that TARNN achieves much better performance than iRNNs/AntisymmetricRNNs which in turn beat the remaining methods by significant margins.

Table 4.3: Results for Penn Tree Bank Character and Word level language modelling tasks. These use shorter sequence length (typically 50-150) and use more than one RNN layer for modelling. For the PTB-w dataset, where ever applicable, all the baselines report the results with dynamiceval(Krause et al., 2018). Our model uses 3 layer composition. It can be seen that we report reasonable performance with much smaller models than other methods. With comparable model sizes as the baselines we report higher performance. In the table, NAS stands for Neural Architecture Search baseline.

Dataset	**PTB-c**			**PTB-w**		
	Hidden Dim.	**BPC**	**#Params**	**Hidden Dim.**	**Perplexity**	**#Params**
(GAM) RHN(Luo and Yu, 2019)	600	**1.147**	16M	830	66.0	24M
Trellis-Net (Bai et al., 2019b)	1000	1.158	13.4M	1000	54.19	34M
AWD-LSTM (Merity et al., 2018)	1000	1.175	13.8M	1150	51.1	24M
NAS (Zoph and Le, 2016)	800	1.21	16.3M	800	62.4	54M
IndRNN (Li et al., 2018b)	2000	1.21	22M	2000	60.21	28M
Residual IndRNN (Li et al., 2019b)	2000	1.19	50.7M	2000	58.99	57M
Dense IndRNN (Li et al., 2019b)	2000	1.18	45.7M	2000	**50.97**	52M
TARNN	500	1.29	**7M**	500	60.90	**11M**
TARNN	1400	1.19	42M	1200	53.21	56M

(D) Adapts well on short-term dependency tasks. We benchmark TARNN on PTB-300 dataset. We do not report expRNN and nnRNN results as they perform poorly in comparison to LSTM (Kerg et al., 2019). Table 4.2 reports all the evaluation metrics for the PTB Language modelling task with 1 layer as setup by (Kusupati et al., 2018). It can be clearly seen that TARNN outperforms the baselines by roughly ≈ 10 point difference in the test perplexity for similar model complexity while it achieves ≈ 20 points for a larger model. Likewise, TARNN adapts well to other short-term dependency tasks as observed by Table 4.3 and Table 4.4.

(E) Low model complexity. Table 4.1, 4.2 show TARNN performance with two different hidden state dimensions, namely one configuration with similar model size as iRNN and other one with similar model size as larger RNNs. With model complexity similar to iRNNs, which are much compact than the other baselines, we

achieve better performance than iRNNs. With larger model complexity, we achieve much better performance on Permute-MNIST, Noisy-CIFAR and PTB datasets. The other tasks are relatively saturated as almost all the methods are near optimal. We point out that the number of parameters reported in the Table 4.2 only count the RNN parameters and omit the embeddings. We achieve 102 perplexity with lower hidden dimension, i.e. 128. This means we require less number of parameters for the embedding representation. Similarly, Table 4.3, 4.4 compare TARNN's performance on larger multi-layered RNN tasks, namely PTB-c, PTB-w, and Action recognition. It can be seen that TARNN achieves similar performance as known baselines with much smaller model.

Table 4.4: Results for NTU RGB-d dataset (Skeleton based action recognition). We do not use augmentation on top of the Skeleton data. We point out that TARNN achieves competitive performance with much lower complexity model. We also ran a dense variant of TARNN similar to IndRNN that results in better performance.

Dataset	**NTU RGB-d**		
	Accuracy CS (%)	**Accuracy CV (%)**	**#Params**
2-Layer LSTM (Shahroudy et al., 2016)	60.09	67.29	>1M
2-Layer PLSTM (Shahroudy et al., 2016)	62.93	70.27	>1M
Enhanced Visualization+CNN (Liu et al., 2017b)	80.03	87.21	-
Pose Conditioned STA-LSTM (Baradel et al., 2017)	77.10	84.50	-
6-Layer IndRNN (Li et al., 2018b)	81.80	87.97	2M
Dense IndRNN (Li et al., 2019b)	**84.88**	90.43	2.3M
3-Layer TARNN	80.52	87.54	**180K**
Dense TARNN	82.31	**90.86**	5.6M

RNN Trainability. *TARNN exhibits substantial improvement with respect to (a) size of memory footprint, (b) computational efficiency (faster convergence, training and inference times), and (c) generalization (test performance).* As evident from the Tables 4.1, 4.2, 4.3, and 4.4, TARNN is consistently among the models with lowest number of model parameters. It enjoys faster convergence rate as evident from the convergence plots for addition and copying tasks (Figure 4·2) and toy example (Appendix C.5). Thus improving the training time. It should also be noted that TARNN has similar inference time as vanilla RNNs. It also generalizes well as evident from the test accuracy on multiple synthetic and real-world tasks. This is attributed

to the ability to achieve near identity gradients and effectively skipping uninformative input segments. This leads to the conclusion that TARNN improves vanilla RNN training. Due to the light footprint TARNN is suitable for IoT tasks. We tabulate results for IoT datasets where TARNN outperforms baselines (see Appendix C.3).

4.4 Discussion

Similar to iRNN, TARNN uses the Euler discretization for a simple iterative solver. We can improve this architecture using higher-order solvers such as Runge-Kutta, Leapfrog integration, etc.

In scenarios where compute/storage is not the limit, transformers (Vaswani et al., 2017) have replaced RNNs. It is worth pointing out that we can extend the ideas presented in this chapter to transformer architectures. More specifically, the pairwise dependencies modeled by transformers for the entire sequence lengths can be emulated with RNNs by feeding the entire sequence as the input in a single time step rather than the sequential input per time step. It would be a good exploration direction to see the extensions such as TARNN or other newer architectures could be leveraged as it is in the transformer architectures.

Chapter 5

Forward Propagation Through Time (FPTT)

5.1 Introduction

Given the training dataset $\{x_i, y_i\}_{i=1}^N$ with N examples of $T-$length sequences x_i, y_i, we optimize the following empirical risk function:

$$[W^*, v^*] = \arg\min_{W,v} L(W, v) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \ell(y_t^i, \hat{y}_t^i) \tag{5.1}$$
$$\forall i, t;\ \hat{y}_t^i = v^\top h_t^i;\ h_t^i = f(W, x_t^i, h_{t-1}^i);\quad h_0^i = 0$$

where, RNN parameters W and classifier $v \in \mathbb{R}^D$ are optimized. This objective is minimized by BPTT that has many challenges (see Sec. 2.2).

In this work, our focus is on simplifying the RNN training procedure. Once the RNN parameters are learnt, the inference process remains same as before. Our goal is to reduce computational complexity of BPTT so that each step only involves taking a derivative for a single time.

Challenges. Minimizing Eq. 5.1 poses two challenges:

(a) *Dynamics.* Equation 5.1 enforces a temporal constraint on allowable transitions.

(b) *Time-Invariance.* The transition matrices W are fixed, and as a result the dynamics of RNNs is time-invariant.

Let us examine a few potential directions in this context.

Method of Multipliers allows for eliminating constraints imposed by (a) and (b) by introducing a regularizer based on an augmented Lagrangian. This approach is often adopted in distributed optimization (Boyd et al., 2011). Eliminating (a) could be accomplished with ADMM methods with squared norm penalty, or other specialized functions (Gu et al., 2020). For (b), leveraging the key insight in distributed optimization, we can write the condition (b) as $W_t = W_{t-1}$ for all t, and rewrite it as a penalty. Nevertheless, while computationally block-coordinate descent allows for efficiency, memory expands substantially ($O(TD^2 + NTD)$).

Online Gradient Method (OGD). We can view $\ell_t(W) = \ell(y_t, v^\top f(W, x_t, h_{t-1})$ as the instantaneous loss incurred in round t by "playing" the parameter W. We can update $W_{t+1} = W_t - \eta \nabla \ell_t(W_t)$, based on the observed loss. While this could work, there is no reason why W_t's converge, and furthermore, it is unclear how to choose a constant parameter, W, based on the sequence of updates. In general, we have observed in experiments that training performance based on time-varying transition matrices does not reflect test-time and does not generalize well.

Follow-the-Regularized-Leader Rule. (McMahan et al., 2013) Rather than optimizing the instantaneous loss as in OGD, we utilize all of the previously seen losses, namely, $L_t(W) = \sum_{j=1}^{t} \ell_j(W)$ and attempt to find a update direction. Nevertheless, this approach suffers from the same issue as BPTT, since for large $t \approx T$, finding a descent direction involves back-propagation through $\approx T$ steps. Furthermore, this method adds a multiplicative factor of T in the run-time in comparison to BPTT.

Our Forward-Propagation Method. We propose a novel forward-propagation-through-time (FPTT) method based on instantaneous dynamic regularization. FPTT at each time takes a gradient step to minimize an instantaneous risk function. The instantaneous risk is the loss at time t plus a dynamically evolving regularizer. This dynamics is controlled by a state-vector, which summarizes past

losses. FPTT has the in-built property that the point of convergence of W_t sequence, is also a stationary point of the global empirical risk Equation 5.1. The resulting method has a light-weight footprint and is computationally efficient. For sequence-to-sequence modelling tasks our learning scheme integrates easily since the losses are instantaneous, i.e., at timestep t, we immediately get feedback for the updated W_t. For terminal prediction problems we present a simple scheme to construct surrogate losses at any timestep using the label for the entire sequence.

We then conduct a number of experiments on benchmark datasets and show that our proposed method is particularly effective on tasks that exhibit long-range dependencies. In summary, our proposed method suggests that vanilla LSTMs are effective tools for inferring long-term dependencies, and exhibit performance matching state-of-the-art competitors–even those with higher capacities and well-designed architectures.

Toy Example. As a sneak preview, we demonstrate effectiveness of FPTT on the Add Task (see Sec. 5.3 for details) against BPTT on training LSTMs under an identical test/train split. Figure 5·1 shows that FPTT solves this problem while BPTT fails to find the correct parameters, it stays near the same loss value throughout the training phase. BPTT's poor behavior on this task has been observed in previous works (Kag et al., 2020; Zhang et al., 2018a).

Contributions.

- We proposed forward-propagation-through-time (FPTT) as an alternative to conventional BPTT.
- FPTT takes a gradient step of an instantaneous time-dependent risk function at each time. The risk function is the regularized loss, with a dynamic evolving regularization. The dynamic penalty requires minimal memory, thus allowing for rapid gradient computation.

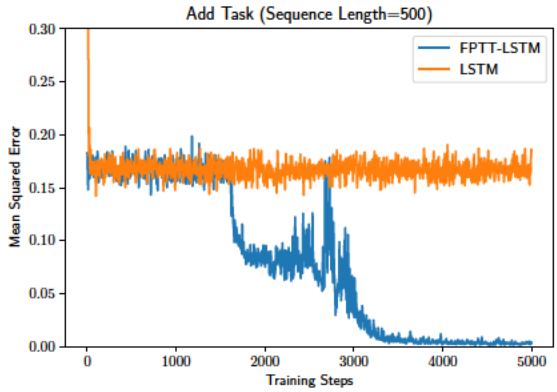


Figure 5·1: Add Task ($T = 500$): Comparison between standard learning and forward propagation.

- We construct surrogate losses for terminal prediction tasks, to guide FPTT to learn in intermediate time.
- We perform empirical evaluations to demonstrate the utility and superiority of FPTT over BPTT.
- Our FPTT algorithm can be readily deployed in any deep learning library. We have released our implementation at https://github.com/anilkagak2/FPTT.

5.2 Method

First, we describe our proposed algorithm and our learning objective. We describe a general pseudo code that can be leveraged to train any RNN architecture. Finally, we will describe a method for dealing with terminal prediction tasks.

Notation. The training set $B = \{x^i, y^i\}_{i=1}^N$ consists of N examples. Each input $x^i \in \mathcal{X}$ is a $T-$length sequence which can be written as $\{x_1^i, x_2^i, \cdots, x_T^i\}$. For sequence-to-sequence modelling tasks, the label $y^i \in \mathcal{Y}$ is a $T-$length sequence written as $\{y_1^i, y_2^i, \cdots, y_T^i\}$. While in the terminal prediction case, the label $y^i \in \mathcal{Y}$ is provided for a single timestep T as the feedback for the entire input sequence x^i. We will drop the superscript i to denote a data point wherever it can be inferred from

Algorithm 2 Training RNN with BackProp

Input: Training data $B = \{x^i, y^i\}_{i=1}^N$, Timesteps T
Input: Learning rate η, #Epochs E
Initialize: W_1 randomly in the domain $\mathcal{W}$
for $e = 1$ **to** E **do**
  Randomly Shuffle B
  for $i = 1$ **to** N **do**
    Set: $(x, y) = (x^i, y^i)$ and $h_0 = 0$
    for $t = 1$ **to** T **do**
      Update : $h_t = f(W, x_t, h_{t-1})$
    Loss: $\ell(W) = \sum_{t=1}^T \ell(y_t, v^\top h_t)$
    Set: : $W_{i+1} = W_i - \eta \nabla_W \ell(W)|_{W=W_i}$
  Reset: $W_1 = W_{N+1}$
Return : W_{N+1}

the context. With initial hidden state $h_0 = 0 \in \mathcal{H}$, an RNN with parameters $W \in \mathcal{W}$ and transition function $f : \mathcal{W} \times \mathcal{X} \times \mathcal{H} \to \mathcal{H}$ generates the hidden state sequence $\{h_1, h_2, \cdots, h_T\}$ for the data point $\{x, y\}$. We use $\ell_t(W) = \ell(y_t, v^\top f(W, x_t, h_{t-1}))$ to denote the loss incurred at time step t, where $v \in \mathcal{V}$ is a linear classifier.

To simplify the equations, (a) we will only use one example and drop the $\sum_i$ and superscript i used in Eq. 5.1, since the motivation behind the proposal remains same, and (b) we will assume v is constant, while in practice to learn v, the operations applied on W, are applied on v as well.

5.2.1 FPTT: Forward Propagation Through Time

Given one example $(x, y) = (\{x_t\}_{t=1}^T, \{y_t\}_{t=1}^T)$ and initial parameter estimate W_0, BPTT (see Algorithm 2) updates the parameter once by taking the gradient of the T length loss $\sum_{t=1}^T \ell_t(W)$. In contrast, we update parameters at every time step t by utilizing (x_t, y_t) to avoid getting penalized by T length gradient dependence. Since the parameters update very frequently, we need to incorporate two mechanisms in parameter updates: (a) *stability* in updates so that a single step does not stray,

(b) since our training does not follow standard RNN transition (i.e. keep a single parameter W through the input sequence), our updates should ensure that the iterates converge to a single parameter. This in turn guarantees that towards the end of the training sequence we will mimic an RNN.

To build motivation into our method Algorithm 3, we refer to the sequence of updates on a single instance, $i \in [N]$ at time step t. The first update is a gradient step of the loss $\ell_t(W)$ for a fixed value of $\bar{W}_t$. As such, we track one additional copy of the parameter, W_t, namely $\bar{W}_t \in \mathcal{W}$. At time step t, we update parameters using the supervision (x_t, y_t) and previous iterates $W_t, \bar{W}_t$. Following T updates, on a new instance, we set the initial weight parameter $W_0 \leftarrow W_{T+1}$, and the iteration follows subsequently.

To understand our scheme, let us consider the situation where the number of gradient steps of the loss approaches infinity. In this case, our equations read as:

$$W_{t+1} = \arg\min_{W} \ell_t(W) + \frac{\alpha}{2}\|W - \bar{W}_t - \frac{1}{2\alpha}\nabla\ell_{t-1}(W_t)\|^2 \tag{5.2}$$

$$\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla\ell_t(W_{t+1}) \tag{5.3}$$

These update equations are loosely inspired by consensus in distributed optimization problems over a star-network[1]

Intuition. The basic concept here is that $\bar{W}_t$ represents, in principle, the running average of all the W_t's seen so far, with a small correction term (Eq. 5.3). Therefore, the updates impose proximity to the running average in the update step. However, this alone is not sufficient to converge to stationary points of Eq. 2.3. We will show

[1]Distributed agents connected over a star-network seek to solve a joint optimization problem, which requires seeking consensus on the decision variables (Boyd et al., 2011). A master agent coordinates with the agents to communicate and synchronize decision variables in an iterative fashion. Eq. 5.1 could be viewed in a number of ways, as a single-agent network, a T node network, or an N node network etc. Each of these in turn lead to different coordinating mechanisms.

Algorithm 3 Training RNN with FPTT

Input: Training data $B = \{x^i, y^i\}_{i=1}^N$, Timesteps T
Input: Learning rate η, Hyper-parameter α, # Epochs E
Initialize: W_1 randomly in the domain $\mathcal{W}$
Initialize: $\bar{W}_1 = W_1$
for $e = 1$ **to** E **do**
  Randomly Shuffle B
  for $i = 1$ **to** N **do**
    Set: $(x, y) = (x^i, y^i)$ and $h_0 = 0$
    for $t = 1$ **to** T **do**
      Update : $h_t = f(W, x_t, h_{t-1})$
      $\ell_t(W) = \ell_t(y_t, v^\top h_t)$
      $\ell(W) = \ell_t(W) + \frac{\alpha}{2}\|W - \bar{W}_t - \frac{1}{2\alpha}\nabla\ell_{t-1}(W_t)\|^2$
      $W_{t+1} = W_t - \eta\nabla_W \ell(W)|_{W=W_t}$
      $\bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla\ell_t(W_{t+1})$
    Reset: $W_1 = W_T$ and $\bar{W}_1 = \bar{W}_T$
Return : W_T

this later. As such $\bar{W}_t$ is a vector that summarize past losses. Eq. 5.3 is also the first order condition for $\ell_t(W_{t+1}) + \frac{\alpha}{2}\|W_{t+1} - \bar{W}_t - \frac{1}{\alpha}\nabla\ell_{t-1}(W_t)\|^2$. Taken together the scheme resembles an alternative optimization method for a joint risk function over $W, \bar{W}$, namely, we hold $\bar{W}_t$ fixed and optimize W, and after the update, optimize $\bar{W}$ with fixed W. However, notice that unlike conventional setting, here the risk functions are time-varying. Note that Eq. 5.3 requires gradient of the loss ℓ_t at the new iterate W_{t+1}. Computational cost for this step can be eliminated by keeping a running estimate λ_t with update equation $\lambda_{t+1} = \lambda_t - \alpha(W_{t+1} - \bar{W}_t)$ and initial value $\lambda_0 = 0$.

Observe that for large α, we expect W_{t+1} to be close to the previous W_t, and this would result in the hidden state sequence $\{h_t\}_{t=1}^T$ to be essentially very close to the one generated by a single $W \approx W_{t+1} \approx W_t$. In effect this would simulate hidden state trajectories with a static time-invariant RNN parameter.

Pseudo Code. Algorithms 2 and 3 enumerates the learning schemes for BPTT and FPTT respectively. These procedures can be utilized to train an RNN architecture in

any popular deep learning framework with minimal efforts. Note that for simplicity we write the algorithms with batch size 1, this is relaxed to the conventional choice of larger batch size in our experiments. In FPTT, starting with small values of α we gradually increase α to enforce the constraint. We explore the impact of this hyper-parameter in the ablative experiments (see Sec. 5.3.2).

Remarks. (a) Note that even though we have separate W_t for each timestep, we do not suffer additional storage overhead of the factor T. This follows from the fact that we solve these sub-problems forward in time and only solve for W_{t+1} at timestep t. (b) We show that the iterates converge in our ablative experiments (see supplementary), below we provide an explanation for the convergence.

Convergence. Let us focus on the arg min step in Eq. 5.2. Taking gradient w.r.t. W results in the following dynamics:

$$\begin{aligned} \nabla \ell_t(W_{t+1}) - \nabla \ell_{t-1}(W_t) + \alpha(W_{t+1} - \bar{W}_t) &= 0 \\ \bar{W}_{t+1} = \frac{1}{2}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha}\nabla \ell_t(W_{t+1}) & \end{aligned} \tag{5.4}$$

Let us see why these equations allow for reaching a stationary point of Eq. 5.1. For now suppose the sequence W_t converges to a limit point W_∞. One way to ensure this happens is to view Eq. 5.4 as a map from $[W_t, \bar{W}_t]^\top \to [W_{t+1}, \bar{W}_{t+1}]^\top$, and show that this map is contractive, and as such invoke the Banach fixed point theorem. Nevertheless, this is difficult to show and we assume that it is true for now.

Proposition 5. *In the Algorithm 3, suppose, the sequence W_t is bounded and converges to a limit point W_∞. Further assume the loss function ℓ_t is smooth and Lipschitz. Let the cumulative loss be $F = \frac{1}{T}\sum_{t=1}^{T} \nabla \ell_t(W_\infty)$ after T iterations [2]. It follows that W_∞ is a stationary point of Eq. 2.3, i.e., $\lim_{T\to\infty} \frac{\partial F}{\partial W}(W_\infty) = 0$.*

[2]For simplicity in exposition, we concatenate all the losses ℓ_t into a single online stream and get rid of the index N, that gets repeated to provide T iterations of the gradient updates.

We sketch the proof below (see Sec. D.7 for detailed proof). Rewriting the first equation in Eq. 5.4 as: $W_{t+1} = \bar{W}_t + \frac{1}{\alpha}(\ell_{t-1}(W_t) - \ell_t(W_{t+1}))$, we note that if $W_{t+1} \to W_\infty$, then invoking Cesaro mean[3] argument, the corresponding averages do as well: $\frac{1}{T}\sum_{t=1}^{T} W_{t+1} \overset{T\to\infty}{\longrightarrow} W_\infty$. In turn, we note that the second term in the above expression telescopes, and consequently, it follows that $\frac{1}{T}\sum_{t=1}^{T} W_{t+1} = \frac{1}{T}\sum_{t=1}^{T} \bar{W}_t - \frac{1}{T}\nabla\ell_T(W_{T+1})$. Now under smoothness and Lipschitz conditions, we can assume that $\frac{1}{T}\nabla\ell_T(W_{T+1}) \to 0$ in all of its components. As a result, we have $\frac{1}{T}\sum_{t=1}^{T} \bar{W}_t \to W_\infty$ as well. Plugging these facts into the second equation, we get $\frac{1}{2\alpha T}\sum_{t=1}^{T} \nabla\ell_t(W_{t+1}) \approx 0$. Now we also know that $W_{t+1} \approx W_\infty$ for sufficiently large T, and using standard arguments it follows that $\frac{1}{2\alpha T}\sum_{t=1}^{T} \nabla\ell_t(W_\infty)$ also approaches zero. This is the proposed stationarity condition, and our claim follows.

Computational Complexity. BPTT gradient cost scales as $\Omega(T)$ as seen from Eq. 2.3. Although FPTT for T times steps leads to $\Omega(T)$ gradient computations, but it is worth noting that the constants involved in taking gradient for the full length T are higher than computing single step gradients. However, FPTT has more arithmetic operations per gradient step, and as such a tradeoff exists. BPTT has higher memory overhead since it stores intermediate hidden states for the full-time horizon T. In contrast, since FPTT only optimizes instantaneous loss functions, it does not require storing hidden states for the full-time horizon. We list computational complexities of different algorithms in Table 5.1.

FPTT-K. Instead of updating parameters at every timestep, we could perform updates in Eq. 5.2 only K times for the sequence length T. To do this, for each example, we consider a window of size $\omega = \lfloor \frac{T}{K} \rfloor$, and define a windowed loss, $\bar{\ell}_{t,\omega}(W) = \frac{1}{\omega}\sum_{\tau=t-\omega}^{t} \ell_t(W)$. We then loop this over K steps instead of T. Setting $K = 1$ is the same as learning RNN through BPTT, while $K = T$ results in the FPTT 3. We provide ablative experiments to study the effect of this parameter on learning

[3] https://www.ee.columbia.edu/~vittorio/CesaroMeans.pdf

efficiency in Section 5.3.

Table 5.1: Per-instance computational cost for gradient, parameter update & memory storage overhead. Parameter update involves several arithmetic operations (see Algo. 3), exceeding cost of gradient update by a constant factor. Note that constant associated with gradient computation is a monotonically increasing function $c(\cdot)$ of the sequence length, i.e. $c(1) < c(K) < C(T)$.

Algorithm	**Gradient Updates**	**Parameter Updates**	**Memory Storage**
BPTT	$\Omega(c(T)T)$	$\Omega(1)$	$\Omega(T)$
FTRL	$\Omega(c(T)T^2)$	$\Omega(T)$	$\Omega(T)$
FPTT	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$
FPTT-K	$\Omega(c(K)T)$	$\Omega(K)$	$\Omega(T/K)$

Intermediate Losses for Terminal Prediction. In our exposition so far, we assumed that we have access to an instantaneous loss ℓ_t at timestep t. While this is true for Seq-to-Seq modelling tasks, for terminal prediction tasks we only get one label y for the entire input sequence x. Let $\hat{P} = \text{softmax}(v^\top h_t)$ be our current estimate of label distribution and Q be our estimate in last training epoch. We use cross-entropy for the classification loss. We construct intermediate losses ℓ_t for anytime step t as a convex combination of two terms : (a) cross-entropy using the current label distribution $\hat{P}$, and (b) divergence like term to enforce $\hat{P}$ and Q to stay close by. This results in the following loss:

$$\ell_t = \beta \ell_t^{CE} + (1-\beta)\ell_t^{Div}$$
$$\ell_t^{CE} = -\sum_{\bar{y}\in\mathcal{Y}} \mathbf{1}_{\bar{y}=y} \log \hat{P}(\bar{y}); \;\; \ell_t^{Div} = -\sum_{\bar{y}\in\mathcal{Y}} Q(\bar{y}) \log \hat{P}(\bar{y})$$

where $\beta \in [0,1]$. Our intuition is that for timesteps near T, classification loss is weighted more and less to the divergence term. In the beginning, we should have much less confidence in the classification loss and more on the divergence term. This leads to a natural choice of $\beta = \frac{t}{T}$ achieving the desired effect.

Extensions to Stacked/Hierarchical RNNs. There can be various extensions

of our scheme to multi-layered RNNs. One simple scheme is to treat the transition in the stacked RNN as a multi-layered function which transforms hidden states from one time step to the next. Our language modelling experiments (Sec. 5.3.3) on PTB-word and character level uses this extension for the 3-layered stacked LSTM models. Note that ideally such an extension should work for hierarchical RNNs as well as state transitions can be seen at the most frequent update equation. We leave this as a potential future direction.

5.3 Experiments

In this section we empirically demonstrate that the proposed algorithm outperforms BPTT. First, we provide ablative experiments to justify our default choice of hyper-parameters and the chosen architecture. Next, we run FPTT on sequence-to-sequence modelling tasks. Finally, we benchmark FPTT on terminal prediction tasks which provide the true label only for the full input sequence.

5.3.1 Experimental Setup

We implement FPTT in Pytorch using the pseudo code given by Algorithm 3. We perform our experiments on single GTX 1080 Ti GPU. The benchmark datasets used in this study are publicly available along with a train and test split. For hyper-parameter tuning, we set aside a validation set on tasks where a validation set is not available. Wherever applicable we use grid search for tuning hyper-parameters (details in supplementary). In our experiment, we use LSTM(Hochreiter and Schmidhuber, 1997b) as the default RNN architecture for evaluation purpose. They have been shown to suffer from vanishing/exploding gradients on many tasks (Zhang et al., 2018a; Chang et al., 2019; Kag et al., 2020). Our reasoning follows from the fact that they are widely available with most efficient CUDA implementation on many popular

deep learning libraries. This also reduces our experimentation cost. Although we use LSTMs to show that FPTT works on many benchmark datasets, we provide ablative study to show that FPTT works on many RNN architectures (see Sec. 5.3.2).

Since many RTRL algorithms do not scale to the large-scale benchmark tasks, we do not show their performance in our results. TBPTT has been shown to perform poorly in comparison to BPTT (Trinh et al., 2018). Hence, we will only consider BPTT as the baseline and add the prefix FPTT whenever RNNs are trained with the proposed algorithm, otherwise the training algorithm is assumed to be BPTT. Wherever applicable we will include known results from the literature to compare our BPTT implementation.

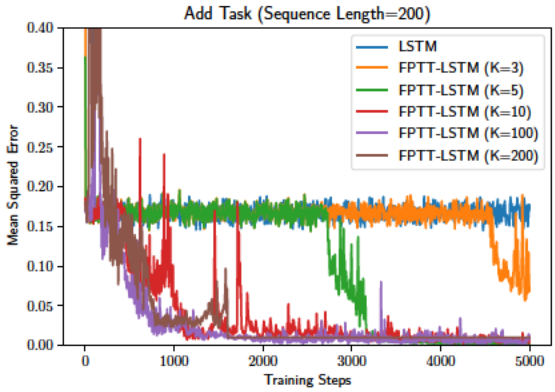


Figure 5·2: Ablative Experiment: Add Task ($T = 200$) solved by splitting in multiple parts. Note that FPTT with $K = 1$ corresponds to BPTT for LSTM while $K = 200$ updates W_t at every timestep. This figure demonstrates as K increases the performance of the algorithm improves.

5.3.2 Ablative experiments

Below ablative experiments highlight key aspects of FPTT.

Effect of the parameter K. As described in the Sec. 5.2 each round of RNN parameter update is more expensive than gradient update. We can address this issue

by choosing a suitable K and run FPTT-K. Larger K decreases number of parameter updates, but can impact convergence. For the Add-Task with sequence length $T = 200$, we learn LSTMs with different K values (ranging from $K = 1, 3, 5, 10, 100, 200$). Note that $K = 1$ is essentially BPTT as we only update the RNN parameter after seeing the entire sequence. Figure 5·2 shows that higher K values result in better convergence. On the other hand higher values of K are more expensive since, as described in Table 5.1, cost of parameter updates exceeds gradient by a constant factor[4]. Suppose this factor is C^2, the highest computational efficiency is achieved by FPTT-K with $K = \lfloor \frac{T}{C} \rfloor$. On the other hand small values of K could lead to poor training as observed in Figure 5·2. As a rule of thumb we use $K \approx \sqrt{T}$ for all our other experiments. This results in meaningful performance (trainability), and computational efficiency matching GPU LSTM implementation.

Table 5.2: CIFAR-10 : Different RNN architectures.

	Accuracy	**#Params**
LSTM	60.11%	67K
FPTT LSTM	**71.03%**	67K
GRU	66.28%	51K
FPTT GRU	**71.37%**	51K
Antisymmetric	62.41%	37K
FPTT Antisymmetric	**72.13%**	37K

Choice of architecture. In this experiment we show that FPTT provides non-trivial gains for many RNN architectures. We train one layer LSTM, and GRU architectures on the CIFAR-10 dataset with the same setting as the described in section 5.3.4. Table 5.2 shows that RNNs trained with FPTT provide gains of about $5-10$ points in accuracy over the RNNs trained with BPTT. In the remaining experiments, we reduce experimentation cost by only performing evaluations on LSTMs as they are readily available in PyTorch with very efficient CUDA implementation. We also want to show that replacing BPTT with FPTT allows LSTMs to achieve

[4]This factor is difficult to pin-down since gradients leverage CUDA-pytorch, while parameter updates are not optimized.

performance near state-of-the-art performance achieved by recent architectural improvements.

Auxiliary Losses in BPTT vs FPTT. In this experiment we augment BPTT with the auxiliary losses (proposed for terminal prediction in section 5.2) similar to (Trinh et al., 2018) in order to isolate the gains from auxiliary losses in the BPTT routine. Table 5.3 shows that our auxiliary losses helps BPTT to improve the performance but still lack behind the proposed algorithm.

Table 5.3: CIFAR-10 : BPTT+Auxiliary Loss vs FPTT.

	Accuracy	**#Params**
LSTM	60.11%	67K
Aux-Loss+LSTM	65.65%	67K
FPTT LSTM	**71.03%**	67K

Sensitivity to α hyper-parameter. Note that very small value of α, i.e. $\alpha \to 0$ would lead FPTT to ignore the regularizer and would only optimize the instantaneous loss at every step resulting in diverging iterates. While very high value of α would lead FPTT to only optimize the regularizer and hence very poor generalization performance. We explore the sensitivity to the α hyper-parameter in the Algorithm 3. We use the PTB-300 language modelling dataset. In this experiment we train FPTT on the following α values: $\{1.0, 0.8, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. Best perplexity is reached at $\alpha = 0.5$ while $\alpha = 1.0$ fails to converge to a good solution. Also, the performance starts to decrease with $\alpha \leq 0.05$. We show the full result in the appendix (see Table D.2 in Sec. D.3).

5.3.3 Sequence Modelling

We perform experiments on three variants of the sequence-to-sequence benchmark Penn Tree Bank (PTB) dataset (McAuley and Leskovec, 2013). We provide full details of these experiments in the supplementary.

PTB-300 is a word level language modelling task with the difficult sequence length of 300 and has been studied in many previous works to study long range dependencies in language modeling (Zhang et al., 2018a; Kusupati et al., 2018; Kag et al., 2020). Table 5.4 shows the test perplexity for our experimental runs along with results from earlier works. Note that improved architectures such as FastGRNN (Kusupati et al., 2018), SpectralRNNs(Zhang et al., 2018a), IncrementalRNNs (Kag et al., 2020) show improvements over the LSTMs trained using backpropagation algorithm. By incorporating FPTT as the training algorithm we improve LSTM's test perplexity by nearly 11 points and thus outperforming the reported LSTM results.

Table 5.4: Results for PTB word level language modelling : Sequence length (300), 1-Layer LSTM.

Dataset	**PTB-w**	
	Perplexity	**#Params**
FastGRNN (Kusupati et al., 2018)	116.11	53K
IncrementalRNN (Kag et al., 2020)	**115.71**	30K
SpectralRNN (Zhang et al., 2018a)	130.20	31K
LSTM (Zhang et al., 2018a)	130.21	64K
LSTM (Kusupati et al., 2018)	**117.41**	210K
LSTM	117.09	210K
FPTT LSTM	**106.27**	210K

PTB-w is the traditional word level language modelling variant of the PTB dataset. It uses 70 as sequence length and we follow (Yang et al., 2018) to setup this experiment. We use three-layer LSTM model for this task with embedding dimensions 280 and hidden size 1150. We report the results with dynamic evaluation(Krause et al., 2018) on the trained model. We use the same architecture and training setup to train LSTMs with both BPTT and FPTT. Table 5.5 demonstrates that LSTM trained with FPTT result in better performance than the ones trained with BPTT.

PTB-c is the character level modelling task that uses 150 sequence length. We utilize (Merity et al., 2018) to setup the character level task. We use 3-layer LSTM

models as recommended with hidden size 1000 and embedding dimension 200. We train this model with both BPTT and FPTT with the same setting. As shown in Table 5.5, LSTM trained with FPTT results in better bits-per-characters and has comparable performance with existing state-of-the-art results present in this table.

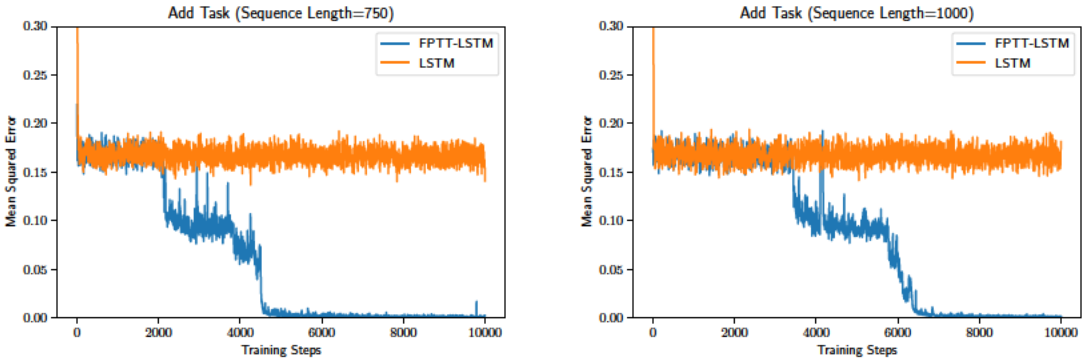
Table 5.5: Results for PTB-w and PTB-c datasets. We use AWD-LSTM model in our PTB-c experiments and AWD-LSTM with Mixture-of-Softmaxes(Yang et al., 2018) in the PTB-w experiments. For PTB-w dataset, wherever applicable, all the baselines report the results with dynamiceval(Krause et al., 2018). It can be seen that training with FPTT outperforms the model trained with BPTT.

Dataset	**PTB-c**			**PTB-w**		
	Hidden Dim.	**BPC**	**#Params**	**Hidden Dim.**	**Perplexity**	**#Params**
Trellis-Net (Bai et al., 2019b)	1000	**1.158**	13.4M	1000	54.19	34M
AWD-LSTM (Merity et al., 2018)	1000	1.175	13.8M	1150	51.1	24M
Dense IndRNN (Li et al., 2019b)	2000	1.18	45.7M	2000	**50.97**	52M
LSTM	1000	1.183	13.8M	1150	51.9	22M
FPTT LSTM	1000	**1.165**	13.8M	1150	**50.96**	22M

5.3.4 Terminal Prediction

We benchmark FPTT on popular terminal prediction tasks to demonstrate that the proposed algorithm provides non-trivial gains over BPTT in this setting as well. For fair comparison, following previous works(Zhang et al., 2018a; Kusupati et al., 2018; Kag et al., 2020), we use LSTMs with 128 dimensional hidden state and Adam as the choice of optimizer with initial learning rate $1e-3$ for both algorithms. We provide other hyper-parameter tuning details in the supplementary (see Sec. D.1).

Add-Task (Hochreiter and Schmidhuber, 1997b) has been used to evaluate long range dependencies in RNN architectures. An example data point consists of two sequences (x_1, x_2) of length T and a target label y. x_1 contains real-valued entries drawn uniformly from $[0, 1]$, x_2 is a binary sequence with exactly two 1s, and the label y is the sum of the two entries in sequence x_1 where x_2 has 1s. For both the algorithms (BPTT and FPTT), we use episodic training where a train batch size of 128 is presented to the RNN to update its parameters and evaluated using



(a) **(b)**

Figure 5·3: Results for Add Task with large sequence lengths : (a) $T = 750$, and (b) $T = 1000$.

an independently drawn test set. We use difficult sequence lengths $T = 750$ and $T = 1000$ in this task.

Figure 5·3 shows the convergence plots for both the algorithms on these two settings. This demonstrates that FPTT helps LSTMs solve this task while BPTT stays around the same loss value throughout the training phase. Note that this observation is consistent with previous works (Kag et al., 2020; Zhang et al., 2018a).

Table 5.6: Results for Sequential MNIST, Permute MNIST and Sequential CIFAR-10. Models listed below use 1-Layer except IndRNN and TrellisNet as they are multi-layered architectures. Legend Acc. stands for Accuracy of the method.

Dataset	**Seq-MNIST**		**Permute-MNIST**		**CIFAR-10**	
	Acc.	**Params**	**Acc.**	**Params**	**Acc.**	**Params**
AntisymmetricRNN (Chang et al., 2019)	98.8%	10K	93.1%	10K	62.20%	37K
IncrementalRNN (Kag et al., 2020)	98.13%	4K	95.62%	8K	-	-
IndRNN (6 Layers) (Li et al., 2018b)	99.0%	-	96.0%	-	-	-
TrellisNet (16 Layers) (Bai et al., 2019b)	**99.20%**	8M	**98.13%**	8M	**73.42%**	8M
r-LSTM (Trinh et al., 2018)	98.4%	100K	95.2%	100K	72.20%	101K
LSTM (Chang et al., 2019)	97.3%	68K	**92.6%**	68K	**59.70%**	69K
LSTM (Trinh et al., 2018)	**98.3%**	100K	89.4%	100K	58.80%	101K
LSTM (TBPTT-300) (Trinh et al., 2018)	11.3%	100K	88.8%	100K	49.01%	101K
LSTM	97.71%	66K	88.91%	66K	60.11%	67K
FPTT LSTM	**98.67%**	66K	**94.75%**	66K	**71.03%**	67K

Pixel & Permute MNIST, CIFAR-10 are sequential variants of the popular image classification datasets: MNIST (Lecun et al., 1998) and CIFAR-10 (Krizhevsky and Hinton, 2009). MNIST consists of images of 10 digits with shape $28 \times 28 \times 1$, while

CIFAR-10 consists of images with shape $32 \times 32 \times 3$. The input images are flattened into a sequence (row-wise). At each time step, 1 and 3 pixels are presented as the input for MNIST and CIFAR datasets respectively. This construction results in Pixel MNIST and CIFAR datasets with 784 and 1024 length sequences respectively. While Permute-MNIST is obtained by applying a fixed permutation on the Pixel MNIST sequence. This creates a harder problem than the Pixel setting since there are no obvious patterns to explore.

Table 5.6 lists the performance of LSTMs trained using BPTT and FPTT, along with the known results in the literature on these datasets. This shows that LSTMs trained with FPTT outperforms the ones trained with BPTT. We point out that FPTT LSTMs performance is reasonably close to the best performance reported on this dataset with better architectures and higher complexity.

Below we list benefits of the proposed algorithm.

(A) Better Generalization. *FPTT provides better generalization on many benchmark tasks compared to BPTT.* Tables 5.4, 5.5, and 5.6 shows that FPTT yields better test performance as compared to training with BPTT. Note that table 5.2 shows similar gains in architectures other than LSTMs.

(B) Learning Long Term Dependency tasks. *Training with FPTT enables LSTMs to solve LTD tasks.* Our experiments evaluate FPTT on many LTD datasets (Add-Task, Permute/Pixel MNIST, CIFAR-10 and PTB-300). Figure 5·3 shows that FPTT enables LSTMs to solve Add-Task while BPTT was unable to solve this task. Similarly, tables 5.4, and 5.6 shows that FPTT outperforms BPTT on LTD tasks.

(C) Better Model Efficiency. *FPTT trained LSTMs compete with higher complexity models.* Table 5.6 shows our 1-layer LSTMs are competitive with multi-layered deep RNN higher capacity models (TrellisNet(Bai et al., 2019b), IndRNN(Li et al., 2018b)). In retrospect it is worth considering that the higher complexity models

have often stemmed from inability to train LSTMs on long-term dependency tasks. FPTT points to the fact that the issue is not capacity but the training method.

(D) Learning Short-term dependency tasks. *FPTT shows competitive performance on sequence modelling tasks.* Our experiments on language modelling tasks with shorter sequence lengths (PTB-w, PTB-c datasets) demonstrates the proposed method can learn short term dependencies present in these datasets. Table 5.5 shows that FPTT provides better test performance than BPTT.

(E) Computational Efficiency/Convergence. We discussed the computational trade-off of the proposed method in table 5.1. This trade-off allows FPTT to provide training complexity similar to BPTT while providing better statistical trainability and generalization. We show training time comparison in the supplementary (see Sec. D.2). Additionally, FPTT reduces the memory overhead by only storing hidden states between two iterate updates, in contrast BPTT stores all the intermediate states in the time horizon T.

5.4 Discussion

While our theoretical analysis only provides weak guarantees in that if the parameter sequence converges, the convergent point would be the stationary point for the BPTT loss. It also requires an assumption that the sequence length is large. We can extend this analysis by only looking at the online update and analyzing the regret in a similar manner as FedDyn paper(Acar et al., 2021). Similarly, we can piggyback on the generalization bounds (Kuznetsov and Mohri, 2015; Kuznetsov and Mohri, 2016) for non-stationary non-mixing stochastic processes (such as time-series forecasting problem) for generalization analysis under mild assumptions on the input sequential process.

From an algorithmic viewpoint, there can be multiple extensions of the proposed

FPTT scheme. Currently, we use a single data point to move forward in time and update parameters in this manner. This setup is very similar to the online learning setup. Instead, we can also look at all the input data per time step and update parameters per time step. It is a bit different strategy than the proposed scheme. Similarly, many other variants can be derived. We leave the exercise of finding the best scheme for future research. Another line of thought is the extension of this scheme to transformer training. The bigger challenge that FPTT would encounter running into transformers is that they do not operate at a time step level. It would be interesting to ask what modifications would FPTT require to be applicable for transformer training.

Part II

Efficient Low Complexity CNNs

Chapter 6

Convolutional Neural Network: Background

Convolutional neural networks (CNNs) have been the backbone for recent advances in image recognition (Krizhevsky et al., 2012; Howard et al., 2019; Tan and Le, 2019), object detection (Zhao et al., 2019; Jiao et al., 2019), and other applications (Wang et al., 2020) interfacing the image modalities. At the heart of these architectures lie basic building blocks such as residual blocks and other variants. Researchers have designed resource-efficient convolutional blocks that yield a better trade-off between performance and resource usage. For instance, Inverted Residual Block (Sandler et al., 2018) and its derivatives have been used in many efficient CNN architectures such as MobileNetV3(Howard et al., 2019), EfficientNet(Tan and Le, 2019), MNASNet(Tan et al., 2019), etc.

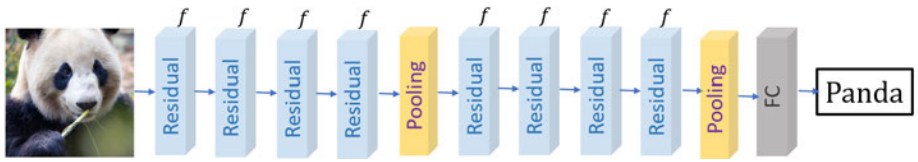
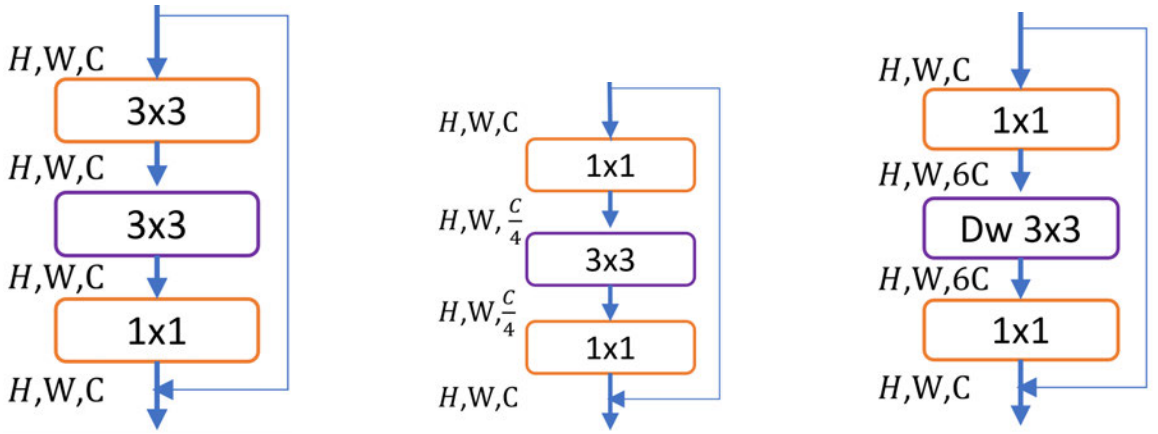


Figure 6·1: Generic CNN Architecture. We show a generic CNN architecture composed of multiple repetitions of the convolutional residual block f. Note that f can be replaced by many popular residual blocks such as Bottleneck(He et al., 2016) or Inverted Residual(Howard et al., 2019).

6.1 Generic Convolutional Architecture

We show a generic convolutional neural network in Figure 6·1 that uses the convolutional residual layer f. These residual blocks take a feature map of size $H \times W \times C$ as input, where C denotes the number of channels, H, and W denotes the height and width of the feature maps. Residual block f applies a few non-linear operations and outputs the resulting feature map. For simplicity, we will assume that the input and output activation maps have the same size. Different choices of the residual block f yield different convolutional architectures. For instance, f = basic and bottleneck blocks (Figure 6·2.a, b) results in ResNet(He et al., 2016) architecture, while inverted residual block (Figure 6·2.c) yields MobileNetV3(Howard et al., 2019) and EfficientNet(Tan and Le, 2019) architectures.



(a) Basic (b) Bottleneck (c) Inverted Residual

Figure 6·2: **(a)**: Basic Residual Block. **(b)**: Bottleneck Residual Block. **(c)**: Inverted Residual Block. Acronyms are as follows: 3×3 (Full 3-d Convolution with 3×3 kernel), 1×1 (Pointwise projection), Dw 3×3 (Depthwise Convolution, followed by Squeeze-and-Excitation). Note that after the each convolutional operation (such as 3×3, 1×1 and Dw 3×3), there is a batch-norm followed by a non-linear activation such as ReLU(Nair and Hinton, 2010), Swish (Tan and Le, 2019) or Hardswish (Howard et al., 2019) except the last 1×1 where only a batch-norm is applied.

6.2 Resource-Efficiency Challenges

Convolutional models suffer from many issues both from the architectural design perspective as well as training. Below, we list issues that are at the heart of the next few chapters.

1. **Low Receptive Field.** Convolutional filters with limited receptive fields (such as 3×3) act on localized input regions to generate low-level features. Features used for decision-making are complex functions of these low-level features, achieved through the composition of many convolutional operators applied in sequence. It requires successive application of such convolutional blocks to yield rich features for downstream applications, resulting in very deep neural networks. Thus, resulting CNNs have large resource footprints, such as high inference/train time and large model size.

   While it is possible to increase the convolutional kernel size significantly, such a change significantly increases the storage and computational resource requirements. For instance, let us apply a full 3-d convolution with kernel $K \times K$ on the input feature map of size $H \times W \times C$ to output a feature map of similar size. We will require $\mathcal{O}(K^2C^2)$ storage (number of parameters) and $\mathcal{O}(HWC^2K^2)$ compute (multiply-add operations). It shows that the effect of kernel size is quadratic on the storage and compute requirements. As a result, for large K ($\gg 3$), this cost becomes excessive and not feasible for deployment in CNNs with real-time inference.

2. **Exploiting Spatial Smoothness.** Typically, convolutional blocks compute high-dimensional activation maps to capture key features while being computationally inexpensive. For example, the *Inverted Residual Block* (see Figure 6·2) projects the input onto a high dimensional space, and performs a depth-wise convolution, followed by a projection onto a lower dimension space. Thus they compute high-dimensional features without paying for full 3-d convolutions, with the intuition

that such a “low-rank" structure is enough to capture the key features.

However, such methods typically do not exploit any redundancies in the spatial features. The key observation of our work is that spatially features are quite “smooth". That is, while the features of each pixel might be different, they tend to be “close" to each other. It enables accurate predictions without paying for expensive computations for each pixel.

3. **Promoting Inductive Biases.** By design, CNNs promote many inductive biases. For instance, translation invariance, i.e., convolutional kernels operate as local feature extractors in the sense that they search for some template over the entire input. Many such properties have been integrated into the design phase (such as residual skip connections leading to wide minima(Li et al., 2018a) of the training loss as well as less prone to vanishing/exploding gradients(He et al., 2016); wide convolutional layers lead to better generalization (Doimo et al., 2022)). While these biases are a function of architectural design, less emphasis has been given to inducing such inductive biases during training.

6.3 Related Works

6.3.1 CNN Architectures

Early Skip CNNs. Resnet(He et al., 2016), Highway networks(Srivastava et al., 2015), Wide-Resnets (Zagoruyko and Komodakis, 2016), Dense-Nets (Huang et al., 2017), etc. proposed networks with skip/residual connections. These changes helped alleviate the vanishing gradient issues in the deep neural networks. These trained much deeper models and hence achieved significantly better performance than the previous generation models like AlexNet (Krizhevsky et al., 2012), VGG-Nets (Simonyan and Zisserman, 2015), etc. Note that deeper models implicitly mean larger storage costs and higher compute requirements.

Mobile/IoT ready CNNs. Many initial attempts (SqueezeNet (Iandola et al., 2016), SqueezeNext (Gholami et al., 2018), MobileNets (Howard et al., 2017)) at designing low complexity models included handcrafted feature blocks (with low rank filters, separable convolutions, etc.) whose composition yielded small models with low floating-point operations. Recently, EfficientNets (Tan and Le, 2019) were proposed to systematically study the effect of width, depth, channels, etc., along with memory and MACs constraints. There have also been efforts (Liu et al., 2017a; Zoph and Le, 2016) to search for neural architectures that outperform hand-crafted architectures. Note that these are complementary to our proposal.

Residual Blocks. ResNet (He et al., 2016) proposed the identity shortcuts wherein the input is added to the output of the feature processing branch, but full 3-d convolutions can be computationally expensive. MobileNet (Howard et al., 2017) replaced the 3×3 convolution with a 3×3 depth-wise convolution followed by a 1×1 projection. Highway Networks (Srivastava et al., 2015) used learned gating functions for shortcut connections.

MobileNetV2 (Sandler et al., 2018) proposed an Inverted Residual Block wherein they inverted the residual connection in ResNet's Bottleneck block. It consists of a 1×1 high dimensional projection, followed by 3×3 depth-wise convolution, and finally 1×1 low dimensional projection. In parallel, ShuffleNets (Zhang et al., 2018b; Ma et al., 2018) introduced groups in the 1×1 projection step and channel shuffle operation to bridge the performance gap between the full projection step. Note that shuffle operations are non-trivial to implement in general-purpose hardware.

EfficientNetV2 (Tan and Le, 2021) introduced Fused Inverted residual blocks which replace the projection and depth-wise convolution with a full convolution. They demonstrate better throughput on GPU / TPU hardware as Inverted Residual blocks do not effectively leverage the accelerators. That is, the existing techniques mostly

focus on efficient computation despite the presence of a large number of channels.

In contrast, the proposed interpolation scheme focuses on the spatial smoothness of activations and is complimentary, and generic enough to be applicable to most existing residual blocks.

Neural Architecture Search (NAS). Several works have introduced techniques to search for efficient architectures using above mentioned CNN blocks as building blocks. There are three main approaches for NAS in the literature: 1) reinforcement learning approaches (Zoph and Le, 2016; Zoph et al., 2018) where a controller learns to generate new architectures based on the delayed rewards through REINFORCE rule, 2) evolutionary search algorithms (Real et al., 2019; Liu et al., 2017a; Dai et al., 2021) proceed by starting at some initial parent architectures and evolve into children architectures through mutations (adding or deleting operations), 3) differentiable architecture search methods (Liu et al., 2019a; Wu et al., 2019; Cai et al., 2019; Dong and Yang, 2019) that create a super-network as a combination of all the possible operations at a block level, and use standard SGD style techniques to find the sparse set of weights for each operation/block in the super-network. NAS is an orthogonal direction to our work and can be used to further optimize the hyper-parameters of our spatially interpolated blocks.

Model Compression/Distillation. An alternate strategy to obtain small models require model compression. Deep-compression (Han et al., 2016) is an early work where a pre-trained network is pruned, quantized, and compressed to yield small networks which can be deployed on the edge devices. Other works include distilling (Hinton et al., 2015) knowledge from a larger pre-trained network into a small compact model. We do not pursue these techniques to simplify our exposition.

ODE Inspired CNNs. Related work in this class has the most similarity with our approach. Neural ODEs (NODEs) (Chen et al., 2018) introduce continuous time

layers following an ODE. It uses black-box ODE solvers along with the adjoint method for back-propagation. Augmented Neural ODEs (Dupont et al., 2019) extend NODEs to a richer class of functions while ANODE (Gholami et al., 2019) addresses the gradient computation in the adjoint method to allow for more accurate gradients matching the discretization. Neural ODEs and their variants do not demonstrate their scalability to tasks like ImageNet.

There have been previous works that utilize ODE-inspired models for sequential processing. Some of these models require the architecture to achieve equilibrium (Kag et al., 2020; Kag and Saligrama, 2021a), (Bai et al., 2019a), where the later has been extended to image modalities by Multi-scale deep equilibrium models (MDEQs) (Bai et al., 2020). MDEQs use implicit layers at multiple feature scales to scale to large datasets such as ImageNet. Although they show good performance on large-scale tasks, the model capacity still needs to be nearly the same as the discrete counterparts. Although implicit models offer low memory cost training, they still do not offer much flexibility for inference. In addition, their inference cost is much higher in comparison to discrete variants such as Resnet.

PDE Inspired CNNs. (Ruthotto and Haber, 2020) proposed new architectures based on parabolic and hyperbolic partial differential equations. These connections with PDEs enable theoretical reasoning, such as the stability of the resulting network. Although the resulting models are small, these models take a significant hit in performance. NeuPDE (Sun et al., 2020) uses the convolutional filters to approximate the differential operators for generic second order PDE. NeuPDEs downsample the input image to a manageable feature map through many convolutional layers and then finally applies the PDE blocks. This construction helps in reducing the model size but the gains are not sufficient enough.

Miscellaneous. There are several general techniques to reduce the resource foot-

print of any architecture. It includes weight quantization (Han et al., 2016; Gholami et al., 2021), network pruning (Dong and Yang, 2019; Han et al., 2016; Liang et al., 2021), and distillation (Hinton et al., 2015; Gou et al., 2021). Finally, progressive learning (Tan and Le, 2021) trains models by progressively increasing image sizes. Note that these techniques work with any CNN block and can be applied directly to our spatially interpolated blocks. For simple exposition and to clearly demonstrate the impact of our technique over SOTA CNN blocks, we use these blocks in the original form without any quantization/distillation, etc.

6.3.2 Training Algorithms

Consistency Loss. Prior works on semi-supervised learning have proposed consistency loss (Volpi et al., 2018; Xie et al., 2020; Sohn et al., 2020; Laine and Aila, 2017; Sajjadi et al., 2016) to inductively bias model predictions to be consistent under reasonable modifications to the input. (Sajjadi et al., 2016) predicts the same label amidst stochasticity in dropout, random max-pooling, and randomized affine data augmentations. (Volpi et al., 2018) preserves predictions in the presence of perturbations such as adversarial noise. (Xie et al., 2020) improves upon these works to include advanced data augmentations (such as RandAugment(Cubuk et al., 2020)) as multiple viewpoints of the underlying input across which the model prediction remains consistent. (Sohn et al., 2020) uses weak augmentation to preserve the label information and maintains consistency by penalizing the model prediction on strong data augmentation if it differs from weak augmentation prediction. Similarly, Π model (Laine and Aila, 2017) penalizes when the model predictions differ between random data augmentations and dropout for the same input, while Temporal Ensembling (Laine and Aila, 2017) uses an average of predictions for various viewpoints of the data during the training trajectory and this prediction is used as the target label.

The survey (Yang et al., 2022) presents an in-depth review.

Contrastive Loss. Contrastive learning(Jaiswal et al., 2021) brings closer various alternate views of the data point and maximizes the distance between the negative of the data point. (Schroff et al., 2015) designed a triplet loss function, given an image (called the anchor), finds positive and negative views w.r.t. the anchor and minimizes the distance between positives and maximizes the distance between negatives. Since triplet loss only uses one negative example in each update, it suffers from slow convergence. (Sohn, 2016) addresses this issue by using $N-1$ negatives and one positive example and using a modified softmax loss to model this N way relationship with the anchor data point. (Gutmann and Hyvärinen, 2010) proposed the Noise Contrastive Estimation (NCE) to estimate model parameters by learning a logistic regression model to distinguish observed data from artificially generated noise. (van den Oord et al., 2019) introduced the InfoNCE loss by extending NCE to include multiple negatives. Note that the quality of the learned solution improves by using more negative examples. (Chen et al., 2020a) uses a large batch size to generate sufficient negatives (in a sense, mining hard negatives for the data point). Others(Wu et al., 2018; Tian et al., 2020) have proposed using a separate memory bank to address this issue.

Wide Minima. In parallel, (Tarvainen and Valpola, 2017; Grill et al., 2020; Caron et al., 2021; Hochreiter and Schmidhuber, 1997a; Foret et al., 2021; Cha et al., 2021; Chen et al., 2020b; He et al., 2020) have also explored variance arising due to different model views. Mean-Teacher(Tarvainen and Valpola, 2017) regularizes model predictions to be close to the teacher predictions, which is an exponential moving average (EMA). Self-supervised learning (Caron et al., 2021; Grill et al., 2020) explores a similar penalty in the absence of ground truth. Typically, the regularization minimizes the difference between different views of the input forwarded through the EMA

and model. Although this requires careful design considerations to avoid representation collapse, such a strategy successfully learns resilient latent feature space without labels. In parallel, we can also view model invariability from the flat/wide minima (Hochreiter and Schmidhuber, 1997a) perspective in the parameter space. Many works have promoted this notion, such as sharpness-aware minimization(Foret et al., 2021), averaging of the stochastic gradient models(Izmailov et al., 2018), etc.

Chapter 7

Global Layered Convolutional Neural Networks (PDE-CNNs)

7.1 Introduction

Convolutional neural networks (CNNs) have been the backbone for recent advances in image recognition (Krizhevsky et al., 2012), object detection (Zhao et al., 2019), and other applications (Wang et al., 2020) interfacing the image modalities. Convolutional filters with limited receptive fields act on localized input regions to generate low-level features. Features used for decision-making are complex functions of these low-level features, achieved through the composition of many such convolutional operators applied in sequence, resulting in deep networks with high inference/train time and large model size.

Recent works (Chen et al., 2018; Gholami et al., 2019) have explored neural networks inspired by ordinary differential equations (ODEs), offering richer representation than their discrete counterparts. Resnets (He et al., 2016) can be viewed as a discretized form of ODEs. The final architecture based on these continuous layers leads to higher computational cost in comparison to their discrete counterpart (Bai et al., 2020), namely due to the costly fixed point solvers. In contrast, we explore novel constraints on the feature maps, based on partial differential equations (PDEs) that offer similar rich representation but with shallower neural networks. In addition,

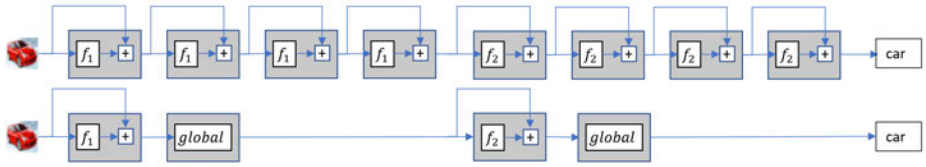


Figure 7·1: Replacing repeated blocks in a given CNN architecture with the Global layer for compute and model savings.

we provide efficient and scalable solvers to provide computational and storage savings.

Proposed Method. We explore a hybrid approach wherein we modify discrete models by embedding a new layer with a global receptive field that operates on the input feature map and computes complex compositions of these low-level features. We call this layer the Global feature layer. It approximately solves a PDE constraint that couples the input and output feature maps. In a typical discrete model, at every input resolution, the same convolutional block is applied repeatedly m times. We modify this structure by keeping only one convolutional block and replacing the $m-1$ blocks with a single global feature layer (see Figure 7·1). Thus, reducing the deep neural network to a much shallower network without any significant performance loss. It leads to smaller models with low computational and storage costs. In addition, it improves both the train and inference times.

By keeping at least one block from the original architecture, we are incorporating the signature of this architecture. It allows the application of this generic global feature layer to any architecture. Also, since a good start for any iterative solver implies smaller steps to reach the solution, this original block helps to initialize the PDE solution.

Estimated Savings. Suppose a Resnet architecture constructed with three resolutions has m residual blocks, and for the three resolutions, the compute cost is $= \{c_1, c_2, c_3\}$ respectively. The total compute cost for operating this network is $m \times (c_1 + c_2 + c_3)$. Global residual block replaces $m - 1$ residual blocks with

just one global block and assuming that the cost of this global block is similar, i.e. $= \{c_1, c_2, c_3\}$, then the cost to operate the modified network is $2 \times (c_1 + c_2 + c_3)$. Given that $m > 2$, the modified network can lead to computational savings over Resnet. Similar conclusions can be drawn for storage savings.

Motivational Example. To motivate our approach, we apply the Global feature layer to the Resnet32 (He et al., 2016) architecture, where at each feature map resolution, the same block repeats 5 times. With the experimental setup described in the section 7.3, we train three models on the CIFAR-10 dataset: (a) Resnet32 : same architecture as used in (He et al., 2016), (b) ODE based Resnet32, i.e., MDEQ (Bai et al., 2020) modified to match feature map configuration as in Resnet32, and (c) ResNet32-Global : replaced repeated blocks with global layers. Table 7.1 shows that both Resnet32 and MDEQ have similar performance. Note that MDEQ is significantly costly in terms of the floating-point multiply-add operations (MACs). In contrast, the proposed Resnet32-Global results in a much smaller model and a significantly lower computational footprint without any hit in the performance. This experiment clearly shows that the Global layer results in the following benefits:

1. **Shallow network.** Resnet32-Global has $\approx 3\times$ less depth.
2. **Less storage.** Resnet32-Global has $\approx 3\times$ less parameters.
3. **Less compute.** Resnet32-Global uses $\approx 5\times$ less MACs.
4. **Readily embedable in any network.**

Table 7.1: CIFAR-10 : Comparison between discrete Resnet32, ODE based Resnet32 (MDEQ(Bai et al., 2020)), and our PDE embedded Resnet32-Global. We compute the depth as the number of blocks in the network. Train and Inference time denote the cost of processing one pass of the train and test dataset on a V100 GPU. Supplementary Table 7.14 lists results for Resnet ($m = 2$) and CIFAR-100 dataset.

	Accuracy	**#Params**	**#MACs**	**Train Time(s)**	**Inference Time(s)**	**Depth**
Resnet32	92.49%	460K	70M	78	4.45	15
MDEQ	92.28%	1.1M	1.5B	409	23.32	-
Resnet32-Global	91.93%	162K	15M	24	1.91	6

Contributions.

- Proposed a Global feature layer that imposes PDE constraints on the input and output feature maps. Embedding this layer in deep networks results in their shallower variants with a smaller footprint with similar performance.
- Embedded the proposed global layer in many existing CNN architectures and conducted an extensive empirical study on benchmark image recognition datasets to show computational and storage savings.
- Proposed an efficient and approximate PDE solver to embed in the neural network wherein model accuracy can be traded-off for the computational budget.
- We provide pseudo-code for the Global layer that is readily deployable in any popular deep learning library. Our PyTorch implementation is available at https://github.com/anilkagak2/PDE_GlobalLayer.

7.2 Method

In this section, we will formalize the PDE constraint on the feature representation. We will describe the proposed Global layer, including our PDE choice, and embed an approximate numerical solver in the neural network. Finally, we will provide building blocks and pseudo-code to improve the understanding of our architecture.

Notation. For simplicity, we will assume that the output shape is the same as the input, and we are dealing with only a $2D$ feature map represented by the $X - Y$ plane. Let $I(x, y) \in \mathbb{R}^{h \times w}$ denote the input feature map with $h \times w$ entries. Let $H(x, y) \in \mathbb{R}^{h \times w}$ be the output feature map. We will denote Δ_{xy} as the differential operator (contains partial differential operators for various interactions between the two dimensions in the input).

7.2.1 PDE Constrained Features.

We enforce the following PDE constraint on the output feature map H

$$\Delta_{xy} H(x, y) = f(I(x, y)) \tag{7.1}$$

where f is a function applied on the input feature map. The above operator applies globally on the feature map and does not restrict itself to the local receptive field of operators such as one-layer convolutions.

Illustration. Before delving into further details about the global feature layer (namely the exact PDE and the numerical solver), we provide an intuitive example on the MNIST dataset to demonstrate the effectiveness of such a strategy. We construct a network with one feature layer followed by average pooling operation and classifier layer (see Figure 7·2). Note that the feature map layer constructs only one feature map. It gives rise to three networks by using different feature layers : (a) CNN-Net: convolutional layer as the feature map, (b) Residual-Net: a residual connection between the convolutional layers, and (c) PDE-Net: PDE constrained layer as the feature map. Note that all three networks have 524 parameters to ensure a fair comparison. Thus the only difference between these networks remains in the way features are processed. We train these networks on the MNIST dataset with the same settings (optimizer, learning rate, epochs, no data-augmentations, see Sec. 7.3) to provide a fair evaluation.

On the held-out test set, CNN-Net achieves 92.01% accuracy, Residual-Net achieves 92.53% accuracy, while PDE-Net achieves 95.03% accuracy. Since the network architecture apart from the feature layer is the same, we can analyze the feature map easily to see the contrast between the two feature representations. Figure 7·2 shows the intermediate representation from these neural networks. It shows that the feature maps generated by PDE-Net effectively highlight the input object by smoothing the noisy background and increasing brightness around the object edges.

7.2.2 Global Feature Layer

To embed a PDE in the neural network layer, we need to describe four components of the PDE: (a) its exact form, i.e. Δ_{xy}, (b) a numerical solver, (c) initial guess of the solution, and finally (d) choice of free parameters such as the function f. We refer to this new layer as the Global feature layer and describe these components below.

(a) PDE: At the heart of the Global feature layer is the following generic advection-diffusion PDE [1]

$$\frac{\partial}{\partial t}H = \nabla \cdot (D\nabla H) + \nabla \cdot (\mathbf{v}H) + f(I) \tag{7.2}$$

It lets us treat the input feature map pixels as particles in motion with velocity $\mathbf{v}$ that interact with their neighborhood through diffusion coefficient D. Starting at time $t = 0$ with initial guess of the concentration $H(t = 0)$, the solution of this advection-diffusion equation provides the final particle concentration $H(t = T)$ at time T. It is the output representation of the global feature layer. The motion of the particles affects the concentration and is modelled by the *advection* term $\nabla \cdot (\mathbf{v}H)$. Similarly, the term $\nabla \cdot (D\nabla H)$ describes the *diffusion* phenomenon, where particles shift between low and high concentrations to reach a steady state. Note that both D and $\mathbf{v}$ can be a function of the particle locations. Finally, the term $f(I)$ is the source of the particle concentration.

In our 2D world, the velocity and diffusion coefficients have two components, i.e. $\mathbf{v} = (u, v)$ and $D = (D_x, D_y)$, and the Eq. 7.2 boils down to the following form (Hutomo et al., 2019)

$$\frac{\partial}{\partial t}H(x,y,t) + \frac{\partial}{\partial x}\big(u(x,y,t)H(x,y,t)\big) + \frac{\partial}{\partial y}\big(v(x,y,t)H(x,y,t)\big)$$

[1] https://en.wikipedia.org/wiki/Convection-diffusion_equation

$$= \frac{\partial}{\partial x}\Big(D_x \frac{\partial}{\partial x} H(x,y,t)\Big) + \frac{\partial}{\partial y}\Big(D_y \frac{\partial}{\partial y} H(x,y,t)\Big) + f(I(x,y)) \tag{7.3}$$

(b) Iterative Solver: For an efficient implementation of the global layer, we need a simple and efficient PDE solver that can be embedded in the neural network and can achieve approximate solutions easily. To obtain a finite element scheme, it is standard in the literature to expand the partial differential operators with their finite-difference elements. Assume the discrete steps for x, y and t by δ_x, δ_y and δ_t respectively. Following (Hutomo et al., 2019), we discretize the Eq. 7.3 as (see Supplementary Sec. E.2 for detailed derivation),

$$\begin{aligned} LH^{k+1}_{x,y} &= MH^{k-1}_{x,y} - 2(u_x + v_y)\delta_t H^k_{x,y} + 2\delta_t f(I(x,y)) \\ &+ (-A_x + 2B_x)H^k_{x+1,y} + (A_x + 2B_x)H^k_{x-1,y} \\ &+ (-A_y + 2B_y)H^k_{x,y+1} + (A_y + 2B_y)H^k_{x,y-1} \end{aligned} \tag{7.4}$$

where $L = (1 + 2B_x + 2B_y)$, and $M = (1 - 2B_x - 2B_y)$

$$u_x = \frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x}; v_y = \frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y};$$

$$A_x = \frac{u\delta_t}{\delta_x}; A_y = \frac{v\delta_t}{\delta_y}; B_x = \frac{D_x\delta_t}{\delta_x^2}; B_y = \frac{D_y\delta_t}{\delta_y^2};$$

Given a suitable initialization of the output feature map H at $t = 0$, Eq. 7.4 provides an update rule to find the PDE solution at any time $t = T$. We need to initialize the algorithm and we can take K steps of this iteration on the entire 2D map to get the solution at time $T = K\delta_t$.

(c) Initialization: An initial guess of the solution is crucial for the convergence of the previous recursion. A better initial guess leads to faster convergence. Multiple strategies exist to initialize the output feature map, namely (a) input feature map I, (b) fixed function of the input, and (c) a learnable function of the input. We

follow the last option. Given an architecture, we use one of its building blocks as the initialization point and learn its parameters during the training stage with back-propagation. Thus, for architecture such as Resnet, at any resolution level, we use the first block as the output of the global feature layer at $t = 0$. We run the PDE for K steps to get the final feature map at time $\frac{K}{\delta_t}$.

(d) Choice of the free parameters: There are some free parameters in the Eq. 7.3. To complete the description of the Global feature layer, we list our parameterization for these free parameters, namely (a) function f (b) particle velocity (u, v), and (c) diffusion coefficient (D_x, D_y). For simplicity, we keep f as identity operator on the input and learn other parameters as the depth-wise convolution over the initialization. We point out that, for compute savings, one can further fix these parameters by keeping identity operations or treating these as hyper-parameters. We study the impact of different choices for free parameters in our ablations (see Sec. 7.3.5). We leave the design choice improvements (ex. employing architectural search for better combinations) for future work.

Finally, as a passing remark, we point out that we have not handled the boundary conditions explicitly in our formulation. Ideally, one should carefully design the behavior of the PDE at the boundary. Instead, we roll the image such that the first particle is a neighbor of the last. Since our goal is only to find an approximation, this modification suffices.

Note that our choices for the PDE and the numerical solver are motivated by the ease of implementation and simplicity in exposition. We leave the exploration of various other PDEs (Laplace equations, Heat equations, Navier-Stokes etc.) and better solvers to future work.

Implementation. Algorithm 4 shows the pseudo-code for the Global layer. This feature layer integrates easily in any architecture with appropriate initializations. By

default, we take the discrete step sizes to be $\delta_t = 0.2$, and run the recursions till $K = 5$ steps, resulting in the output state at $T = K\delta_t = 1$. We take $\delta_x = \delta_y = 1$ as the pixel values are not available at any finer details. For all our experiments, free parameters in Eq. 7.3 are depthwise convolutional operators with the same kernel size as the original block. For CIFAR-10 low-budget experiments, we use constant diffusion coefficients and set 1 as their default values.

Differences from existing PDE/ODE CNNs. Existing ODE-based CNNs (Chen et al., 2018) have focused on showing an equivalence between ResNet like architectures and the continuous-time ODEs. Although such connections provide new insights, few efforts utilize such ideas for compact CNNs due to expensive fixed point solvers and costly residual functions. Thus, these architectures are unable to scale-up to large datasets such as ImageNet (see Supplementary Sec. E.7 for further details).

Existing PDE-based CNNs (Ruthotto and Haber, 2020; Sun et al., 2020) apply generic stencil operators and do not solve any specific PDE. Furthermore, most of the works use such operators on the heavily downsampled initial feature map. In contrast, we apply the proposed Global layer at every resolution level and remove the dependence on the repetition of the architectural blocks.

We also point out that our update Eq. 7.4 is not a simple residual connection. It is a discretization corresponding to the PDE Eq. 7.3, wherein different elements interact in time and spatial dimensions. In contrast, a recursive update of any generic convolution would not necessarily correspond to an iterative scheme and will not converge. In addition, we do not have expensive non-linearities in the update equation that slow down the recursion. Typically in ODE/PDE CNNs, the function f is a residual block with multiple full convolutions and non-linearities like batch-norm and activation functions.

Algorithm 4 Pseudo Code for the Global Feature Block

Input : Input feature map $I \in \mathbb{R}^{h \times w}$
Input : Initial solution guess $F(I)$, Function f
Output : Output feature map O
Init : $H^{-1} = H^0 = F(I)$
Compute velocity (u, v), diffusion coefficient (D_x, D_y)
for $k = 1$ **to** K **do**
  Compute $f(H^{k-1}, I)$
  for $x = 1, y = 1$ **to** h, w **do**
    Set $H^{k+1}[x, y]$ as per Eq. 7.4
Set output feature map $O = H^K$

Architectures with Global layer. Fig. 7·3 shows an schematic of the proposed Global layer. As discussed earlier, the free parameters in this layer are constructed as depthwise convolutions. This layer can be embedded in any existing architectures as seen in Sec. 7.3.

7.3 Experiments

In this section, we will apply the proposed Global layer in popular architectures. We will show that the resulting architectures have a much smaller computational and storage footprint than the original models. We will evaluate these models on various benchmark image recognition datasets (see Appendix A.1).

7.3.1 Experimental Setup

We implement the Global feature layer in PyTorch using the Algorithm 4. Our experiments include strong baselines such as Resnet (He et al., 2016), Densenet (Huang et al., 2017), Wide Resnet (Zagoruyko and Komodakis, 2016), DARTS (Liu et al., 2019a). We embed global feature layers in these architectures and remove feature block repetitions resulting in Resnet-Global, Densenet-Global, Wide Resnet-Global, and DARTS-Global. For the ImageNet experiments, we perform similar adjustments

to the state-of-the-art architectures such as MobileNetV3 (Howard et al., 2019) and EfficientNet (Tan and Le, 2019). We follow guidance from these works for a fair comparison. Unless the original papers recommend extra augmentation or training techniques, we minimize cross-entropy loss with the stochastic gradient descent with momentum optimizer in all our experiments. In addition, we cite known results from the literature for baseline reference.

We primarily report accuracy, the number of parameters, and the number of floating-point multiply-add operations[2]. These metrics measure the performance, model size, and computational footprint of a model. In addition, we tabulate depth, inference, and training times for few experiments. We do not pursue compression-related ideas (quantization, deep-compression, distillation, etc.) or hardware optimizations in this work to simplify the crux of our exposition. These can be further incorporated in our scheme to provide similar gains as reported in earlier works.

7.3.2 Results on MNIST-10

Since the ODE baselines do not scale to large-scale datasets, we compare our architectures with these baselines on MNIST-10 dataset. We use the Resnet architecture in this experiment. Similar to (Chen et al., 2018; Sun et al., 2020), we use one Resnet architecture where we apply one Global or one ODE layer or 6 residual layers after downsampling the input twice. In addition, we use a budget ($<$ 5M MACs) Resnet architecture where we apply one Global layer or residual layers without downsampling (see Supplementary Sec. E.3 for details). We minimize the cross-entropy loss using the SGD optimizer with momentum. We follow a similar experimental setup (epochs, learning rate, scheduler, etc.) as the baselines.

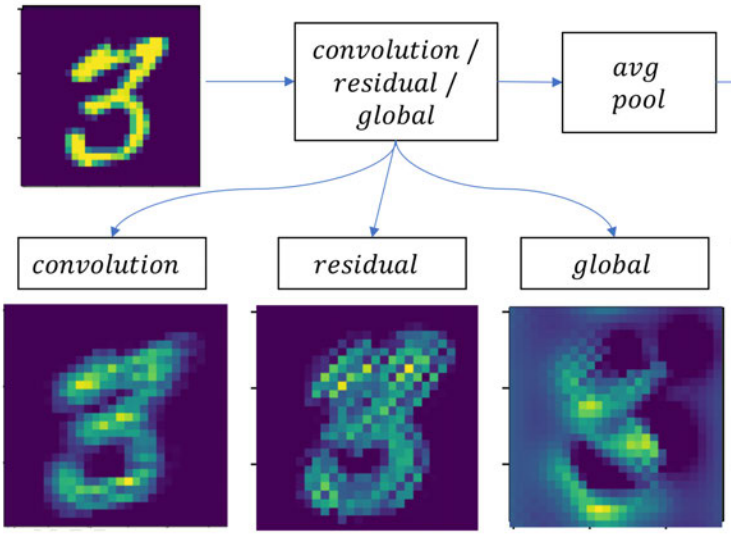
Results. *Resnet-Global at same accuracy has* $3-5\times$ *storage gains (number of pa-*

[2]Following convention (Howard et al., 2019; Tan and Le, 2019) we leverage the benchmarked PyTorch utility https://github.com/Lyken17/pytorch-OpCounter for MACs.

rameters) and $2.5 - 3\times$ *compute gains.* Table 7.2 compares the Resnet-Global model with Neural ODEs(Chen et al., 2018), NeuPDE (Sun et al., 2020), Resnet (He et al., 2016). It shows that Resnet-Global achieves similar performance as the baselines while reducing the number of parameters and the compute requirements. In particular, compared to Resnet, our architectures reduce the storage by $3 - 4\times$ and reduce the compute by $2 - 3\times$. On the other hand, Neural-ODEs have $3\times$ higher compute footprint when compared to Resnet.

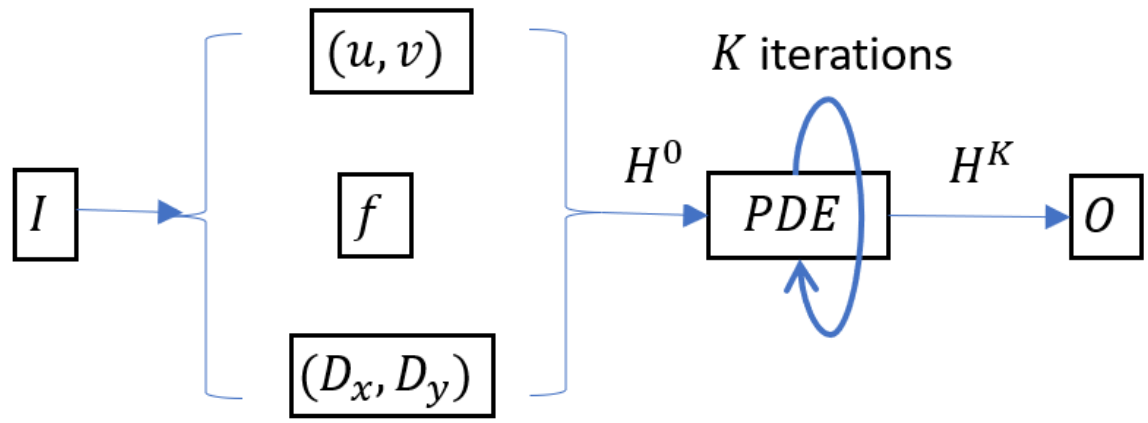
Table 7.2: Results on MNIST-10. Networks with a Global layer have significantly less storage and compute requirements than ODE, PDE, and discrete CNNs.

Architecture	**Accuracy**	**#Params**	**#MACs**
Neural ODEs (Chen et al., 2018)	99.49	220K	100M
NeuPDE (Sun et al., 2020)	99.49	180K	
Resnet (He et al., 2016; Chen et al., 2018)	99.59	600K	30M
Resnet-Global (ours)	99.51	136K	14M
Resnet	99.61	33.3K	5.7M
Resnet-Global (ours)	99.43	9.94K	1.7M

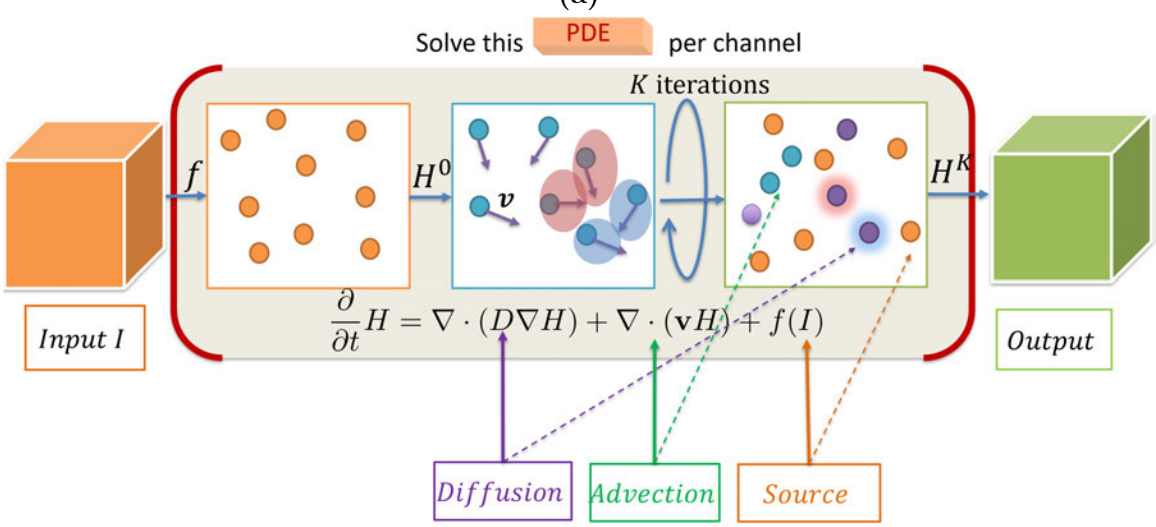


Feature Backbone	Acc. (%)	Confusion b/w 3 & 5
Convolution	92.01	27
Residual	92.53	30
Global	**95.03**	**9**

Figure 7·2: Toy Example comparing different backbones: Convolutional, Residual, and Global. We show network representation for the input image for the letter three. Intermediate features from Convolutional and Residual backbones do not show bright intensity around the edges and have an uneven background. In contrast, the Global layer smoothens it out and shows bright spots around the digit. Thus, the Global layer provides a better and markedly different representation than the other two backbones. All three networks have 524 parameters. Network with Global layer achieves 95% accuracy while the other two achieves $\approx 92.5\%$ accuracy. It also has a significantly lesser confusion between the letters 3 and 5. See other visualizations in supplementary Sec. E.8.



(a)



(b)

Figure 7·3: Schematic for the Global layer using the diffusion PDE.

Table 7.3: Results on CIFAR-10 and CIFAR-100. Architectures with Global layer require $2-5\times$ less computational and storage budget. For reference, we borrow results from existing literature: ANODE (Gholami et al., 2019; Sun et al., 2020), Hamiltonian PDE (Ruthotto and Haber, 2020), and DenseNet-BC (Huang et al., 2017).

Architecture	**CIFAR-10**			**CIFAR-100**		
	Accuracy	**Params (Savings)**	**MACs (Savings)**	**Accuracy**	**Params (Savings)**	**MACs (Savings)**
DenseNet-BC	95.49	800K	300M	77.73	800K	300M
Resnet56 (He et al., 2016)	93.03	850K	127M	-	-	
NeuPDE (Sun et al., 2020)	95.39	9M		76.39	9M	
ANODE	94.96	11M		71.28	11M	
Hamiltonian PDE	89.3	262K		64.9	362K	
MDEQ (Bai et al., 2020)	93.8	10M	8.3B	-	-	
Resnet32 (m=5)	92.49	460K (1.0×)	70M (1.0×)	68.57	473K (1.0×)	70M (1.0×)
Resnet-Global (m=1)	91.93	162K (2.8×)	15M (4.7×)	68.01	168K (2.8×)	15M (4.7×)
Resnet56 (m=9)	93.03	850K (1.0×)	127M (1.0×)	70.48	861K (1.0×)	127M (1.0×)
Resnet-Global (m=2)	93.01	330K (2.6×)	30M (4.2×)	70.06	336K (2.6×)	30M (4.2×)
Densenet	95.32	769K (1.0×)	297M (1.0×)	77.21	800K (1.0×)	297M (1.0×)
Densenet-Global	95.01	465K (1.7×)	136M (2.2×)	75.69	481K (1.7×)	136M (2.2×)
Wide-Resnet	95.91	9.0M (1.0×)	1.30B (1.0×)	79.11	9.0M (1.0×)	1.30B (1.0×)
Wide-Resnet-Global	95.54	2.8M (3.2×)	425M (3.1×)	78.13	2.8M (3.2×)	427M (3.1×)
DARTS	97.11	3.3M (1.0×)	539M (1.0×)	82.51	3.4M (1.0×)	539M (1.0×)
DARTS-Global	96.83	783K (4.2×)	213M (2.5×)	81.89	835K (4.1×)	213M (2.5×)
Resnet	80.76	13K	3.42M	35.21	14K	3.42M
Resnet-Global	82.55	14K	3.6M	43.62	16K	3.6M
Wide-Resnet	83.83	22K	9.8M	39.01	23K	9.8M
Wide-Resnet-Global	85.51	23K	8.7M	50.23	24K	8.7M
DARTS	86.05	39K	7.7M	54.57	43K	7.7M
DARTS-Global	88.44	34K	8.2M	60.68	41K	8.2M

7.3.3 Results on CIFAR-10 & CIFAR-100

Architectures. We evaluated popular residual architectures in this task, namely Resnet, Wide-Resnets, and DenseNets. We ran two variants of Resnet (He et al., 2016): Resnet32 repeats same residual block $m = 5$ times at every resolution while Resnet56 increases the repetitions to $m = 9$. We strictly adhere to the configuration described in the original paper (He et al., 2016). We replace the repeated blocks in these architectures with the Global layer, resulting in two Resnet-Global architectures where we keep $m = 1$ and $m = 2$ repetitions.

We used Wide-Resnet (Zagoruyko and Komodakis, 2016) with 40 layers and 4× the width of the residual architecture, commonly referred to as WRN-40-4. It is constructed similar to the above-described Resnet with $m = 6$ repetitions except with 4× the width. We replace these repetitions with a Global layer, resulting in Wide-Resnet-Global with $m = 1$. For DenseNet, we borrowed the cost-efficient variant DenseNet-BC (Huang et al., 2017) which has a growth rate of 12 and three dense blocks each with 16 dense layers, also known as Densenet-BC (k=12, L=100).

In addition, we apply the Global layer to an efficient architecture found by neural architecture search DARTS (Liu et al., 2019a) which uses a cell found by the search. We modify this network by replacing the cell repetitions with a global layer.

Training Details. All the global architectures and their respective original baselines are trained with SGD+momentum optimizer for 300 epochs. We set the initial learning rate to 0.1 and decay by 10 at epoch 150 and 225. Note that for each architecture we use the recommended hyper-parameter settings (batch size, weight decay, data-augmentation etc.). When hyper-parameter recommendations are missing, we use grid search over weight decay $\{3e-4, 2e-4, 1e-4, 5e-5, 1e-5\}$ and batch size $\{32, 64, 128, 256\}$ on a validation set (see Supplementary Sec. E.4 for final hyper-parameters).

Results. We tabulate primary evaluation metrics in the the Table 7.3. Additional evaluation metrics such as train/inference times and architecture configurations are tabulated in Table 7.4. Below we summarize the main findings.

(a) Computational and storage savings. Table 7.3 shows that Global layer enabled architectures are compact and computationally efficient. In comparison to the baseline architectures, models with Global layer achieve $2.5-4.5\times$ flops reduction and $2.5-4.2\times$ parameters reduction.

(b) Lower training and inference times. It is evident from Table 7.4 that Global architectures have better training and inference times, with at least $2\times$ reductions. Note that we can discard the training time in many IoT applications as a one-time cost. In contrast, the inference time directly affects the battery drain and the responsiveness of the device.

(c) Shallower neural networks. From Table 7.4, one can deduce that Global architectures are shallower as they reduce the number of cells in the architecture by nearly $3\times$ in many instances. For example, the popular Resnet56 model has 27 cells while Resnet-Global only has 9 cells.

(d) Integrable in many popular architectures. Table 7.3 and 7.4 show that Global layer can be successfully applied across a range of architectures with the aforementioned benefits.

(e) Comparison at a fixed computational budget. It is worth noting that for an IoT device, although storage space could be a prohibitive factor for large models, the main issue with such small devices is computational in nature. The number of floating-point operations used for inference directly impacts the battery drain as well as the real-time latency required for any successful ML application. Keeping this in mind, we compare Global architectures with the baselines under a low computational budget, i.e., $<$ 10M MACs. At this regime, Global models achieve much higher

accuracy than the baseline architectures. Thus, demonstrating that our models are better suited for IoT applications.

7.3.4 Results on ImageNet-1K

Experiments on MNIST and CIFAR datasets demonstrate that architectures with a Global layer are compact, shallower, and computationally efficient. In this section, we show that the Global layer improves the state-of-the-art models for the ImageNet dataset. It has been already shown in the literature that MobileNet (Sandler et al., 2018; Howard et al., 2019) and EfficientNet (Tan and Le, 2019) models are more cost-efficient than the Resnet/Densenet models.

Architectures. We apply the global layer to MobileNetV2, MobileNetV3, and EfficientNet and replace the repetitions with the Global layer. For MobileNetV2, we use the baseline with a width multiplier of 1. We obtain MobileNetV2-Global by replacing all the invertible residual block repetitions with one Global layer at each feature resolution. We use the large variant of MobileNetV3 with a width multiplier of 1. We create MobileNetV3-Global by replacing the building block (invertible residual + squeeze-and-excite) with a Global layer. Finally, for the EfficientNet family, we only pick the B0 variant due to its low compute requirements. We provide the architecture details along with building blocks in the Supplementary Sec. E.5.

Training Details. Due to computing limitations, we do not re-train the baselines and only report their publicly known performance metrics in Table 7.5. We train the architectures with the Global layer from scratch using the hyper-parameter recommendations from the baselines. We use the RMSProp optimizer with 0.9 momentum to minimize the cross-entropy loss along with a weight decay term with a value of $1e-5$. The remaining hyper-parameters are available in the Supplementary Sec. E.5.

Results. Table 7.5 reports the top1 accuracy along with the number of parameters

Table 7.4: CIFAR-10: Train & Inference times (cost of one pass through train and test dataset on a V100 GPU) along with the number of cells. Total cells are a proxy for depth of the network.

Architecture	Accuracy	Train Time(s)	Inference Time(s)	original cells	global cells	total cells
Resnet56	93.03	119	6.71	27	0	27
Resnet-Global	93.01	56	2.33	6	3	9
Densenet	95.32	138	10.22	48	0	48
Densenet-Global	95.01	24	3.4	24	2	26
Wide-Resnet	95.91	34	4.88	18	0	18
Wide-Resnet-Global	95.54	20	2.71	3	3	6
DARTS	97.11	126	6.86	20	0	20
DARTS-Global	96.83	61	3.43	6	3	9

and number of multiply-add operations. Below we summarize the main findings.

(a) Computational and storage savings. Table 7.5 shows that Global layer enabled architectures are compact and computationally efficient. Our architectures achieve similar performance as the baselines with 1.5× fewer MACs and 2.5× parameters savings. In addition, with a slight reduction in accuracy, we save 2× MACs and 3× parameters.

(b) Lower training and inference times. Global architectures have better training and inference times, with nearly 2× reductions (see Supplementary Sec. E.5).

(c) Integrable in many popular architectures. Table 7.5 show that the Global feature layer can be successfully applied across a range of architectures.

Table 7.5: Results for ImageNet dataset.

Architecture	ImageNet				
	Top-1	#Params		#MACs	
MobileNet (Howard et al., 2017)	70.6	4.2M		575M	
SqueezeNext (Gholami et al., 2018)	67.44	3.23M		708M	
DenseNet-169 (Huang et al., 2017)	76.2	14M		3.5B	
Resnet-152 (He et al., 2016)	77.8	60M		11B	
MDEQ-Small (Bai et al., 2020)	75.5	18M			
MobileNetV2(Sandler et al., 2018)	72.0	3.4M	(1.0×)	300M	(1.0×)
MobileNetV2-Global	71.63	1.6M	(2.1×)	193M	(1.6×)
MobileNetV2-Global-s	69.03	1.2M	(2.8×)	150M	(2.0×)
MobileNetV3(Howard et al., 2019)	75.2	5.4M	(1.0×)	219M	(1.0×)
MobileNetV3-Global	74.11	3.0M	(1.8×)	156M	(1.4×)
MobileNetV3-Global-s	71.89	1.8M	(3.0×)	110M	(2.0×)
EfficientNet-B0 (Tan and Le, 2019)	77.1	5.3M	(1.0×)	390M	(1.0×)
EfficientNet-B0-Global	76.12	2.4M	(2.2×)	244M	(1.6×)
EfficientNet-B0-Global-s	74.53	1.8M	(2.9×)	201M	(1.9×)

7.3.5 Ablative Experiments

In this section, we probe various aspects of the proposed method.

(A) Impact of K in iterative solver. We study the effect of the K hyper-parameter in the iterative solver. On CIFAR-10 and CIFAR-100 datasets, Table 7.6 shows the performance of the Resnet-Global model (see Sec. 7.3.3) as we vary K. Note that the increasing K does not increase the number of parameters in our formulation. In this case, for CIFAR-10, Resnet-Global has 162K parameters, while for CIFAR-100, Resnet-Global has 168K parameters. Our default choice of $K = 5$ is justifiable from the marginal improvement in accuracy with increased computational cost.

Table 7.6: Effect of the hyper-parameter K in update Eq. 7.4.

Architecture	K	CIFAR-10		CIFAR-100	
		Accuracy	#MACs	Accuracy	#MACs
Resnet-Global	1	90.69	14.3M	66.89	14.3M
Resnet-Global	3	91.34	14.7M	67.37	14.7M
Resnet-Global	5	91.93	15M	68.01	15M
Resnet-Global	10	92.01	15.5M	68.23	15.5M
Resnet-Global	20	92.21	17M	68.24	17M

(B) Baseline and Global model with same MACs. Our earlier experiments showed that the Global layer improves the computational footprint of any architecture. In this ablation, we compare the Global architectures with the baselines by keeping the same computational budget. Table 7.7 shows the performance of these models on the CIFAR-100 dataset. Global models improve baselines by up to 4%.

Table 7.7: Global models with similar budget as original models.

Architecture	CIFAR-100		
	Accuracy	#Params	#MACs
Resnet56	70.48	861K	127M
Resnet-Global	74.33	1.32M	119M
Densenet	77.21	800K	297M
Densenet-Global	78.91	922K	247M
Wide-Resnet	79.11	9M	1.3B
Wide-Resnet-Global	80.53	9M	1.3B
DARTS	82.51	3.4M	539M
DARTS-Global	84.19	2.4M	519M

(C) Non-linear Residual block as f. We compare the cost of using a standard

Residual block as the function f in the update Eq. 7.4 instead of the identity function. Table 7.8 shows the performance characteristics of Resnet and WideResnet on CIFAR-10 and CIFAR-100. It shows that using Residual block only brings marginal improvements in accuracy ($< 0.5\%$) with a significant increase in compute cost.

Table 7.8: Ablative experiments to study the effect of the using a Residual block instead of our current choice in update Eq. 7.4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	f	**Acc.**	**#Params**	**#MACs**	**Acc.**	**#Params**	**#MACs**
Resnet	Identity	91.93	162K	15M	68.01	168K	15M
Resnet	Residual	92.28	175K	27M	68.37	181K	27M
WideResnet	Identity	95.54	2.8M	425M	78.13	2.8M	427M
WideResnet	Residual	95.67	3M	566M	78.34	3.1M	567M

(D) Neural ODEs without equilibrium. We compared the proposed layer vs Neural ODE layer with smaller number of iterations. Instead of running the Neural ODEs up to equilibrium, we unrolled the update equation to $K = 5$ steps (similar unrolling as our Global layer). Table 7.9 shows that even such an optimization does not improve its compute footprint when compared to a standard Resnet architecture. In contrast, Global layer uses at least $2\times$ less MACs than Resnet.

Table 7.9: Neural ODEs without equilibrium.

Architecture	Accuracy	#Params	#MACs
Neural ODEs (Chen et al., 2018)	99.49	220K	100M
Neural ODEs (no equilibrium)	99.46	220K	27M
Resnet (Chen et al., 2018)	99.59	600K	30M
Resnet-Global (ours)	99.51	136K	14M
Resnet	99.61	33.3K	5.7M
Resnet-Global (ours)	99.43	9.94K	1.7M

(E) Impact of increasing K on compute time. Table 7.10 provides inference time for the ablative experiment discussed in the main text (see Sec. 7.3.5(A) and the Table 7.6). This shows that the inference time increases sub-linearly with K.

(F) Choice of free parameters. In the main text, we studied some ablations on the choice of one free parameter, i.e., f. Here, we will study the impact of (u, v) as well as (D_x, D_y) on the performance characteristics of the global architectures (see

Table 7.10: Adding inference numbers for Resnet-Global architecture in Table 6 (Effect of the hyper-parameter K in update Eq. 4). Inference time is measured as the amount of time taken to pass through the test set on a V100 GPU.

K	CIFAR-10			CIFAR-100		
	Accuracy (%)	**#MACs**	**Inference Time (s)**	**Accuracy (%)**	**#MACs**	**Inference Time (s)**
1	90.69	14.3M	1.43	66.89	14.3M	1.57
3	91.34	14.7M	1.74	67.37	14.7M	1.83
5	91.93	15M	1.91	68.01	15M	1.98
10	92.01	15.5M	2.18	68.23	15.5M	2.22
20	92.21	17M	2.48	68.24	17M	2.59

Table 7.11, 7.12, & 7.13). We will use the following options for the free parameter initialization :

- 'Residual(Basic)' : Used as the building block in many residual networks (He et al., 2016)
- 'Residual(Bottleneck)' : Used as the building block in residual networks with larger depth (He et al., 2016)
- 'FullConv': One layer full 3×3 convolution
- 'PwConv': One layer 1×1 convolution, also referred as the pointwise-convolution
- 'DwConv': One layer 3×3 depthwise convolution (does not involve any channel mixing)
- 'Identity' : No operation (simply pass through the features)

(G) Training & Inference Time Comparison. We further extend the motivational example to CIFAR-100 dataset as well as another Resnet32 model with $m = 2$ repetitions. We show the results in Table 7.14. This shows that even when Resnet is equipped with same depth as the Resnet-Global model, it still has much worse compute and storage footprint. In addition, by reducing the number of repetitions, Resnet suffers a significant degradation in performance.

Table 7.11: (CIFAR-10/100) Ablative experiments to study the effect of the using different free parameter choice (D_x, D_y) than our current choice in update Eq. 7.4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	(D_x, D_y)	**Acc.**	**#Params**	**#MACs**	**Acc.**	**#Params**	**#MACs**
Resnet	Identity	91.26	159K	14.7M	67.15	165K	14.7M
Resnet	DwConv	91.93	162K	15M	68.01	167K	15.3M
Resnet	PwConv	92.17	170K	16.4M	68.34	176K	16.4M
Resnet	FullConv	92.12	256K	28.9M	68.39	262K	28.9M
Resnet	Residual(Basic)	92.68	353K	43M	68.53	359K	43.3M
Resnet	Residual(Bottleneck)	92.57	279K	32M	68.48	284K	32.4M

Table 7.12: (CIFAR-10/100) Ablative experiments to study the effect of the using different free parameter choice (u, v) than our current choice in update Eq. 7.4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	(u, v)	**Acc.**	**#Params**	**#MACs**	**Acc.**	**#Params**	**#MACs**
Resnet	Identity	91.36	159K	14.7M	67.07	165K	14.7M
Resnet	DwConv	91.93	162K	15M	68.01	167K	15.3M
Resnet	PwConv	92.11	170K	16.4M	68.23	176K	16.4M
Resnet	FullConv	91.98	256K	28.9M	68.28	262K	28.9M
Resnet	Residual(Basic)	92.47	353K	43M	68.45	359K	43.3M
Resnet	Residual(Bottleneck)	92.43	279K	32M	68.41	284K	32.4M

Table 7.13: (MNIST-10) Ablative experiments to study the effect of the using different free parameter choice (u, v, D_x, D_y) than our current choice in update Eq. 7.4. Here, all architectures use the Global layer.

Global	Function	(D_x, D_y)			(u, v)		
Architecture		**Acc.**	**#Params**	**#MACs**	**Acc.**	**#Params**	**#MACs**
Resnet	Identity	99.36	134K	13.9M	99.38	134K	13.9M
Resnet	DwConv	99.49	136K	14M	99.46	136K	14M
Resnet	PwConv	99.56	142K	14.3M	99.52	142K	14.3M
Resnet	FullConv	99.43	208K	16.6M	99.47	208K	16.6M
Resnet	Residual(Basic)	99.62	282K	19.3M	99.53	282K	19.3M
Resnet	Residual(Bottleneck)	99.59	225K	17.2M	99.54	225K	17.2M

7.4 Discussion

Global layers demonstrate that carefully designed differential equations reduce storage and computational footprints of convolutional architectures. While we explored advection-diffusion PDE in this chapter, other PDEs may yield different desirable properties. We leave the extension to PDEs such as Heat-Dissipation, Navier-Stokes equations, etc., to future work.

Our design choice of keeping one residual block followed by one Global layer was

Table 7.14: CIFAR-10 & CIFAR-100 : Comparing discrete Resnet32 (m=5), ODE based Resnet32 (MDEQ[10]), our PDE embedded Resnet32-Global and Resnet32 (m=2) replacing Global layer with a Residual block in Resnet32-Global. We compute the depth as the number of blocks in the network. Inference time denote the cost of processing one pass of the test dataset on a V100 GPU.

CIFAR	Method	Acc.	#Params	#MACs	Train Time(s)	Inf. Time(s)
10	Resnet32 (m=5)	92.49%	460K	70M	78	4.45
	MDEQ	92.28%	1.1M	1.5B	409	23.32
	Resnet32-Global	91.93%	162K	15M	24	1.91
	Resnet32 (m=2)	89.42%	175K	27M	33	2.32
100	Resnet32 (m=5)	68.57%	473K	70M	82	4.57
	MDEQ	67.89%	1.1M	1.5B	413	25.66
	Resnet32-Global	68.01%	168K	15M	29	2.03
	Resnet32 (m=2)	63.35%	182K	27M	41	2.65

motivated by the goal of reducing the model footprint. This design restriction should be lifted by using multiple repetitions of the Global layer to increase model capacity.

One issue with the differential equations is that their iterative solver implementation requires full numerical precision during training. While it is possible to carefully design half-precision schemes that utilize modules like AMP in PyTorch, the current implementation suffers from numerical instabilities when training in half-precision mode.

Chapter 8

Spatially Interpolated Convolutional Neural Networks (SI-CNNs)

8.1 Introduction

Convolutional Neural Networks (CNNs) are de-facto architectures for vision applications such as image classification(Howard et al., 2019; Tan and Le, 2019), object detection(Jiao et al., 2019), and semantic segmentation (Minaee et al., 2022). At the heart of these architectures lie basic building blocks such as residual blocks and other variants. Researchers have designed resource-efficient convolutional blocks that yield a better trade-off between performance and resource usage. For instance, Inverted Residual Block (Sandler et al., 2018) and its derivatives have been used in many efficient CNN architectures such as MobileNetV3(Howard et al., 2019), EfficientNet(Tan and Le, 2019), MNASNet(Tan et al., 2019), etc.

Typically, such blocks compute high-dimensional activation maps to capture key features while being computationally inexpensive. For example, the *Inverted Residual Block* projects the input onto a high dimensional space, and performs a depth-wise convolution, followed by a projection onto a lower dimension space. Thus they compute high-dimensional features without paying for full 3-d convolutions, with the intuition that such a “low-rank" structure is enough to capture the key features.

However, such methods typically do not exploit any redundancies in the spatial

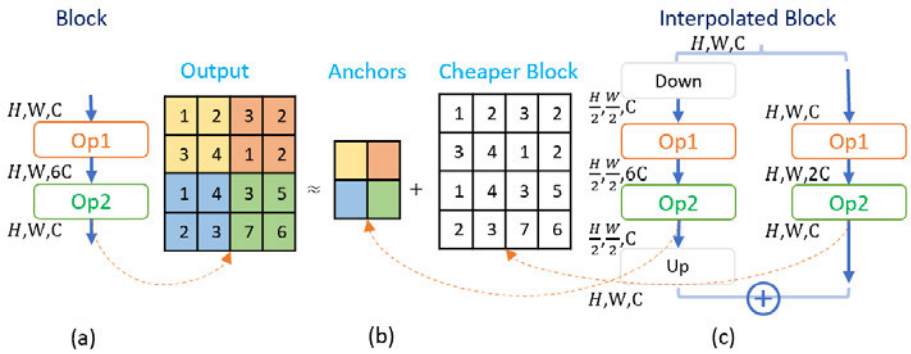


Figure 8·1: Illustration. On the left is a generic convolutional block that projects the input features to a high dimension (Op1) and then projects it down to the low dimensional output space (Op2). On the right is the interpolated variant of this block. It creates two parallel branches, one where it computes anchor features from downsampled input and the other where it computes cheaper features from the full input image. Our inductive bias is that the output of the original block can be written as an interpolation between these two different features.

features. The key observation of our work is that spatially features are quite "smooth". That is, while the features of each pixel might be different, they tend to be "close" to each other. It enables accurate predictions without paying for expensive computations for each pixel.

Our Idea. In this work, we exploit the above mentioned observation by using a simple and novel *Spatial Interpolation (SI)* scheme widely applicable to most convolutional models. At a high level, our approach is to compute spatial "anchors" for a given activation map and only perform expensive computation on the anchors, while we use significantly cheaper computation for the rest of the image (say, decreasing the number of features/channels). That is, while the method performs computationally intensive feature computation, it does so only for a fraction of pixels in the activation map and perform relatively cheap operations on the full activation map.

We illustrate our scheme by applying it to the popular *inverted residual block* (Sandler et al., 2018) in Figure 8·1(a). Note that the computational cost of the inverted

residual block originates from the presence of the high dimensional space. For example, projection from $C \rightarrow 6 \cdot C$ channels and projection back from $6 \cdot C \rightarrow C$ channels. As shown in Figure 8·1(b), we reduce this cost by expressing the output of this block as an interpolation between two computationally cheaper processing branches. First, *anchor features* generated from a down-sampled $(\frac{H}{2}, \frac{W}{2})$ input features, and second, *cheaper features* generated from the full-size (H, W) input map but with significantly smaller channel dimensions. Figure 8·1(c) represents our spatially interpolated variant of this block. Anchor features are common for a neighborhood (in Figure 8·1(b), for a 2×2 area), and the cheaper computation over the remaining image denotes the perturbation from the anchors unique to the individual pixels.

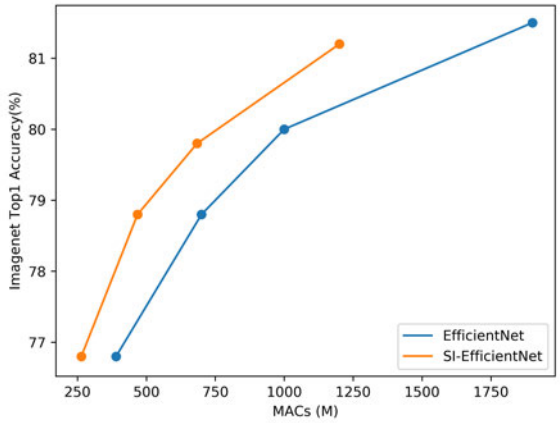


Figure 8·2: ImageNet Classification. Top-1 accuracy vs computation (MACs) on ImageNet dataset. Plot compares EfficientNet models against the proposed spatially interpolated (SI) blocks based models (SI-EfficientNet).

That is, the anchor features capture the high-dimensional features, whereas the low-cost perturbations in the remaining image capture subtle spatial variations. Naturally, for higher-resolution images, we can allow a smaller fraction of anchors compared to the low-resolution images, thus helping CNN models scale. Also, parallel computation of anchor features and cheap perturbations implies that the method should help neural accelerators as they can schedule both these computations simultaneously.

Toy Example. To illustrate, let us dive into an image classification task on the CIFAR-10 (Krizhevsky and Hinton, 2009) dataset. We use a simple architecture with a convolutional stem followed by Spatially Interpolated or Inverted Residual block and a classifier layer. We describe the experimental setup and training details in Appendix F.1. Table 8.1 shows the performance and resource usage of these two architectures. For a fair comparison, both architectures have the same number of parameters. It is clear that the Interpolated block reduces the multiply-addition operations (MACs) by nearly half and results in nearly 33% savings in the inference time. In addition, it has higher accuracy than the Inverted Residual block (see Table 8.1). This experiment shows that the interpolation scheme results in the following benefits:

- **Less Compute.** It has low resource footprint that translates into low inference and training time.
- **Little Loss in Performance.** By exploiting the spatial smoothness of activation maps, the method ensures that the loss in accuracy compared to the full model is relatively small.
- **Similar Storage Footprint** as the original residual block.
- **Easily Integrable.** It is simple to implement. Sec. 8.2 gives architecture updates for various blocks.

While our method is general and applicable to most standard convolutional blocks, in this work, we focus mainly on the Inverted Residual Block due to its widespread usage in low-resource deployments. Figure 8·2 compares MACs vs ImageNet accuracy trade-off of our spatially interpolated inverted residual block (SI-IR) against the standard inverted residual block deployed in EfficientNet (Tan and Le, 2019) architectures. It clearly demonstrates that the proposed spatial interpolation scheme is able to improve accuracy for the same number of computational steps as measured by

MACs. In Sec. 8.2, we will describe the Interpolated Inverted Residual with various design choices. In Sec. 8.2.4 and Sec. 8.3.4, we show how to adapt this interpolation scheme to other building blocks such as Fused-MBConv(Tan and Le, 2021).

Table 8.1: Illustration on CIFAR-10 : Comparison between a convolutional network with Inverted Residual and Spatially Interpolated Residual Block. Train and Inference time denote the cost of processing one pass of the train and test dataset on a V100 GPU.

	Accuracy	**#Params**	**#MACs**	**Train Time(s)**	**Inference Time(s)**
ConvNet-IR	75.88%	14K	11M	1.5	0.3
SI-ConvNet-IR	80.24%	14K	6M	1.1	0.2

Contributions. We summarize our contributions below.

- We propose a novel interpolation scheme that exploits spatial smoothness in convolutional residual blocks to improve the accuracy-vs-computation tradeoff and show that the proposed interpolation scheme can be applied to most standard residual blocks.
- Our proposal reduces the computational cost by up to $\approx 40\%$ for SOTA architectures such as MobileNetV3 and EfficientNet on the ImageNet classification task.
- We demonstrate applicability of our method by designing spatially interpolated backbones in semantic segmentation and show similar accuracy-vs-computational benefits as the ImageNet task.
- Our proposed scheme is simple to implement in any major deep-learning library. Our PyTorch implementation is available at `https://github.com/anilkagak2/Spatial_Interpolation`.

8.2 Method

In this section, we will discuss our spatial interpolation scheme (SI-f) for a generic residual block f. Then, we will instantiate it for inverted residual blocks, resulting

in spatially interpolated inverted residual blocks. We will analyze the computational footprint of the proposed block. Finally, we will conclude this section with applications of the spatial interpolation to other residual blocks such as fused MBConv(Tan and Le, 2021) and Bottleneck(He et al., 2016).

8.2.1 Spatially Interpolated Convolutional Blocks

In this section, we describe our spatially interpolated (SI) convolutional blocks. Let $X \in \mathbb{R}^{H \times W \times C}$ be an intermediate activation map generated after say i-th convolutional block in a DNN; e.g., say after 4-th residual block in ResNet-18. The activation map has H rows, W columns, and C channels. Also, let $f(\cdot)$ be a convolutional block applied by the network on X, i.e., $Y = f(X)$ where $Y \in \mathbb{R}^{H' \times W' \times C'}$ is the $(i+1)$-th convolutional block.

With this notation, we now describe the SI block as a combination of the following three steps:

1. **Down-sampling.** The first step of SI-block is to downsample the activation map onto a small set of *anchors*, i.e., $A = \mathcal{D}(X)$ where $\mathcal{D} : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{(s \cdot H) \times (s \cdot W) \times C}$ is a downsampling operator and $s < 1$ is a scaling factor. There are multiple options for downsampling, including average-pooling or max-pooling. In this work, we use strided depth-wise convolution for downsampling.
2. **Parallel and Cheaper Convolutional Block.** Next, we apply two different convolutional blocks to X and A, where the block applied to X can be of the same type as f but should be significantly cheaper; for example, it might have a lower kernel size. On the other hand, the block applied to A can be as expensive as f or can be even more expensive. That is, $\widetilde{A} = f_A(A) \in \mathbb{R}^{(s \cdot H') \cdot (s \cdot W') \cdot C'}$, $\widetilde{X} = f_X(X) \in \mathbb{R}^{H' \times W' \times C'}$, and both have C' channels. The computational cost of f_X is significantly lesser than that of f, while f_A can be the same as f.

3. **Up-sampling and addition.** Finally, we up-sample $\widetilde{A}$ and add it to $\widetilde{X}$, i.e., $Y = \alpha \cdot \mathcal{U}(\widetilde{A}) + \beta \cdot \widetilde{X}$. For up-sampling, we use a simple scheme where $\mathcal{U}(\widetilde{A})_{ij} = \text{mean}(\widetilde{A}_{i'j'}; |i - i'| + |j - j'| = \min_{a,b} |i - a| + |j - b|)$, i.e., upsampled value is an average of the spatially neighboring pixels in the feature map $f_A(\mathcal{D}(X))$. Thus, the spatially interpolate block (SI-f) is given by:

$$\text{SI-}f(X) = \alpha \cdot \mathcal{U}(f_A(\mathcal{D}(X))) + \beta \cdot f_X(X).$$

The above formulation allows α and β to be non-linear functions for increased representation capacity. In our setup, for simplicity, $\alpha, \beta \in \mathbb{R}^{1\times1\times C}$ are learnable coefficients, two scalars per channel.

8.2.2 SI-Inverted Residual Blocks

We now instantiate the SI block for the widely used inverted residual (InvRes) blocks (Howard et al., 2019; Tan and Le, 2019); see Figure 8·3. For simplicity, we assume that the block takes in an input feature map with dimensions $H \times W \times C$ and outputs a feature map of the same size. Following notation from the previous section, the InvRes block applies function f to activation map $X \in \mathbb{R}^{H\times W\times C}$, where f is defined as follows:

$$f = \mathcal{P} \circ \mathcal{SE} \circ \mathcal{DC} \circ \mathcal{E},$$

where $\mathcal{E}$ represents a convolution operator with 1×1 kernel size, typically with Swish(Tan and Le, 2019) or Hardswish(Howard et al., 2019) as the non-linearity. Generally, the operator increases the number of channels from say C to $6 \cdot C$. $\mathcal{DC}$ represents depth-wise convolution of size 3×3, and $\mathcal{SE}$ represents Squeeze-and-Excite operator (Hu et al., 2018). Finally, $\mathcal{P}$ represents the linear projection operator that applies a convolutional operator with 1×1 kernel and without any non-linearity.

Generally, this operator decreases the number of channels from say $6 \cdot C$ to C.

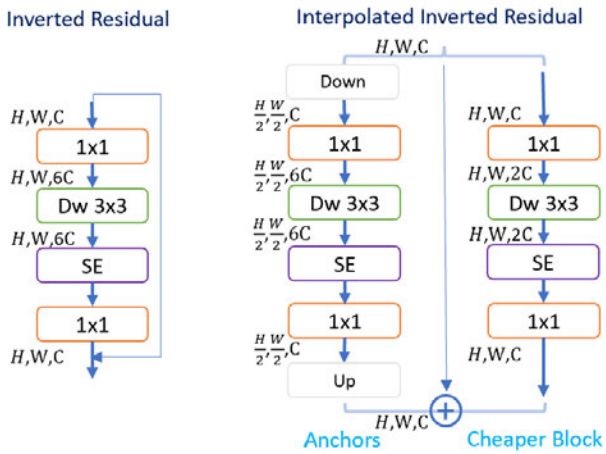


Figure 8·3: Typical Inverted Residual and Spatially Interpolated Inverted Residual blocks. Acronyms are as follows: SE (Squeeze-and-Excitation), 1×1 (Pointwise projection), Dw 3×3 (Depthwise Convolution). Note that after the first 1×1 and Dw 3×3 operation, there is a batch-norm followed by a non-linear activation such as Swish (Tan and Le, 2019) or Hardswish (Howard et al., 2019). Last 1×1 is followed by a batch-norm.

Spatially Interpolated Inverted Residual (SI-InvRes) Block. We first note that typically the largest computational footprint of InvRes block resides in the $\mathcal{E}$ and $\mathcal{P}$ operations. So, to decrease the overall computational footprint, we use much cheaper f_X operators compared to the larger 1×1 operators used by standard InvRes. Below we describe the $\mathcal{D}$ and f_A, f_X operators for SI block below:

1. *Downsampling*: Here, we apply a depth-wise convolution with stride $1/s$ with say $s = 1/2$. This implies that $A = \mathcal{D}(X)$ is of size $\frac{H}{2} \times \frac{W}{2}$.
2. f_A: Here, we apply the standard InvRes block to the anchor points. That is $f_A = f = \mathcal{P} \circ \mathcal{SE} \circ \mathcal{DC} \circ \mathcal{E}$ with essentially same set of hyper-parameters as the baseline InvRes block.
3. f_X: We apply the standard InvRes block but with much smaller *expansion* factor. That is, $f_X = f = \mathcal{P}' \circ \mathcal{SE} \circ \mathcal{DC} \circ \mathcal{E}'$ where $\mathcal{E}'$ is 1×1 convolution but where the expansion factor is much smaller than that of f_A. That is, typically, we set the expansion factor to be $2 \cdot C$, i.e., $\mathcal{E}'(X) \in \mathbb{R}^{H \times W \times 2 \cdot C}$.

Finally, after applying the two parallel operations, we upsample and add.

$$Y = \alpha \cdot \mathcal{U}(f_A(\mathcal{D}(X))) + \beta \cdot f_X(X). \tag{8.1}$$

Note that in Eq. 8.1, each output channel is an interpolation between an anchor and a cheaper block channel. Although we can use fancier merge operations such as concatenation or other non-linear transformations, this simple scheme suffices in our empirical evaluations.

8.2.3 Computational Cost Analysis

We will use Fig. 8·3 to compare the compute requirements of an interpolated residual block and its vanilla counterpart. For simplicity, let us assume the Squeeze-and-Excite (SE) operation has no cost (in practice, most of the compute cost originates in the 1×1 convolutions in the inverted residual block). Also, let the expansion factor of f and f_A be 6, and the factor of f_X is 2. The inverted residual block requires $6C^2HW$ operations for the first 1×1 projection from $C \rightarrow 6C$ space. Next, depthwise convolution with kernel K uses $6CK^2HW$ operations. Final 1×1 projection uses $6C^2HW$ operations. The overall cost is $12C^2HW + 6CK^2HW$. Typically, the number of channels is much greater than kernel size squared, i.e., $C \gg K^2$. Thus, the cost is dominated by $12C^2HW$.

In contrast, the interpolated block has two branches: (a) anchors, and (b) cheaper block. Its anchor branch downsamples the input to $(\frac{H}{2}, \frac{W}{2})$, as a result, the computational overhead associated with this branch is $\frac{12C^2HW+6CK^2HW}{4} \approx 3C^2HW$. Similarly, the cheaper branch projects only to $2C$ channels as compared to $6C$ channels, resulting in the cost $4C^2HW + 2CK^2HW$. Thus, the overall cost of these operations is close to $7C^2HW$. With additional overhead in the down-sampling, upsampling, and interpolation, we can achieve about $30 - 40\%$ savings. This can be further reduced

by using group convolutions in the cheaper block feature processing.

Remarks. There are various ways to balance the computation between anchors and cheaper block features. In this work, we follow a simple and widely applicable guideline to achieve substantial computational benefits without significant loss in performance. As a general rule in spatial interpolation, we one-third the channels in the cheaper branch that uses full input resolution and down-sample the input to half the size for processing the anchor features with the same number of channels as in the original architecture. We discuss future architecture search directions in Appendix F.5.

8.2.4 Other Residual Blocks

So far, we have created a spatially interpolated variant of the popular inverted residual block. However, our spatial interpolation technique is general and can apply to most standard convolutional blocks such as Bottleneck (He et al., 2016) and Fused-MBConv (Tan and Le, 2021) residual blocks. As a general rule, we set $f_A = f$, i.e., apply the same block as the standard block for the anchor points, while we use a cheaper version of f for f_X; usually, we use cheaper convolutions in f_X such as projection onto low dimensional space.

In this section, we create spatially interpolated variants of the Fused MBConv(Tan and Le, 2021) and Bottleneck residual blocks(He et al., 2016). These blocks are heavily used in the EfficientNetV2(Tan and Le, 2021) and Resnet(He et al., 2016) architectures respectively. Figure 8·4 shows the original residual block and its spatially interpolated variant using the similar guideline as discussed in the Sec. 8.2. We train the representative architectures that use these residual blocks in the Sec. 8.3.4 and show the results in Table 8.6. We provide the implementation details in Appendix F.4.

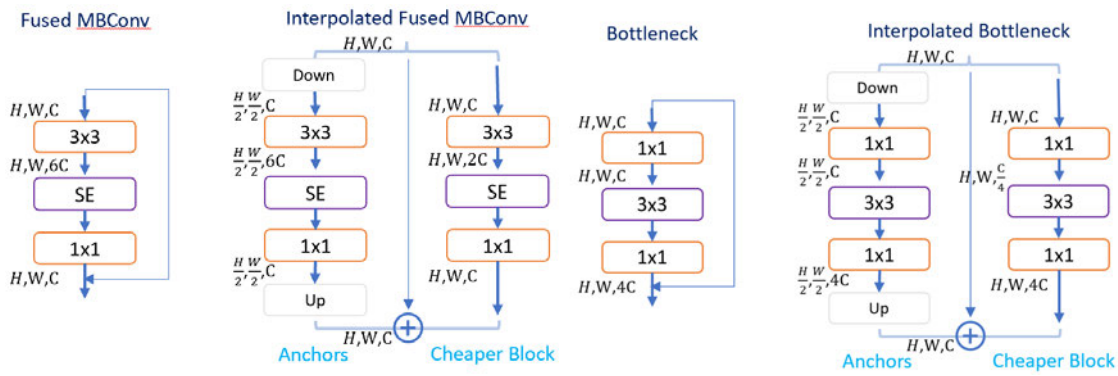


Figure 8·4: Fused MBConv and Bottleneck blocks and their interpolated variants.

8.3 Experiments

In this section, we will evaluate convolutional neural networks by replacing their residual blocks with our interpolated variants. We will show that the proposed spatial interpolation scheme yields up to 40% computational benefits without any noticeable loss in performance. First, we will benchmark spatially interpolated MobileNetV3(Howard et al., 2019) and EfficientNet(Tan and Le, 2019) models on the ImageNet dataset (Russakovsky et al., 2015). Next, we will train segmentation models using MobileNetV3 backbone on the Cityscapes dataset (Cordts et al., 2016). We will conclude this section with design choice ablations.

8.3.1 Experimental Setup

Datasets. We will study two applications in the vision domain, namely, image classification and semantic segmentation. For the classification task, we will use the large-scale ImageNet dataset A.1.5 and for the semantic segmentation task, we will use the Cityscapes dataset A.1.7.

Models. We borrow the publicly available definitions for MobileNetV3 (Howard et al., 2019) and EfficientNet (Tan and Le, 2019) architectures for image classification(Wightman, 2019). We create their interpolated variants (SI-MobileNetV3 and

Table 8.2: ImageNet Classification. We compare MobileNetV3 and EfficientNet architectures with the proposed Spatially Interpolated (SI) variants. It clearly shows that SI-MobileNetV3 and SI-EfficientNet achieve up to 40% compute reduction without any significant loss in accuracy. In addition, this improvement does not come with additional storage overhead. We report mean and deviation over 3 runs in Appendix Table F.15.

Architecture	Image Size	Accuracy	Params	MACs (Savings)	CPU (4 threads) Latency (ms)	CPU (1 thread) Latency (ms)
MobileNetV3-Large	224×224	75.2%	5.4M	219M (1.0×)	150 (1.0×)	305 (1.0×)
SI-MobileNetV3-Large	224×224	75.1%	5.2M	171M (0.8×)	110 (0.7×)	240 (0.8×)
EfficientNet-B0	224×224	76.84%	5.3M	390M (1.0×)	290 (1.0×)	626 (1.0×)
SI-EfficientNet-B0	224×224	76.75%	5.4M	264M (0.7×)	220 (0.8×)	493(0.8×)
EfficientNet-B1	240×240	78.83%	7.8M	700M (1.0×)	467 (1.0×)	950 (1.0×)
SI-EfficientNet-B1	240×240	78.84%	7.8M	477M (0.7×)	320 (0.7×)	780 (0.8×)
EfficientNet-B2	260×260	80.09%	9.2M	1B (1.0×)	730 (1.0×)	1457 (1.0×)
SI-EfficientNet-B2	260×260	79.81%	9.2M	0.7B (0.7×)	480 (0.6×)	1109 (0.8×)
EfficientNet-B3	300×300	81.5%	12.3M	1.9B (1.0×)	1377 (1.0×)	2819 (1.0×)
SI-EfficientNet-B3	300×300	81.2%	12.4M	1.3B (0.7×)	998 (0.7×)	2465 (0.9×)

SI-EfficientNet) by replacing the inverted residual blocks with our interpolated inverted residual blocks. For the semantic segmentation application on mobile devices, we use the state-of-the-art MOSAIC (Wang and Howard, 2021) segmentation model. Extensive empirical evaluation shows that this segmentation architecture achieves superior semantic image segmentation results on multiple hardware platforms including, CPU, GPU, and EdgeTPU. We replace the backbone MobileNet architecture with our interpolated variant. We follow the training setup as proposed in the baseline architectures. We refer the reader to Appendix F.2 for full architectural details, including the training hyper-parameters.

Evaluation Metrics. For characterizing the performance, we use the Top-1 accuracy on the ImageNet dataset and the mIoU metric on the Cityscapes dataset. For compute and storage characterization, we calculate the number of parameters and multiply-addition operations (MACs) required for a single image inference using standard PyTorch utilities. In addition, we benchmark the latency required for one image inference on a single CPU core with 1 and 4 threads on a server with 128GB RAM, 512GB NVMe storage, and an Intel Core-i9-9820X CPU with 3.30GHz clock

speed. This setup mimics the compute constraint typically available on resource-constrained devices.

8.3.2 Image Classification

We train the MobileNetV3(Howard et al., 2019) and EfficientNet(Tan and Le, 2019) architectures on the ImageNet dataset. These are well known for their resource efficiency compared to other architectures such as ResNet (He et al., 2016). For the MobileNet family, we use the MobileNetV3-large architecture as the baseline and create an interpolated variant (SI-MobileNetV3) by replacing the inverted residual blocks. Similarly, for the EfficientNet family, we use the EfficientNet-B0 to EfficientNet-B3 architectures as the baseline and create their interpolated variants (SI-EfficientNet). Due to our computing infrastructure limitations, we do not train the remaining larger models from the EfficientNet family. We provide the architecture details along with building blocks in Appendix F.2.

Training Details. We follow the publicly available training procedure (Wightman, 2019) to train the baseline and the interpolated architectures on four V100 GPUs. We minimize the cross-entropy loss with a weight decay of $1e-5$ using the standard RMSProp optimizer(Tieleman et al., 2012) with 0.9 momentum. We use 5 linear warm-up epochs from $1e-6$ to the learning rate 0.064 for a batch size of 1024 images. We decay the learning rate by 0.97 every 2.4 epoch. We train the MobileNet and Efficient models for 600 and 450 epochs respectively. Similar to the earlier works, we use an exponential moving average (EMA) of the trained models with a decay factor of 0.9999. Note that these hyper-parameters have been successfully used to reproduce the MobileNet and EfficientNet baselines in the PyTorch community (Wightman, 2019).

Results. Table 8.2 compares the baseline architectures and their spatially inter-

polated variants. It clearly highlights the following aspects of the proposed spatially interpolated inverted residual blocks.

(a) Computational Savings. Table 8.2 shows that interpolated architectures achieve up to 1.5× reduction in multiply-addition operations without any significant loss in performance. For instance, our interpolated variant of the EfficientNet-B0 architecture achieves similar accuracy (76.8%) at 264M MACs while the original architecture requires 390M MACs.

(b) Lower Inference Times. Since multiply-add operations do not always directly correspond one-to-one to the inference latency, we benchmark the inference latency on a single CPU core as described in Sec. 8.3.1. Spatially Interpolated architectures achieve up to 30% reduction in inference latency compared to the original architecture. Furthermore, on multi-threaded CPUs, parallelization of architecture can also be exploited to translate most of the MACs savings into latency reduction.

(c) Similar Storage Footprint. Although the interpolated residual block introduces two lightweight feature processing branches compared to a single branch in the inverted residual block, we do not add any significant storage overhead due to this change. As seen from Table 8.2, our storage requirements are on-par with the original MobileNetV3 and EfficientNet architectures. Thus, we can deploy these interpolated architectures with reduced inference latency without any additional storage footprint.

(d) Easily Integrable. The proposed interpolated inverted residual block is easily integrable within state-of-the-art architectures, as evident from Table 8.2. In addition, the interpolation scheme is agnostic to the nature of the residual block. We create interpolated variants of other convolutional residual blocks such as Bottleneck (He et al., 2016) and Fused Inverted Residual (Tan and Le, 2021) in Sec.8.2.4 and show similar benefits in the ablations (see Sec 8.3.4).

Table 8.3: Cityscapes Semantic Segmentation. We use the state-of-the-art segmentation model, MOSAIC (Wang and Howard, 2021) for mobile devices for evaluation. We replace the ImageNet pre-trained MobileNetV3 (MNV3) and Multi-Hardware MobileNet (MHMN) backbones with their spatially interpolated (SI) variants (see Appendix F.2 for architecture details). It clearly shows that spatially interpolated (SI) segmentation models yield significant compute reduction without any significant loss in mIoU metric.

Segmentation Architecture	Backbone	mIoU	Params (M)	MACs (B)	CPU (4 threads) Latency (ms)	CPU (1 thread) Latency (ms)
MOSAIC	MNV3-Small	69.12	0.61	3.46 (1.0×)	450 (1.0×)	913 (1.0×)
	SI-MNV3-Small	68.94	0.68	2.79 (0.8×)	320 (0.7×)	840 (0.9×)
MOSAIC	MNV3-Medium	71.41	1.22	7.81 (1.0×)	827 (1.0×)	1612 (1.0×)
	SI-MNV3-Medium	71.43	1.38	6.61 (0.8×)	629 (0.8×)	1367 (0.8×)
MOSAIC	MNV3-Large	74.54	1.83	10.51 (1.0×)	978 (1.0×)	1943 (1.0×)
	SI-MNV3-Large	73.6	1.76	7.79 (0.7×)	723 (0.7×)	1628 (0.8×)
MOSAIC	MHMN	75.67	1.83	20.81 (1.0×)	960 (1.0×)	2006 (1.0×)
	SI-MHMN	75.44	2.31	14.86 (0.7×)	890 (0.9×)	1785 (0.9×)

8.3.3 Semantic Segmentation

We train segmentation models on the Cityscapes dataset to further demonstrate the benefits of the spatial interpolation scheme. We implement the state-of-the-art semantic segmentation model MOSAIC (Wang and Howard, 2021) for mobile devices. We follow the publicly available implementation details to reproduce the baselines. This architecture uses many ImageNet pre-trained backbones such as MobileNetV3 (Howard et al., 2019) and Multi-Hardware-MobileNet (Chu et al., 2021). For the MobileNetV3 architecture, we train three models (large, medium, and small) to show a pareto frontier of the compute and performance characteristic curve. Note that we use the segmentation variant of these models as described in the MobileNetV3(Howard et al., 2019). It reduces the number of filters in the last few residual blocks and reduces the output stride (see Appendix F.2 for full architecture details). For the Multi-Hardware-MobileNet family, we use the MNMH architecture used in the MOSAIC paper.

Training Details. Following MOSAIC (Wang and Howard, 2021), we first pre-train the backbones on the ImageNet classification task using the training setup

described in Sec. 8.3.2. Then, we remove the classification head attached to these backbones during the segmentation training. We construct the segmentation model using the MOSAIC encoder-decoder architecture. Similar to (Wang and Howard, 2021), we use the polynomial learning rate decay and minimize the cross-entropy loss at the pixel level for the Cityscapes classes with a weight decay of 0.00001. We minimize the loss using the SGD optimizer with 0.9 momentum and a learning rate of 0.1 with a batch size of 32. We use four A100 GPUs to train this segmentation model. We train all the models, including baselines up to 1000 epochs with a single input image scale 1024×2048. We use the standard data augmentations (Wang and Howard, 2021; Howard et al., 2019) in all our experiments.

Results. Table 8.3 enlists the performance stats and the mIoU of various segmentation models with varying degree of computational and storage requirements. Below, we highlight the benefits of the spatial interpolation scheme on the semantic segmentation.

(a) Computational Savings. Our spatially interpolated MobileNetV3 backbone yields similar computational benefits on the Cityscapes segmentation task as the image classification task. The interpolated architectures achieve up to 1.4× reduction in compute without any significant loss in the mIoU metric. For instance, segmentation model with MNMH backbone achieves 75.67mIoU with 20.81B MACs, while the interpolated model achieves 75.44mIoU with 14.86B MACs.

(b) Lower Inference Times. We evaluate the inference time of various backbones with MOSAIC segmentation model on a single CPU core in Table 8.3. Interpolated architectures achieve up to 30% reduction in inference latency as compared to their original counterparts.

(c) Similar Storage Footprint. Similar to image classification, spatially interpolated architectures require no additional storage to achieve the baseline mIoU at a

reduced computational footprint.

8.3.4 Ablations

In this section, we perform ablations to probe various aspects of the spatial interpolation scheme.

(A) Baselines with smaller resolution. In the ImageNet experiments (Sec. 8.3.2 and Table 8.2), we have shown that spatially interpolated architectures dominate in the compute and accuracy trade-off. In this experiment, we match the computational budget of the baselines to our interpolated variants by reducing their input resolution. For instance, by reducing input resolution from 224 to 192, we achieve MobileNetV3-large model that has a similar computational footprint as our interpolated variant (see Table 8.4). Although we can reduce the budget, the interpolated architecture still outperforms this model by 1.4% Top-1 accuracy. A similar observation holds true for the EfficientNet architecture. Thus, the proposed architectures yield superior accuracy vs computational trade-off than the baselines.

Table 8.4: Baselines with lower resolution. We reduce the input resolutions for the baseline architectures to measure the accuracy vs compute trade-off. It clearly shows that there is a significant gap between spatially interpolated architectures and the baselines with reduced computation.

Architecture	Resolution	Accuracy (%)	Params (M)	MACs (M)
MobileNetV3-Large	224	75.2	5.4	219
MobileNetV3-Medium	224	73.3	4.0	155
MobileNetV3-Large	192	73.7	5.4	160
SI-MobileNetV3-Large	224	75.1	5.2	171
EfficientNet-B0	224	76.8	5.3	390
EfficientNet-B0	192	75.3	5.3	271
SI-EfficientNet-B0	224	76.8	5.4	264

(B) Hard 0-1 weights on Anchors and Cheaper block. In the interpolation scheme, we learn coefficients α and β that combine the anchors and the cheaper block features. In this ablation, we study the effect of each branch in isolation. We use two configurations to study this: only anchors ($\alpha = 0, \beta = 1$) and only cheaper

features($\alpha = 1, \beta = 0$). As a reference, ($\alpha = 1, \beta = 1$) refers to the standard spatial interpolation scheme used in the ImageNet and Cityscapes experiments. Table 8.5 shows the accuracy and resource usage for the MobileNetV3-large architecture for all of these configurations. It shows that at a fixed computational budget, using only anchors or only cheaper branch is not beneficial as compared to combining these two feature branches.

Table 8.5: Only keep Anchors or Cheaper branch. Note that in the configuration $(1, 1)$ we learn the addition coefficients for Anchors and Cheaper. While in the the other two configurations we only keep one branch or the other. It shows that at a fixed computational budget, using only anchors or only cheaper branch is not beneficial as compared to combining these two feature branches.

Architecture	Anchors Weight	Cheaper Weight	Accuracy (%)	Params (M)	MACs (M)
SI-MobileNetV3-Large	1	1	75.1	5.2	171
SI-MobileNetV3-Large	0	1	72.6	3.69	135
SI-MobileNetV3-Large	0	1	73.9	4.12	190
SI-MobileNetV3-Large	1	0	70.1	4.66	82
SI-MobileNetV3-Large	1	0	73.7	7.93	178

(C) Other Spatially Interpolated Blocks. So far, in the empirical evaluations we have mainly focused on improving the inverted residual block. Since, our interpolation scheme is generic as shown by the extension in Sec. 8.2.4, we apply the interpolation scheme to reduce the computational footprint of the Bottleneck(He et al., 2016) and Fused-MBConv (Tan and Le, 2021) residual blocks. We create interpolated variants of the Resnet50 and EfficientNetV2-s architectures that utilize Bottleneck and Fused-MBConv residual blocks respectively. We train these networks on the ImageNet dataset using the training procedure described in (Wightman, 2019). For EfficientNetV2-small architecture we follow a similar training setup as EfficientNet architectures in Sec. 8.3. We describe the training procedure for Resnet50 in Appendix F.3. We show the results in Table 8.6. It shows that the spatially interpolated architectures achieve similar Top-1 accuracy at a 1.5× reduction in computational cost.

Table 8.6: Spatially Interpolated ResNet-50 and EfficientNetv2-small.

Architecture	Resolution	Accuracy (%)	Params (M)	MACs (B)
ResNet-50	224	79.51	25.56	4.09
ResNet-50 (ours)	224	79.19	27.51	2.67
EfficientNetv2-s	384	83.88	21.46	7.96
EfficientNetv2-s (ours)	384	83.45	21.84	4.93
EfficientNetv2-l	480	85.67	118.52	54.96
EfficientNetv2-l (ours)	480	85.23	140.1	36.35

(D) Effect of Anchors and Cheaper Block Features. In this ablation, we study the impact of number of anchors and cheaper block features on the spatial interpolation. We pick the MobileNetV3-large architecture and train networks with different number of anchors and cheaper features. We pick the number of cheaper features from the set $\{1, \frac{1}{2}, \frac{1}{3}\}$, which means the fraction of channels kept in the cheaper branch as compared to the original MobileNetV3-large architecture. Similarly, we pick the number of anchors from the set $\{1, 2, 3\}$, where $x = 1$ makes every pixel as an anchor, and $x = 2$ reduces the image size into half, and so on. Note that the configuration Anchors=1 and Cheaper=1 will be the most computationally and storage wise expensive. Also note that changing number of anchors does not affect the storage as it only impacts the computational aspects of the architecture.

We present these results in the Table 8.7. It shows that our design choice of balancing anchors and cheaper features (see Sec. 8.2.2) provides a good trade-off between accuracy, computation, and storage footprint.

(E) Choice of upsampling operator. In the anchor branch, there are multiple choices for the upsampling operator. We tried out the following choices: proposed nearest, bilinear, and bicubic interpolation. Table 8.8 shows that our default upsampling operator fairs well in the accuracy vs latency trade-off.

Table 8.7: Vary number of anchors and cheaper features. Anchors (x=1,2,or 3 refers to the number of anchors selected at every x location. $x = 1$ simply means that every pixel is an anchor, $x = 2$ means the image has been halved in the resolution and so on.). Cheaper features ($\frac{1}{3}$, $\frac{1}{2}$, 1) refers to the amount of reduction in the number of channels as compared to the original Inverted Residual block configuration in the MobileNetV3-large architecture. 1 means the same number of channels as in the original architecture. $\frac{1}{2}$ means that the number of channels have been halved, and so on. Note that the configuration Anchors=1 and Cheaper=1 will be the most computationally and storage wise expensive. Also note that changing number of anchors does not affect the storage as it only impacts the computational aspects of the architecture.

Architecture	**Anchors**	**Cheaper**	**Accuracy (%)**	**Params (M)**	**MACs (M)**
SI-MobileNetV3-Large	2	-	75.1	5.2	171
SI-MobileNetV3-Large	2	$\frac{1}{3}$	74.2	4.8	146
SI-MobileNetV3-Large	2	$\frac{1}{2}$	74.9	4.95	168
SI-MobileNetV3-Large	2	1	76.2	5.41	230
SI-MobileNetV3-Large	3	$\frac{1}{3}$	73.6	4.8	127
SI-MobileNetV3-Large	3	$\frac{1}{2}$	73.9	4.95	149
SI-MobileNetV3-Large	3	1	75.4	5.41	212
SI-MobileNetV3-Large	1	$\frac{1}{3}$	75.7	4.8	276
SI-MobileNetV3-Large	1	$\frac{1}{2}$	76.1	4.95	298
SI-MobileNetV3-Large	1	1	76.7	5.41	361

Table 8.8: Choice of Upsampling Operator. We tried out other upsampling schemes in the anchor branch. This table clearly shows that more complex schemes yield somewhat better performance but they fall behind in their accuracy vs latency trade-off.

Architecture	**Upsampling**	**Accuracy (%)**	**CPU(1 thread) Latency (ms)**
SI-EfficientNet-B0	nearest	76.75	493
SI-EfficientNet-B0	bilinear	76.85	540
SI-EfficientNet-B0	bicubic	77.2	681

8.4 Discussion

In this work, our focus has been on leveraging spatial smoothness in convolutional feature maps. This notion comes naturally to these features since the activations have spatial dimensions. It is worth noting that the interpolation scheme is pretty generic and is applicable to other architectures where an asymmetry exists. For instance, in a transformer architecture, the bulk of the computation arises due to long sequence lengths. In such cases, we can split this computation into anchors and

cheaper features in a similar logic. Anchors operate on partial sequences, and the cheaper branch operates on the entire input sequence.

Due to a limited computational budget, many design choices were made to keep the storage/compute costs low. It geared the experiments towards achieving similar performance as the original architectures with a reduced model footprint. In the presence of a large compute budget, these restrictions should be lifted to achieve better accuracy at the same footprint. More so, neural architecture search should be leveraged to find the optimal configurations in the anchor and cheaper feature branches.

Chapter 9

Distributionally Constrained Learning (DCL)

9.1 Introduction

The success of Deep Neural Networks (DNNs) in a wide range of applications has motivated theoreticians to propose various complexity measures to bound the gap between training and test error (generalization gap). These include carefully crafted bounds such as PAC-Bayes bounds (Dziugaite and Roy, 2017), mutual-information bounds (Bu et al., 2020), VC-dimension (Bartlett et al., 2019), Rademacher complexity (Mohri et al., 2018), and path-norm bounds (Neyshabur et al., 2015). Others have characterized generalizability by examining the quality of the realized solution by virtue of training, such as convergence to flat-minima (Keskar et al., 2017) or implicit regularization of stochastic gradient descent (SGD) (Neyshabur, 2017). In parallel, (Jiang et al., 2020) empirically examined the diverse set of complexity measures and tabulated the correlation between these measures and the generalization gap observed in DNNs trained with CE loss.

Distributionally Constrained Learning. Drawing upon insights from the aforementioned prior works, we propose to enforce data-dependent constraints during training, so that both the solution trajectory as well as the loss landscape favors generalizability. Specifically, we propose the following two constraints that are relatively easy to

optimize:

- *Input-Variability* constraint reduces variance in model predictions on inputs from the same class.
- *Model-Variability* constraint reduces variance in model predictions due to training on random data.

While input-variance and model-variance bound the generalization gap, a fundamental challenge arises in operationalizing these constraints during training. DNN models tend to overfit training data (Zhang et al., 2017), and as such, the manifested loss variances observed over inputs or models tend to be negligibly small.

Generalization Proxies. To overcome overfitting, we propose KL divergence measures in the logit space as proxies. First, we establish that our proposed divergence measures are sound measures for generalization. Next, we show empirically that at a suitable temperature parameter, the KL divergences on augmented training data closely track the behavior of test data across epochs. These result in concrete constraints during training yielding better generalization. As these constraints are data-dependent, we refer to these constraints as distributional constraints.

Cosine-Scheduled Multi-Phase Constrained Training. Typically, neural networks incorporate constraints through a regularization term where a suitable hyper-parameter imposes the penalty for constraint violation. This parameter balances a trade-off between the primary objective (such as classification loss) and the constraint. Such a strategy also requires careful and extensive tuning of the hyper-parameter to achieve an optimal balance between the primary and side objective. We show that such a strategy yields suboptimal performance (see empirical evaluations in Sec. 9.5).

To address this issue, we propose to enforce proxy constraints gradually. Specifically, we design a cosine scheduled exploration strategy that promotes exploration in the beginning while allowing for gradual hardening of constraints. Further, since

the budget for a constraint requires careful design, we develop a multi-phase strategy to dynamically update the budget. The proposed strategy starts with a conservative budget in the initial phase. It halves the budget every time the neural network satisfies the constraint and starts the next phase with an updated budget and halved learning rate. This process repeats till there is an improvement in performance. Such a simple strategy yields superior performance and alleviates the burden of careful budget hyper-parameter tuning.

Experimental Results. We ran experiments on several benchmark datasets (CIFAR-100, TinyImageNet, ImageNet-1K) and architectures (ShuffleNetV2, Resnet-18, Resnet-50, and ViT-Tiny) and show that our proposed input and model variability proxies together with our multi-phase cosine scheduled exploration lead to improved SOTA performance. In particular, on CIFAR-100, our results, obtained without any pre-training, can realize performance gains similar to those that utilize pre-training. In general, pre-training results in a computational bottleneck while adapting to target data. Our proposed method suggests that we can overcome these bottlenecks through cleverer training. On ImageNet-1K using the ViT-Tiny model, we gain roughly 2% compared to standard CE training.

Contributions. We list the salient contributions of our paper.

- *Distributional Constraints.* We propose to train with data-dependent constraints. Different from data independent constraints such as L2 norm on parameters, data-dependent constraints adapt to the input distribution. We identify input-variability and model-variability as critical components for enhancing generalizability.

- *Cosine-scheduled Constrained Learning.* In contrast to prior works that incorporate side objectives by a fixed Lagrangian parameter, which is then hyperparameter tuned, we adopt *Cosine Scheduled Exploration* that encourages exploration and exploitation trade-offs during training while enforcing the constraints. Further, we

design a *Multi-Phase Budget Update* strategy to dynamically adjust per constraint budget in phases, starting with a conservative budget and adaptively lowering it until no further performance improvement is observed.

- We demonstrate SOTA performance on diverse benchmark datasets and architectures. We show that training from scratch, without pretraining, can realize similar accuracies as models that incorporate pretraining, thereby overcoming the computational overhead of adapting pretraining to target data. Our PyTorch implementation is available at https://github.com/anilkagak2/DCL_Distributionally_Constrained_Learning

9.2 Related Works

Input-variability (IV). Prior works on semi-supervised learning have proposed consistency loss (Volpi et al., 2018; Xie et al., 2020; Sohn et al., 2020; Laine and Aila, 2017; Sajjadi et al., 2016) to inductively bias model predictions to be consistent under reasonable modifications to the input. (Sajjadi et al., 2016) predicts the same label amidst stochasticity in dropout, random max-pooling, and randomized affine data augmentations. (Volpi et al., 2018) preserves predictions in the presence of perturbations such as adversarial noise. (Xie et al., 2020) improves upon these works to include advanced data augmentations (such as RandAugment(Cubuk et al., 2020)) as multiple viewpoints of the underlying input across which the model prediction remains consistent. (Sohn et al., 2020) uses weak augmentation to preserve the label information and maintains consistency by penalizing the model prediction on strong data augmentation if it differs from weak augmentation prediction. Similarly, Π model (Laine and Aila, 2017) penalizes when the model predictions differ between random data augmentations and dropout for the same input, while Temporal Ensembling (Laine and Aila, 2017) uses an average of predictions for various viewpoints of the

data during the training trajectory and this prediction is used as the target label. The survey (Yang et al., 2022) presents an in-depth review.

Model-variability (MV). In parallel, (Tarvainen and Valpola, 2017; Grill et al., 2020; Caron et al., 2021; Hochreiter and Schmidhuber, 1997a; Foret et al., 2021; Cha et al., 2021; Chen et al., 2020b; He et al., 2020) have also explored variance arising due to different model views. Mean-Teacher(Tarvainen and Valpola, 2017) regularizes model predictions to be close to the teacher predictions, which is an exponential moving average (EMA). Self-supervised learning (Caron et al., 2021; Grill et al., 2020) explores a similar penalty in the absence of ground truth. Typically, the regularization minimizes the difference between different views of the input forwarded through the EMA and model. Although this requires careful design considerations to avoid representation collapse, such a strategy successfully learns resilient latent feature space without labels. In parallel, we can also view model invariability from the flat/wide minima (Hochreiter and Schmidhuber, 1997a) perspective in the parameter space. Many works have promoted this notion, such as sharpness-aware minimization(Foret et al., 2021), averaging of the stochastic gradient models(Izmailov et al., 2018), etc.

In contrast to these prior works, we formalize the fact that IV and MV *jointly* characterize generalizability. We develop distributional proxies for IV and MV and prove that our proxies bound excess loss (see Sec.9.4.1). Then we propose a method that individually penalizes both IV and MV.

Enforcing Constraints as Side Objective. While prior works have incorporated the above-described regularization measures, they are mostly enforced using a fixed Lagrangian (hyper-parameter) that trades off the main objective (such as classification loss) for the amount of violation in the regularization measure. Our experiments in Sec. 9.5 demonstrate that such a strategy is sub-optimal. In a highly non-convex domain such as DNN training, (Sun and Sun, 2021) does not recommend

the traditional ascent-descent mechanism in the constrained formulation. Inspired by this, (Kag et al., 2023a) proposed using an epoch-varying Lagrange parameter instead of a fixed one. (Kag et al., 2023a) empirically shows that dynamic parameter leads to better results in knowledge distillation setting. We further extend the idea by introducing training in phases to improve generalization. We encourage exploration in early training rounds using a scheduler on the Lagrange parameter, where we gradually increase its magnitude. We train our model using phases and a target budget to recover generalizable minima smoothly.

9.3 Intuitive Justification

We present a synthetic example to motivate how distributional constraints lead to generalization.

Example. Consider a regression problem with an average loss function depicted in Figure 9·1. The loss for each data point, i, is a combination of two quadratic functions (see G.1 for details). Figure 9·1 depicts individual losses for different data points ($\ell_i(\theta)$, shadow curves) and average loss ($E_i\ell_i(\theta)$, blue curve) where θ is the parameter we would like to learn.

Generalizable Empirical Minima. Let us examine the different optima, depicted as the black and red circles in Fig. 9·1. The loss corresponding to the black-circle is lower than the loss suffered with the red-circle. Nevertheless, we may prefer the red-circle as the more generalizable solution. This can be understood by considering two uncertain components,

(A) *Randomness in training data S.* This leads to modification of the learner's loss-landscape relative to the ground-truth loss-landscape (blue-curve in Fig. 9·1) resulting in a random trained model θ_S. The landscape around the red-optima being relatively flat, the shifts due to stochasticity of training examples are relatively small.

(B) *Randomness of test input instances, $i \sim \rho$.* This leads to randomness in observed loss, $\ell_i(\theta)$. The effect of this source of randomness are depicted as shadow-curves in Fig. 9·1—the loss landscape for a single example. Comparing black vs. red optima, we notice that the black-optima has a significantly higher loss variance, and in particular, there is a 15% chance that a randomly chosen example will result in a loss larger than 1000. On the other hand, the red-optima has only a 2.5% chance of resulting in a similar loss.

Our goal is to minimize $E_{i,S}[\ell_i(\theta_S)]$, and to do so, we can consider the divergence of the empirical estimate from the true loss. Chebyshev's inequality asserts that, under the probability law, ρ, with probability greater than $1-\delta$, $E_{i,S}[\ell_i(\theta_S)] \leq \ell_i(\theta_S) + \sqrt{\frac{\mathrm{Var}(\ell_i(\theta_S))}{\delta}}$. By appealing to the law of total variance(Weiss et al., 2005), we can decompose the variance term as follows: In particular, w.p $> 1-\delta$,

$$E_{i,S}[\ell_i(\theta_S)] \leq \ell_i(\theta_S) + \sqrt{\frac{E_S[E_{i,j|S}(\ell_i(\theta_S) - \ell_j(\theta_S))^2]}{\delta}} + \sqrt{\frac{E_{S,S'}[(E_{i|S}\ell_i(\theta_S) - E_{i|S'}\ell_i(\theta_{S'}))^2]}{\delta}}$$

where S, S' are two training datasets; i, j are inputs; all sampled from unknown distribution ρ.

Surrogate Objective. The three terms in the RHS correspond to training error, the loss variances induced by randomness of input examples and training data respectively. However, for LHS to be meaningful we need to choose examples i to be independent of S. While this leads to further simplification of terms in the RHS, this expression is not tractable as a surrogate objective for training! We resolve this issue by first bounding the variance terms in terms of KL divergences (see Lemma 1). Second, we empirically verify (Fig. 9·3a, 9·3b, 9·3c) that divergence expressions and

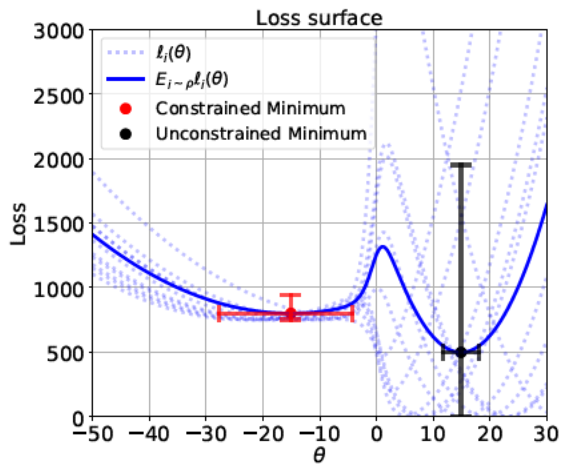


Figure 9·1: Plots of empirically averaged loss-landscape depicting two minima. Shadow curves depict loss-landscape for randomly chosen data samples. Red-minima exhibits relatively small loss variance induced by random training data or test examples.

accuracy evaluated on augmented training data closely follow test data. In summary, our distributional surrogate objective with augmented data leads to better generalization.

9.4 Method

In this section, we will describe a DNN training algorithm to enforce distributional constraints.

Notation. For simplicity, we will focus on a $K-$class classification problem with $\mathcal{X}$ and $\mathcal{Y}$ being the input-output spaces. Assume that we have access to a training set of N i.i.d. data points $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We will parameterize the neural networks with parameters θ. Given an input $\mathbf{x} \in \mathcal{X}$, we will use $f(\theta, \mathbf{x})$ to denote the $K-$dimensional network output representing the predictive probabilities for the K classes. We will use the symbol $\Delta(\cdot, \cdot)$ as the measure of distance between two probability distributions (in this work, we will mainly

use the Kullback-Leibler (KL) divergence as the distance measure by softening with temperature τ).

We will use $\ell\left(y, f(\theta, \mathbf{x})\right)$ to denote the loss of a datapoint, i.e., Cross-Entropy loss. We consider the empirical risk minimization as $\min_\theta \frac{1}{N} \sum_i \ell\left(y_i, f(\theta, \mathbf{x}_i)\right)$.

9.4.1 Distributional Constraints

In Sec. 9.3, we describe distributional constraints that allow for control of the generalization gap. However, imposing constraints on the loss variances is impractical. It is due to:

1. *Simulating Test-Data.* Our notions of input and model variability are based on evaluating examples that are independent of training data. Since we do not have access to such examples, we need proxies, which are evaluated on training data but simulate the effect of test data.

2. *Overfitting.* DNNs tend to overfit, so the loss variances observed for a model on different inputs tend to be negligibly small. Therefore, we need proxies for the loss variances.

KL Divergences as Proxies. As it turns out, at a suitable temperature setting τ, KL divergences on the K-dimensional predictive probabilities, $f(\theta, \mathbf{x})$ satisfies our criteria (1) and (2), namely, these divergences do not overfit and simulate the effect of test data. Additionally, KL divergences serve as upper bounds for loss variances (see Lemma 1). Below, we formally describe the constraints.

- **Input-Variability.** We define input variability $\mathcal{I}_v(\mathbf{x}, \theta)$ as the distance between predictive probabilities of the two different data augmentations $\mathbf{x}^1$ and $\mathbf{x}^2$ corresponding to the input $\mathbf{x}$, i.e.,

$$\mathcal{I}_v(\mathbf{x}, \theta) = \Delta(f(\theta, \mathbf{x}^1), f(\theta, \mathbf{x}^2)) \tag{9.1}$$

This metric measures the variability of the DNN predictions w.r.t. different augmentations of the input. By enforcing $\mathcal{I}_v(\mathbf{x}, \theta)$ to be less than δ during DNN training, we can ensure that the trained network will be robust to input perturbations up to a certain degree. Thus, the training process will explore a more robust loss landscape. In this work, we focus on standard data augmentations and leave augmentations such as adversarial perturbations to future work.

- **Model-Variability.** Our input-variability constraint ensures that the network explores a trajectory that is flat w.r.t. input perturbations. In contrast, model-variability promotes flatness in the loss landscape w.r.t. network parameters. We will track another set of parameters $\hat{\theta}$ to measure model-variability. These weights could be an exponential moving average (EMA) of the DNN or a checkpoint from earlier point in training trajectory. We define model-variability $\mathcal{M}_v(\mathbf{x}, \theta)$ as

$$\mathcal{M}_v(\mathbf{x}, \theta) = \Delta(f(\hat{\theta}, \mathbf{x}), f(\theta, \mathbf{x})) \tag{9.2}$$

Model-variability is related to flatness in the loss landscape w.r.t. parameters θ. As such, by enforcing $\mathcal{M}_v(\mathbf{x}, \theta) \leq \delta$, the output does not change significantly within the vicinity of the parameter θ. As a result, the constraint ensures that the trajectory explores flat loss landscape.

One strategy is to set $\hat{\theta}_t$ as the EMA model defined as $\hat{\theta}_t = \alpha\hat{\theta}_{t-1} + (1-\alpha)\theta_t$, where $\alpha \in [0, 1]$ is the exponential decay parameter and $\hat{\theta}_0$ is initialized randomly. Another strategy is to set $\hat{\theta}_t = \theta_{t-1}$, i.e., the model from the previous iteration. In batched training, one can use the previous epoch model.

We can collect many such constraints in the vector $\mathcal{G}(\mathbf{x}, \theta)$ and associate a budget δ as follows

$$\mathcal{G}(\mathbf{x}, \theta) = \begin{bmatrix} \mathcal{I}_v(\mathbf{x}, \theta) \\ \mathcal{M}_v(\mathbf{x}, \theta) \end{bmatrix}; \;\; \delta = \begin{bmatrix} \delta_{\mathcal{I}_v} \\ \delta_{\mathcal{M}_v} \end{bmatrix} \tag{9.3}$$

Next, we show that KL divergences are effective proxies for loss-variances.

Lemma 1. *Suppose for each example* $(\mathbf{x}, y) \sim \mathcal{D}$*, the ground-truth component of networks with parameters* θ *and* $\hat{\theta}$ *satisfy* $0 < \epsilon \leq f_y(\hat{\theta}, \mathbf{x}) < f_y(\theta, \mathbf{x})$*, it follows that*

$$|CE(\hat{\theta}) - CE(\theta)| \leq \sqrt{\frac{1}{2\epsilon^2} E_{x\sim\mathcal{D}}[D_{KL}(f(\hat{\theta}, \mathbf{x}) \| f(\theta, \mathbf{x}))]}$$

where, $CE(\cdot)$ *is the average cross-entropy loss on* $\mathcal{D}$*.*

Remark. A similar result holds for the input-variability proxy described by Eq. 9.1 and is omitted. Note that the conditions in our Lemma are validated by the fact that DNNs overfit on training data, and as such, the trained network $f(\theta, \mathbf{x})$ closely fits the ground-truth label. The EMA model defined by, $\hat{\theta}$, tends to be somewhat less confident in predicting ground-truth labels. We demonstrate empirically (see Fig. 9·3a, 9·3b) that KL divergences on training examples, at a suitable temperature, mimics test-data evaluations, and thus serve as effective proxies.

9.4.2 Constrained Learning

Having formalized the distributional constraints, we describe the constrained learning objective. We can write the empirical risk associated with the DNN and the training data as

$$\mathcal{L}(\mathcal{D}, \theta) = \frac{1}{N} \sum_i \ell(y_i, f(\theta, \mathbf{x}_i)) \tag{9.4}$$

where $\ell(\cdot, \cdot)$ is the cross-entropy loss. We can formally write the constrained objective as follows:

$$\min_\theta \mathcal{L}(\mathcal{D}, \theta) \quad \text{s.t.} \quad \mathcal{G}(\mathcal{D}, \theta) \leq \delta, \tag{9.5}$$

where $\mathcal{G}(\mathbf{x}, \theta)$ defines constraints such as input and model variability, and δ is the budget associated with these constraints.

Algorithms. We propose two variants to incorporate distributional constraints.

(A) Fixed Lagrangian (FL). We construct a Lagrangian that combines the empirical risk and the distributional constraints with a fixed trade-off hyper-parameter λ as $\mathcal{L}(\mathcal{D}, \theta) + \lambda \|\mathcal{G}(\mathcal{D}, \theta)) - \delta\|^2$. A grid search over various choices of λ is conducted to find the optimal hyper-parameter.

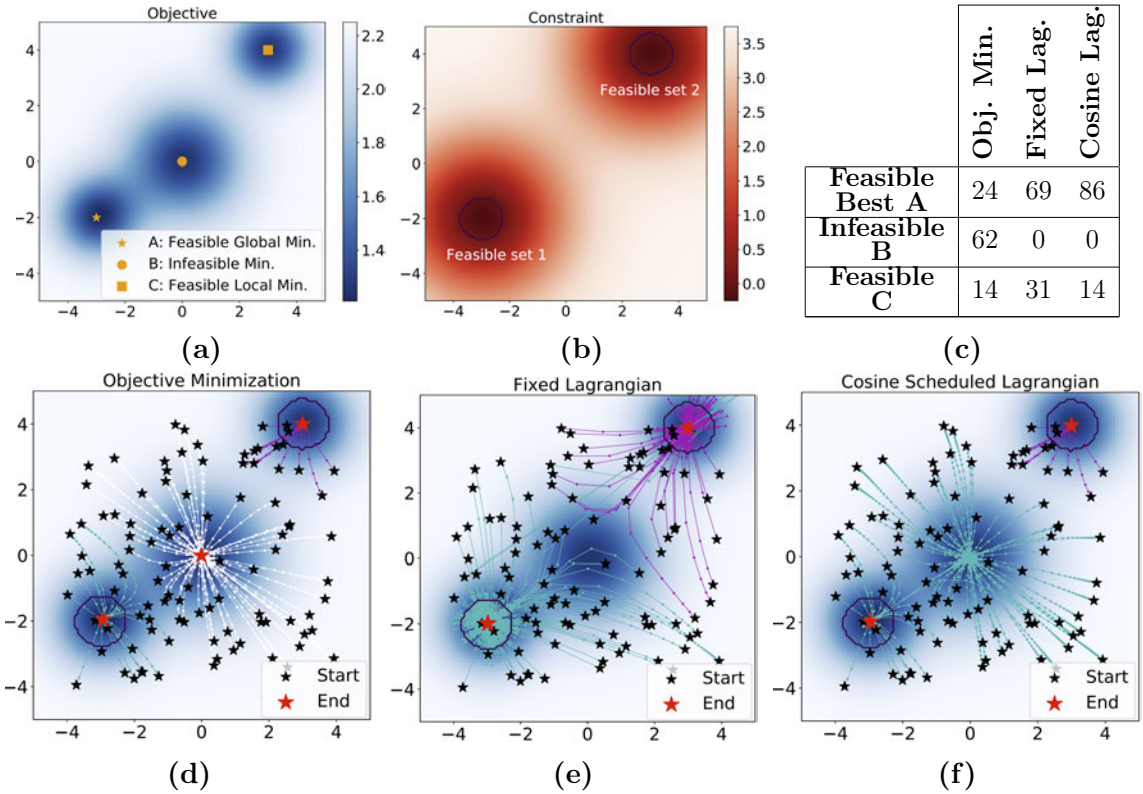
(B) Distributionally Constrained Learning (DCL). Our experiments show that a Fixed Lagrangian strategy yields sub-optimal performance (see Sec. 9.5). Similarly, although it might be tempting to enforce constraints using ascent-descent schemes, recent works(Sun and Sun, 2021; Kag et al., 2023a) have proposed alternatives to such strategy in the non-convex regime. Inspired by this, we adopt the *Cosine Scheduled Exploration* (Kag et al., 2023a) as it encourages exploration during the early epochs in training and allows exploitation towards the end, where it enforces the constraint. We extend this scheme by introducing a phase update strategy (explained below) which adaptively updates the budget in phases. Thus, DCL dynamically adjusts the trade-off parameter using a cosine schedule to balance exploration-exploitation of the training trajectory and ramps up the trade-off significantly so that constraints are strictly enforced at termination.

Algorithm 5 Distributionally Constrained Learning (DCL)

Input: Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
Parameters: $\lambda_{\max}$, λ_T cosine period, Budget δ,
Parameters: Epochs E, Learning Rate η, α, τ
Initialize: Randomly initialize network θ, EMA $\hat{\theta}$
Set phase counter $M = 0$
repeat
  Increase phase counter M by 1
  for $e = 1$ **to** E **do**
    Randomly Shuffle Dataset $\mathcal{D}$
    $\lambda \leftarrow \lambda_{\max} \times \frac{(1-\cos \frac{e \mod \lambda_T}{\lambda_T}\pi)}{2}$
    // batched gradient descent to minimize
    $\theta \leftarrow \arg\min_\theta \mathcal{L}(\mathcal{D}, \theta) + \lambda \|\mathcal{G}(\mathcal{D}, \theta)) - \delta\|^2$
    $\hat{\theta} \leftarrow \alpha\hat{\theta} + (1-\alpha)\theta$
  Compute current constraint value $\hat{\delta}$
  $\delta \leftarrow \left(\frac{\delta}{2} \textbf{ if } \hat{\delta} < \delta, \text{ i.e., Constraint is satisfied } \textbf{else } \hat{\delta}\right)$
  Half the learning rate $\eta \leftarrow \frac{\eta}{2}$
until convergence.
Return : θ, $\hat{\theta}$

Sensitivity to Budget. A crucial design point in the constrained objective in Eq. 9.5 is the budget δ that DCL requires. DCL can be sensitive to the choice of this hyper-parameter. For instance, a large budget would end up simply ignoring the constraint and would focus mainly on the cross-entropy term. In contrast, a small budget would be inconsistent with the cross-entropy objective and might not be compatible with any of its local minima. Thus, we need a scheme to carefully adjust the target budget for the constraints.

Budget Update in Phases. We solve this issue by performing constrained learning in multiple phases. In the initial phase, we start with a conservative budget, i.e., either the constraint budget achieved by the cross-entropy solution or any reasonably high budget value. In the subsequent phases, we evaluate the constraint and verify if the learning algorithm satisfies the earlier budget. In this case, we halved the budget. In case, the budget is not satisfied, we set the budget as the current value of the



	Obj. Min.	Fixed Lag.	Cosine Lag.
Feasible Best A	24	69	86
Infeasible B	62	0	0
Feasible C	14	31	14

Figure 9·2: 2D toy example with Gaussian distribution. Case study of cosine scheduling in finding the optimal feasible minimum. 9·2a: Objective loss landscape with three minima. 9·2b: Constraint function with two feasible sets. 9·2d,9·2e,9·2f: Trajectory of different methods with different initializations. 9·2c: Convergence percentages to different minima for 100 differently initialized runs. Objective minimization can converge to infeasible minima. Cosine scheduled Lagrangian converges to the optimal feasible point more than the fixed Lagrangian.

distributional constraints. We start the next phase with the same hyper-parameters, an updated budget, and halved learning rate. We run this multi-stage training till the performance metric improves.

We provide the end-to-end pseudo code in the Algorithm 5. We provide some intuition below.

Example 2. Consider a 2D function minimization problem. The objective plot shown in Figure 9·2a consists of three Gaussian functions. Figure 9·2b shows the corresponding constraint plot we want to satisfy in minimizing the objective. We refer to Appendix for an explicit form of the functions. There are three local minima:

A- a feasible global minimum, B- an infeasible minimum, and C- a feasible local minimum.

Comparison of methods. We compare direct function minimization, a fixed Lagrangian minimization, and a cosine scheduled Lagrangian minimization. We optimize the methods using gradient descent for 200 iterations. Figure 9·2d, 9·2e, 9·2f show the trajectories of the methods for 100 random initializations. Table 9·2c reports the percentages of the convergence to different minima. As seen from the figure and the plots, cosine scheduled Lagrangian converges to the best feasible point 86% of the time, whereas the best competitor, .i.e, fixed Lagrangian, converges 69% probability. Hence, our method can escape unwanted local minima and converges to the feasible global minima more often than the competitors.

9.5 Experiments

In this section, we evaluate the proposed algorithms, FL and DCL on various datasets and architectures. Additionally, we perform ablations to highlight their behavior.

9.5.1 Experimental Setup

Datasets. We consider publicly available image classification datasets: (a) CIFAR-100 (Krizhevsky and Hinton, 2009) consists of 50K training and 10K test images from 100 classes with size $32 \times 32 \times 3$, (b) Tiny-ImageNet (Le and Yang, 2015) contains 100K training and 10K test images from 200 classes with size $64 \times 64 \times 3$, and (c) ImageNet-1K (Russakovsky et al., 2015) consists of 1.2M training and 100K test images from 1000 classes with size $224 \times 224 \times 3$. We provide the dataset setup in detail in Appendix A.1.

Baselines. We report the performance of following methods

- **CE:** Learning with standard cross-entropy loss function.

- **DCL (ours):** Learning with distributional constraints.
- **FL (ours):** DCL with fixed Lagrangian hyperparameter.

We include three constrained setups: (a) input-invariability, (b) model-invariability, and (c) both. In addition, we list the performance of ImageNet pre-trained models fine-tuned on the CIFAR-100 dataset. We enumerate the fine-tuning procedure in Appendix J.3. Note that the ImageNet pre-trained baselines require input resolution to be $224 \times 224 \times 3$ while the rest of the baselines only require $32 \times 32 \times 3$ resolution. As a result, these baselines are computationally very expensive for inference.

Models. We evaluate ResNet(He et al., 2016) and ShuffleNetV2(Ma et al., 2018) architectures on these datasets. In particular, we benchmark ResNet18, ResNet50, and ShuffleNetV2-1$\times$ models. We provide their architectural details in Appendix J.3. In addition, we list their resource requirements (storage and compute) in Table 13.2. For ImageNet, due to compute limitations, we only train ResNet18, ShuffleNetV2, and ViT-Tiny (Dosovitskiy et al., 2021) architectures with both constraints. We mention model resource usage in Table 9.3. Note that our CE baselines are much stronger than the known baselines in literature (Kag et al., 2023a).

Hyper-parameters. For the CIFAR-100 and Tiny-ImageNet datasets, we use the SGD optimizer with a momentum of 0.9 and weight decay of $5e-4$. We train these models up to 200 epochs with cosine learning rate decay with 0.1 as the initial learning rate and batch size of 128. We run the ImageNet experiments for 90 epochs with SGD optimizer with 0.9 as momentum and 0.1 as the learning rate with cosine decay. We use the batch size of 256 for training with a weight decay term of $1e-5$. We provide remaining hyper-parameter details in Appendix J.4.

Table 9.1: Model Statistics: We list the resource requirements (number of parameters and multiply-addition operations) of various models trained on the CIFAR-100 and Tiny-ImageNet datasets.

Architecture	CIFAR-100		Tiny-ImageNet	
	MACs	Params	MACs	Params
Resnet18	555M	11.22M	2221M	11.27M
Resnet50	1298M	23.71M	5191M	23.91M
ShufflenetV2	44.5M	1.4M	177.8M	1.5M

Table 9.2: CIFAR-100 and Tiny-ImageNet: We benchmark DCL and FL against CE and pre-trained baselines with various models. We report Gain as accuracy difference between DCL and CE. It clearly shows that DCL significantly outperforms CE and FL methods. In addition, it reaches accuracy of ImageNet pre-trained baseline without any additional data and requires far less compute.

Architecture	Constraint	Accuracy (%)								
		CIFAR-100					Tiny-ImageNet			
		CE	Pre-trained	FL	DCL	Gain	CE	FL	DCL	Gain
ResNet18	Input-Variability	79.7	82.2	81.3	82.6	**2.9**	65.1	65.6	66.4	**1.3**
	Model-Variability			81.6	82.8	**3.1**		65.2	66.1	**1.0**
	Both			81.7	83.4	**3.7**		65.4	66.7	**1.6**
ResNet50	Input-Variability	83.5	86.9	83.8	85.2	**1.7**	67.2	67.5	68.1	**0.9**
	Model-Variability			83.2	84.9	**1.4**		67.3	67.9	**0.7**
	Both			83.9	86.1	**2.6**		67.6	68.3	**1.1**
ShuffleNetV2	Input-Variability	75.7	80.4	77.5	80	**4.3**	54.5	54.7	55.6	**1.1**
	Model-Variability			76.8	79.5	**3.8**		54.3	54.9	**0.4**
	Both			77.9	80.4	**4.7**		54.7	55.9	**1.4**

9.5.2 Results

Table 13.2 compares the performance of different methods on CIFAR-100 and Tiny-ImageNet datasets. Table 9.3 shows the performance of various baselines on the ImageNet dataset. We report the accuracy of the EMA model in all the methods. Below, we highlight the main takeaways points.

- **DCL and FL achieve better generalization.** We clearly see that enforcing distributional objectives improves DNN generalization compared to baselines. For instance, on CIFAR-100 with ResNet18 architecture, CE achieves 79.7% accuracy while DCL achieves 83.4% accuracy. Similarly, on CIFAR-100 with ShuffleNetV2 architecture, CE achieves 75.7% accuracy, and FL achieves 77.5% accuracy, while DCL significantly outperforms both of these methods and achieves 80% accuracy.

Table 9.3: ImageNet-1K: We train various architectures on the ImageNet dataset and report their resource usage and Top-1 accuracy. It clearly shows that models trained with DCL outperform the CE and FL baselines.

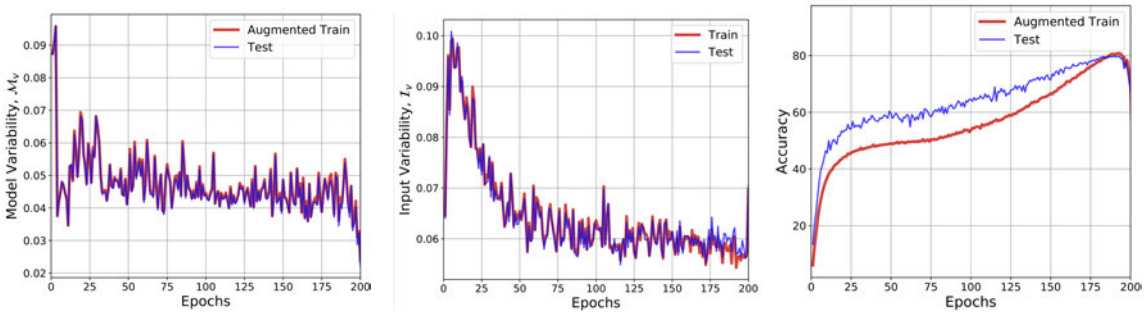
Architecture	Params	MACs	CE	FL	DCL	Gain
ResNet18	11.69M	1.8B	69.58	69.82	71.58	**2.0**
ShuffleNetV2	2.3M	146M	69.36	69.61	70.64	**1.3**
ViT-Tiny	5.72M	1.1B	75.45	76.73	77.29	**1.8**

- **Easily scales to ImageNet dataset.** Table 9.3 shows that DCL scales well to large datasets such as ImageNet. In particular, it achieves better accuracy than the baseline. For instance, with the ResNet18 architecture, the CE method achieves 69.58% accuracy while DCL achieves 71.58% accuracy.

- **DCL trained-from-scratch is competitive with Pre-Trained Models.** Table 13.2 shows the performance of the ResNet50 model pre-trained on ImageNet and fine-tuned on the CIFAR-100 dataset. It takes CIFAR-100 $32 \times 32 \times 3$ image and scales to $224 \times 224 \times 3$ image and runs the inference using this input. In contrast, we trained the ResNet50 model with DCL using only CIFAR-100 data with $32 \times 32 \times 3$ input. It achieves competitive performance as the pre-trained model. This is important because pre-training is expensive. During inference DCL trained model requires 1298M MACs which is much lower than the 4198M MACs required by the ImageNet pre-trained model. In addition, the proposed method requires fewer data to achieve this performance, i.e., only 100K CIFAR-100 images compared to 1.2M ImageNet images. Thus, DCL yields faster inference and requires less sample complexity to achieve competitive performance as ImageNet pre-trained model.

- **Small DCL models outperform large CE models.** DCL enables small models to compete with CE-trained large models. For instance, on CIFAR-100, ResNet18 trained with DCL achieves similar accuracy as the much larger ResNet50 model trained with the cross-entropy method.

9.5.3 Ablations



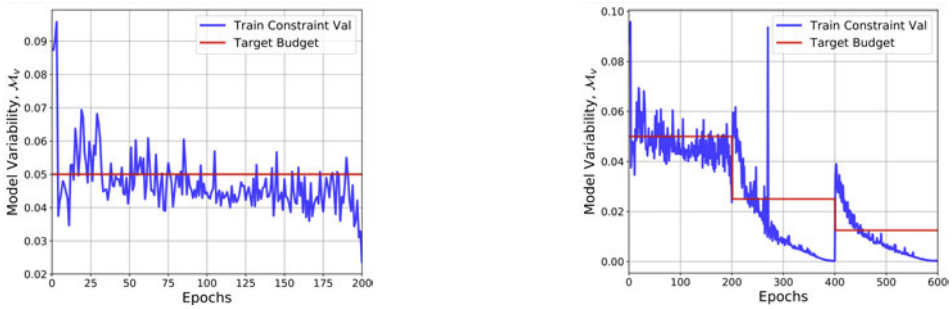
(a) (b) (c)

Figure 9·3: 9·3a: Empirical model variability values for CIFAR-100 and ResNet18 experiment as a function of epochs for the train and test dataset. 9·3b: Empirical input variability values for CIFAR-100 and ResNet18 experiment as a function of epochs for the train and test dataset. 9·3c: Augmented train and test accuracy with epochs. These plots validate our hypothesis that model/input variability and accuracy on augmented data are good surrogates for test-time variability.

- **Distributional Constraints.** Below we analyze the behavior of the constraints during training.

– *Input and Model variability proxies are good generalization surrogates.* In Figure 9·3a, we plot the model-variability metric (Sec. 9.4.1) for the training and test set as a function of the number of training iterations. We note that this metric on train set mimics test datset, validating our assumption that proposed metric is a good generalization proxy. Input-variability plot for CIFAR-100 trained on ResNet18 (Figure 9·3b) leads to a similar observation.

– *Exploration & Exploitation trade-off.* In Figure 9·4a, we plot the empirical model-variability metric along with the target budget value for this constraint while training ResNet18 architecture on CIFAR-100 dataset. It can be clearly seen that the constrained learning promotes constraint violations during the initial phase and towards the end the constraint is satisfied with no violations. This strategy allows exploration-exploitation trade-off during the training.



(a) **(b)**

Figure 9·4: 9·4a: Empirical Model-Variability and Target Budget for CIFAR-100 and ResNet18 experiment are plotted as a function of training iterations. Initially the constraint is violated often, allowing for exploration, and at termination the constraint is satisfied. 9·4b: Change in Empirical Model-Variability and Target budget during various stages as defined in the Algorithm 5 for CIFAR-100 and ResNet18 experiment. It demonstrates that even when the target budget is initialized to a conservative value DCL adapts to a near optimal target budget during multi-stage training.

- **Probing Multi-Stage strategy.** Figure 9·4b shows the trajectory for model-variability constraint and the target budget during different phases as defined by the phase counter M in the Algorithm 5. It clearly shows that even if DCL starts with a conservative budget δ for the constraint, during various M phases it adjusts the target budget adaptively. In this process, the model is exposed to various constraint violations during every stage, enabling a similar exploration-exploitation trade-off as in the first phase.

9.6 Discussion

In this chapter, we explored input and model invariability as constraints during DNN training. To enforce these constraints, we designed a multi-phase budget update strategy coupled with the cosine scheduled exploration. A natural extension of this chapter is a framework where instead of just input and model invariability, we can include other constraints and different methods for enforcing such constraints. For instance, the list of constraints may include entropy of the predictive distribution,

other notions of model invariability such as deep mutual learning, max-margin style constraints, etc.

Part III

Input Hardness Adaptive Models

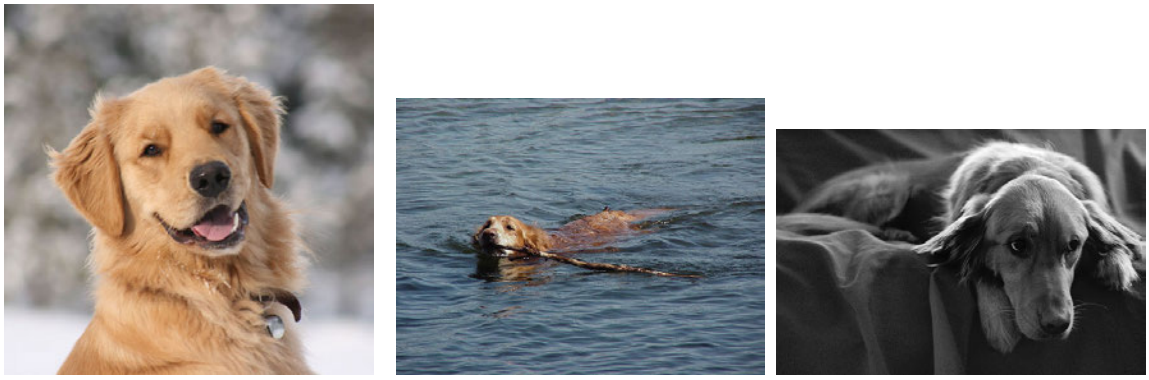
Chapter 10

Input Hardness Adaptive Models: Background

In the first and second parts, we looked at neural architectures that are static in resource allocation (with some exceptions in Chapter 4 and 8). Concretely, these architectures utilize the same amount of resources irrespective of the input characteristics. In a sense, these architectures are not dynamic or adaptive w.r.t. input hardness. The reason why input hardness adaptivity is a desirable property in a DNN is due to the fact that not all input instances are equally hard to predict for a DNN. A model aware of the input hardness would spend considerably fewer resources on easy inputs than difficult ones. In addition, it may even choose to discard prediction on difficult instances due to its prediction confidence.

10.1 Intuition

Let us consider an image classification task with a low-capacity DNN. Since the model has low capacity, it would be beneficial for the DNN to focus learning on the easier part of the feature space. In this example, an image with the object placed in the center under bright lighting and in distribution object would be considered an easy image for classification. In contrast, an image with the object placed in a corner with bad lighting and in the presence of many other objects would be considered difficult input



(a) (b) (c)

Figure 10·1: Easy vs Hard Inputs. We show three images for the 'Golden Retriever' class in the ImageNet-1K dataset. Out of these three images, the first image should be very easily classified by the model as the golden retriever, but the other two might pose some difficulty. In particular, we would desire that the network spent as little time to classify the first image as possible and spend some thoughts on the other images and get the correct prediction.

for the low-capacity model. Even as humans, we exhibit this behavior. We would easily and very quickly classify the first image and take a bit longer to recognize the object in the second image. Thus, in this part, we will focus on integrating this input hardness notion into the DNN architectures and training algorithms with the aim of improving resource utilization.

Concretely, consider Figure 10·1, wherein we draw three images of the golden retrievers from the ImageNet-1K dataset. Image 10·1a should be very easily labelled as the golden retriever class since it highlights key aspects of this dog breed. In contrast, the other two images (Figure 10·1b and 10·1c) are somewhat difficult. Figure 10·1b shows a golden retriever partly submerged underwater, while Figure 10·1c shows the dog under bad lighting and camera conditions. Even as humans, we would be very quick to classify the first image and would spend some time pondering over the other two, before we label them as golden retrievers. Thus, it would be beneficial to expect similar behavior in our neural networks.

10.2 Problems

There are many ways in which input hardness can be integrated into a DNN architecture. Below, we discuss problems relevant to this thesis and cover the related work in the next section.

1. **Selective Classification.** One way to incorporate input hardness in a neural architecture is to enable abstention, i.e., abstain from a prediction on some inputs. This is also referred to as selective classification or learning with a reject option, and such a classifier is referred to as the abstaining classifier. This problem aims to improve DNN performance by abstaining from hard-to-predict examples. Figure 10·2 shows an abstaining classifier that classifies the easy input and abstains on the difficult golden retriever image shown earlier (see Figure 10·1).

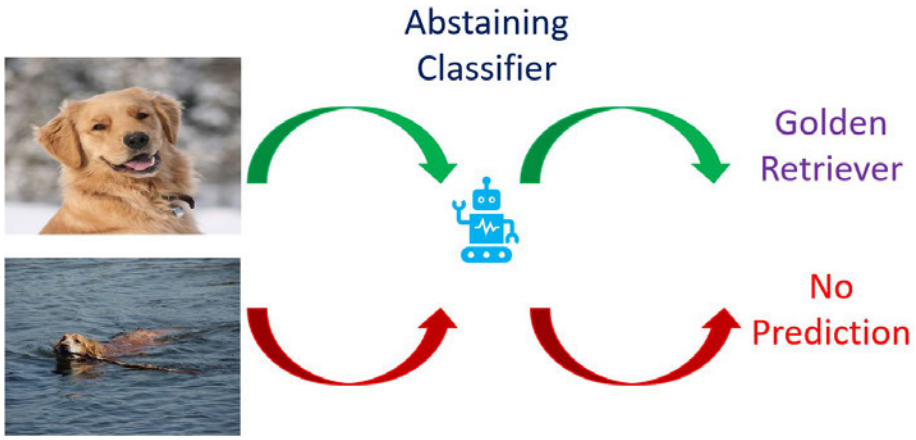
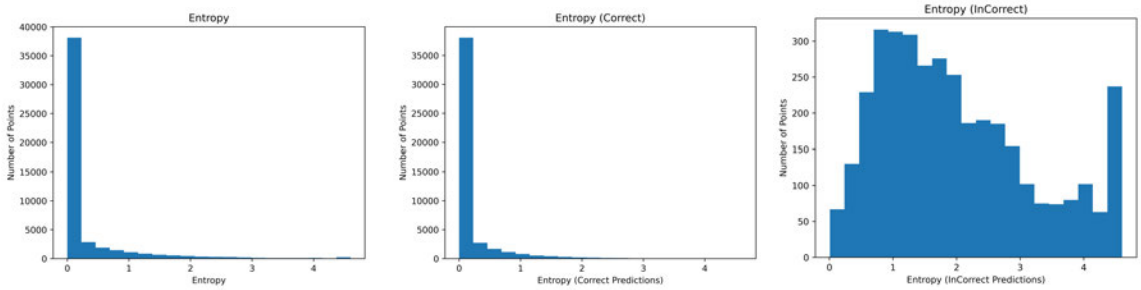


Figure 10·2: Selective Classification.

Consider a DNN learned to classify images between classes from the CIFAR-100 dataset. We show the histogram of the entropy of the predictive distribution in Figure 10·3 along with the predictive entropies corresponding to the correct and incorrect predictions. The higher entropy predictions correspond to low-confidence predictions and there is a non-trivial overlap between high-entropy predictions and incorrect predictions. Thus, it would have been beneficial for the neural network

to allocate as few resources to incorrect low-confidence predictions as possible and focus the model capacity on the remaining feature space.



(a) **(b)** **(c)**

Figure 10·3: Entropy Histogram (ResNet18 model trained on CIFAR-100 data). This figure plots the behavior of the entropy of the predictive distribution on the training data. First figure shows the histogram of the entropy of the all the predictions. Second figure only shows the histogram of the entropy corresponding to the correct predictions, and the last one shows the same for the incorrect predictions.

2. **Routing Abstaining Classifier to an Expert.** For any application, an abstaining classifier would only cover a part of the feature space and learn to abstain prediction on difficult instances. It leaves the question of what to do with the abstained inputs. One strategy is to route these abstained instances to an expert model. This opens the avenue of combining a low-capacity abstaining classifier with a high-capacity expert classifier. Concretely, this raises the issue as to what is the best way to combine an abstaining classifier with an expert model. Figure 10·4 shows an abstaining classifier that sends the difficult input to the expert model and predict only on the easy input.
3. **Integrating Input Hardness During Training.** While the above two problems require architectural changes to incorporate input hardness during inference, we can also learn a DNN with a training algorithm that is aware of the input hardness while keeping the inference static w.r.t. input hardness. For instance, consider a training scheme wherein a network gets additional help on hard inputs, and it gets

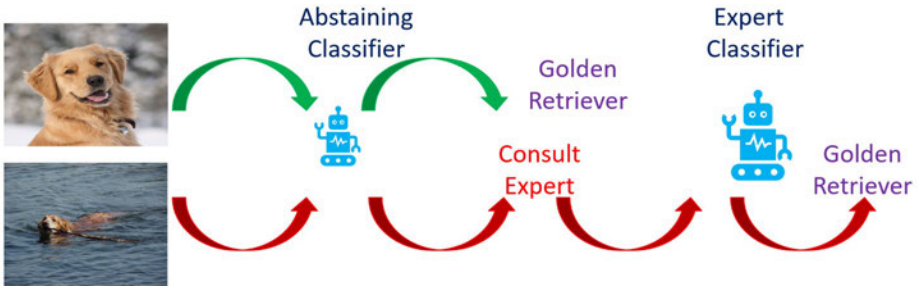


Figure 10·4: Routing Abstaining Classifier to an Expert.

penalized less for mispredicting hard examples compared to easy examples. While input hardness is very much data and model dependent and acquiring this additional supervision is very expensive. With some modifications, it is possible to utilize the popular knowledge-distillation (Hinton et al., 2015) setup to incrementally acquire this supervision. Figure 10·5 shows that during training the student network receives additional help on hard instances and it is not penalized for making mistakes on such examples.

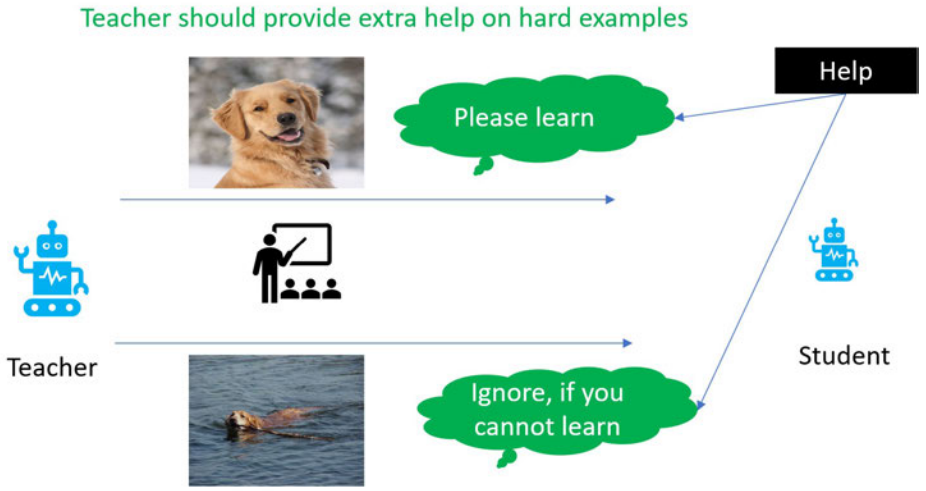


Figure 10·5: Leverageing Input Hardness during Training.

10.3 Related Works

10.3.1 Selective Classification

State of the Art (SOTA) methods: The SOTA, in terms of performance, for SC is encapsulated by three methods. The Naïve method, i.e., rejecting when the output of a soft classifier is non-informative (e.g. classifier margin is too small), and this is surprisingly effective when implemented for modern model classes such as DNNs ((Geifman and El-Yaniv, 2017)). The only other methods that can (marginally) beat this are due to (Liu et al., 2019b), who design a loss function for DNNs, and (Geifman and El-Yaniv, 2019), who design a new architecture for DNNs that incorporates gating.

Both (Liu et al., 2019b) and (Geifman and El-Yaniv, 2019) design methods are based on the **Gating formulation**, mentioned earlier. This formulation was popularised by (Cortes et al., 2016), although similar proposals appeared previously ((El-Yaniv and Wiener, 2010; Wiener and El-Yaniv, 2011)). A number of papers have since extended this approach, e.g. designing training algorithms via alternating minimisation, ((Nan and Saligrama, 2017a; Nan and Saligrama, 2017b)), designing loss functions ((Liu et al., 2019b; Ni et al., 2019; Ramaswamy et al., 2018)), and model classes, such as an architecturally augmented deep neural network (DNN) ((Geifman and El-Yaniv, 2019)). In contrast, our work develops an alternate formulation that directly solves SC without use of specialised losses or model classes.

The naïve method has its roots in the **Direct SC** formulation, which is based on learning a function $f : \mathcal{X} \to \{1, \ldots, K, ?\}$ (where ? denotes rejection), and is pursued by (Herbei and Wegkamp, 2006; Bartlett and Wegkamp, 2008; Wegkamp, 2007; Wegkamp and Yuan, 2011; Yuan and Wegkamp, 2010). The main disadvantage of this formulation is that the methods emerging from it consider very restricted forms of rejection decisions, e.g. $\{|\phi - 1/2| < \delta\}$, where ϕ is a softmax output of a binary

classifier.

An alternate **Confidence Set formulation** has been pursued in the statistics literature by (Lei, 2014; Denis and Hebiri, 2019) (for the binary case), and involves learning sets $\{\mathcal{C}_k\}_{k\in[1:K]}$ such that $\bigcup \mathcal{C}_k = \mathcal{X}$, and each $\mathcal{C}_k$ covers class k in the sense $\mathbb{P}(\mathcal{C}_k|Y = k)$ is large[1]. Points which lie in two or more of the $\mathcal{C}_k$s are rejected, and otherwise points are labelled according to which $\mathcal{C}_k$ they lie in. (Sadinle et al., 2019; Denis and Hebiri, 2017; Chzhen et al., 2019) have subsequently extended this work to the multiclass setting, but they study a 'least ambiguous set-valued classification', which is a different problem from selective classification and does not express it well (see the Appendix of (Gangrade et al., 2021)). A limitation of existing work in this framework is their reliance on estimating the regression function $\eta(x) := \mathbb{P}(Y = k|X = x)$. Proposals typically go via using non-parametric estimates of η, which are then filtered. On a practical level, this reliance on estimation reduces statistical efficiency, and on a principled level, this violates Vapnik's maxim of avoiding solving a more general problem as an intermediate step to solving a given problem ((Vapnik, 2000, §1.9)).

While our formulation is most closely related to the confidence set formulation, and is equivalent to a change of variables of this (§11.2.3), it is directly motivated. Furthermore, our framework naturally leads to relaxations to OSP that let us study discriminative methods on high-dimensional datasets and large model classes, which are unexplored in these works.

In passing, we mention the *uncertainty estimation* (UE), and *budget learning* (BL) problems. UE involves estimating model uncertainty at any point ((Gal and Ghahra-

[1]More accurately, this precise formulation has not appeared for the multiclass setting, and only appears for the binary problem in work by (Lei, 2014; Denis and Hebiri, 2019). Here we are expressing the natural multiclass extension of this, that turns out to be equivalent to selective classification (§11.2.3). The existing literature instead pursues the multiclass extension to LASV classification, as mentioned above.

mani, 2016; Lakshminarayanan et al., 2017)), which can plug into both naïve classifiers, and the other methods. As such, UE is a vast generalisation of SC. BL is a restricted form of SC that aims at reaching the accuracy of a complex model using simple functions, and is relevant for efficient inference constraints.

We highlight a recent *decoupling-based* method for BL by (Acar et al., 2020) that involves the first and last authors. The present work can be seen as considerable extension of this paper to full SC. While the broad strategies of decoupling schemes are similar, significant differences arise since much of the structure developed by (Acar et al., 2020) does not generalise to SC, and development of new forms is necessary. Additionally, our experiments study large multiclass models going beyond best achievable standard accuracy, while (Acar et al., 2020) only study small binary models getting to standard accuracy achievable by larger models.

10.3.2 Dynamic Computation

Split-Computation methods. (Kang et al., 2017) splits the model between the edge and cloud device. More specifically, the initial part of the DNN is stored and executed on the edge device. Then, the edge transmits the intermediate features to the cloud, where the remaining computation of the DNN is performed. This split takes into account the latency budget and the edge transmission cost. More recent variants (Li et al., 2021; Odema et al., 2021) add a branch-out stage that allows for classification on the edge device whenever the decision is deemed correct. A fundamental issue is that requiring copies of the same model on server/edge severely constrains practical use cases. For instance, in Table 12.1 server model is 9X of MCU model. In this case, we could either use the maximum-size edge model and copy it to the server or store a partial copy of the server model locally. The former leads to topping accuracy at 63.5% (blue-line) at high latency, while the latter is plotted as

the red curves in Figure 12·2. While results improve in the dynamic setting, they are not comparable to the other baselines. This fact is not surprising since the amount of pre-processing gained by having a shared model is more than offset by allowing models optimized to the edge. Since the MCUs are too small, only a tiny part of the large DNN can be stored/executed on the edge device. The resulting intermediate feature size is larger than the input size. Thus, cloud accuracy can only be achieved by transmitting all examples to the cloud at a latency of an all-cloud solution.

Dynamic Neural Network methods are surveyed in (Han et al., 2022). These methods budget more compute for hard instances. It includes (a) cascade-based early exit networks (Park et al., 2015; Bolukbasi et al., 2017; Wang et al., 2018), where the constituents are independently designed without feature sharing; and (b) early exit networks (Teerapittayanon et al., 2017; Dai et al., 2020; Li et al., 2019a) where classifiers are introduced at intermediate layers. These works generally disregard the severely resource-constrained setting such as in Table 12.1. Within this framework, we experimented with different approaches and found that (Bolukbasi et al., 2017; Nan and Saligrama, 2017a; Li et al., 2021) serves as a good baseline. These allow for different models optimized for the base and global, but the routing scheme is designed post-hoc based on classifying locally depending on prediction uncertainty. This is depicted as green-line in Figure 12·2.

10.3.3 Training Algorithms

Response Matching Knowledge Distillation. (Zeng and Martinez, 2000; Bucila et al., 2006) distill the response of an ensemble of classifiers into a single neural network by creating a pseudo-labeled dataset using the ensemble of classifiers. (Ba and Caruana, 2014) extend this to the setting with the neural network as the teacher. (Hinton et al., 2015) propose vanilla KD that distilled knowledge from an

ensemble of neural networks into a single network by matching their output logits. This work provided a simple recipe for aligning the teacher and student predictive distributions using the Kullback-Leibler (KL) divergence. Recently, (Beyer et al., 2021) modify the KD procedure to include patient and consistent teacher resulting in substantial gains. Knowledge consistency is enforced by using the same aggressive data augmentation and image views in the student as in the teacher. Patience is promoted using a very long training schedule. This results in a computationally very expensive training process.

(Stanton et al., 2021) analyze response matching KD and suggests that difficulty in optimization leads to poor knowledge distillation. Thus, the teacher and student predictions do not always match, even on the training data. (Cho and Hariharan, 2019) study Vanilla KD through the lens of mismatched student and teacher capacities. It shows that small students are unable to mimic complex teachers. They proposed early stopping teacher training as to remedy to achieve a student-learnable teacher. The above works provide marginal benefits when the gap between student and teacher complexities is large. Specifically, the student cannot learn the complex teacher decision boundaries primarily due to the small student capacity. It becomes imperative to selectively choose only easy-to-learn data points and transfer the teacher knowledge from these points and ignore the hard-to-learn data points during distillation. Thus, our proposal targets the problem of severe capacity gaps between student and teacher models. Additionally, our experiments with standard student and teacher configurations show that our proposed method is still competitive when the capacity difference is small.

Further, (Iliopoulos et al., 2022) proposed a distillation variant in the logit space wherein weights per data point are decided based on a heuristic between teacher and student agreement and the teacher's uncertainty about the prediction. It attempts

to debias the teacher's predictions so that the student learns on the unbiased loss function. This scheme discounts examples wherein the teacher is uncertain and far from being an expert. In contrast, our proposal provides additional help on hard-to-learn data points and learns this mapping iteratively while jointly accounting for the teacher and student feature and logit space. In addition, this weighted distillation does not yield as competitive performance as we achieve through our selective or scaffolded distillation approach.

Feature Matching Knowledge Distillation. In response matching, teacher supervision is limited to its logits. We can enforce intermediate layer feature matching for refined teacher supervision. FitNets (Romero et al., 2015) extend the KD by including the feature matching in the middle layers. (Zagoruyko and Komodakis, 2017) used feature map attention as the teacher supervision. (Tung and Mori, 2019) preserve pairwise similarity in feature maps amongst data points during the distillation. (Chen et al., 2022) modify the student by projecting the student features onto the teacher feature space and by reusing the teacher classifier. While our work focuses primarily on selective distillation in Vanilla KD for simplicity. We can easily extend the proposed framework to incorporate it into feature-matching distillation. We refer the reader to (Gou et al., 2021) for a comprehensive survey on knowledge distillation.

Privileged Information. (Vapnik and Izmailov, 2015) propose the 'learning under privileged information' (LUPI) framework wherein a support vector machine is trained using privileged information unavailable during the inference stage. Later, (Lopez-Paz et al., 2016) unified LUPI and Vanilla KD into generalized distillation, wherein the teacher is learned using the privileged information. Next, the student is trained using the ground truth and the teacher's labels. These works rely on privileged information in the application domain and are shown to work on toy setups. Since our guide function, g is available only during training, it can be thought of as

privileged information from a teacher. We use the teacher's privileged knowledge during training to predict which part of the input space is hard for the student to learn and provide additional help on this space. This strategy enables us to account for the residual margin, i.e., the difference between the teacher and student in the predictive probability space. It also helps to update the student loss landscape such that the student can learn closer to their capacity. By predicting the best margin for the student, the teacher allows the student to reach the local minimum with the best approximation error.

Curriculum Learning & Hard Instance Mining. Curriculum Learning (CL) (Bengio et al., 2009; Hacohen and Weinshall, 2019; Graves et al., 2017) sorts the data based on their hardness as measured by some scoring function (ex., predictive entropy, softmax margin score, etc.). It presents the data points during training in the order of increasing hardness. Similarly, Hard Instance Mining (HIM) (Zhou et al., 2020) reduces the weight of the easy example and increases the weight on hard inputs to promote hard-example learning. We point out that our method is only conceptually related to these works via input hardness. Our method helps the student with hard examples by providing explicitly discounted help g. We learn the helper function g (through teacher representation) that decides whether the student needs help on a given input. Thus, we do not prioritize learning hard examples keeping in mind the fact that student capacity is much smaller than the teacher.

Chapter 11

Selective Classification via One Sided Predictions (OSP)

11.1 Introduction

Selective Classification is a classical problem that goes back to the work of (Chow, 1957; Chow, 1970). The setup allows a learner to classify a query into a class, or to abstain from doing so (we also call this 'rejecting' the query). This abstention models real-world decisions to gather further data/features, or engage experts, all of which may be costly. Such considerations commonly arise in diverse settings, including healthcare[1], security, web search, and the internet of things ((Xu et al., 2014; Zhu et al., 2019)), all of which require very low error rates (lower even than the Bayes risk of standard classification). The challenge of SC is to attain such low errors while keeping coverage (i.e., the probability of not rejecting a point) high. This is a difficult problem because any choice of what points to reject is intimately coupled with the classifiers chosen for the remaining points.

The most common SC method is via 'gating,' in which rejection is explicitly modelled by a binary-valued function γ, and classification is handled by a function π. An instance, x, is predicted as $\pi(x)$ if $\gamma(x) = 1$, and otherwise rejected. Within this

[1]For example, when deciding if a mammary mass is benign or malignant, a general physician may predict based on ultrasound imaging tests, and, in more subtle cases, abstain and refer the patient to a specialist.

formulation, recent work has proposed a number of methods, ranging from alternating minimisation based joint training, to the design of new surrogate losses, and of new model classes to accommodate rejection. Despite this increased complexity, these methods lack power, as shown by the fact that they do not significantly outperform naïve schemes that rely on abstaining on the basis of post-hoc uncertainty estimates for a trained standard classifier. This represents a significant gap in the practical effectiveness of selective classification.

Our Contributions. We describe a new formulation for the SC problem, that comprises of directly learning *disjoint* classification regions $\{\mathcal{S}_k\}_{k\in\mathcal{Y}}$, each of which corresponds to labelling the instance as k respectively. Rejection is *implicitly defined* as the gap, i.e., the set $\mathcal{R} = \mathcal{X} \setminus \bigcup \mathcal{S}_k$. We show that this formulation is equivalent to earlier approaches, thus retaining expressivity.

The principal benefit of our formulation is that it admits a natural relaxation, via dropping the disjointness constraints, into *decoupled* 'one-sided prediction' (OSP) problems. We show that at design error ε, this relaxation has the coverage optimality gap bounded by ε itself, and so the relaxation is statistically efficient in the practically relevant high target accuracy regime.

We pose OSP as a standard constrained learning problem, and due to the decoupling property, they can be approached by standard techniques. We design a method that efficiently adjusts to inter-class heterogeneity by solving a minimax program, controlled by one parameter that limits overall error rates. This yields a powerful SC training method that does not require designing of special losses or model classes, instead allowing use of standard discriminative tools.

To validate these claims, we implement the resulting SC methods on benchmark vision datasets - CIFAR-10, SVHN, and Cats & Dogs. We empirically find that the OSP-based scheme has a consistent advantage over SOTA methods in the regime of

low target error. In particular, we show a clear advantage over the naïve scheme described above, which in our opinion is a significant first milestone in the practice of selective classification.

11.2 Formulation and Methods

Notation. Probabilities are denoted as $\mathbb{P}$, random variables are capitalised letters, while their realisations are lowercase (X and x). Sets are denoted as calligraphic letters, and classes of sets as formal script ($\mathcal{S} \in \mathscr{S}$). Parameters are denoted as greek letters. For a set $\mathcal{S} \subset \mathcal{X}$, $\mathbb{P}(\mathcal{S})$ is shorthand for $\mathbb{P}(X \in \mathcal{S})$.

We adopt the supervised learning setup - data is distributed according to an unknown joint law $\mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$, and we observe n i.i.d. points $(X_i, Y_i) \sim \mathbb{P}$. For K classes, we set $\mathcal{Y} = [1 : K]$, where K is a constant independent of $|\mathcal{X}|$. We use $\mathscr{S}$ to denote the class of sets from which we learn classifiers.

11.2.1 Formulation of SC

We set up the SC problem (Fig. 11·1(top) illustrates binary case) as that of directly recovering disjoint classification regions, $\{\mathcal{S}_k\}_{k\in[1:K]}$ from a class of sets $\mathscr{S}$, under the constraint that the error rate is smaller than a given level ε, which we call the target error. Each such K-tuple of sets induces two events of interest - the rejection event, and the error event.

$$\mathcal{R}_{\{\mathcal{S}_k\}} := \left\{X \in \left(\bigcup \mathcal{S}_k\right)^c\right\}$$
$$\mathcal{E}_{\{\mathcal{S}_k\}} := \bigcup \{X \in \mathcal{S}_k, Y \neq k\}.$$

We will usually suppress the dependence of $\mathcal{R}, \mathcal{E}$ on $\{\mathcal{S}_k\}$. Notice further that $\mathcal{E}$ decomposes naturally into events that depend only on one of the $\mathcal{S}_k$s. We will call

these 'one-sided' error events

$$\mathcal{E}^k_{\mathcal{S}_k} = \{X \in \mathcal{S}_k, Y \neq k\}.$$

With the above notation, we pose the problem as a maximisation program. The value of this is said to be the *coverage at target error level* ε, denoted $C(\varepsilon; \mathscr{S})$.

$$\begin{aligned} C(\varepsilon; \mathscr{S}) = \max_{\{\mathcal{S}_k\}_{k\in[1:K]} \in \mathscr{S}} & \sum_{k=1}^{K} \mathbb{P}(\mathcal{S}_k) \\ \text{s.t.} \quad & \mathbb{P}(\mathcal{E}_{\{\mathcal{S}_k\}}) \leq \varepsilon, \\ & \mathbb{P}(\bigcup_{k,k'\neq k} \mathcal{S}_k \cap \mathcal{S}_{k'}) = 0, \end{aligned} \tag{SC}$$

where the final constraint is expressing the fact that the $\mathcal{S}_k$s must be pairwise disjoint. Note that if ε equals the Bayes risk of standard classification with $\mathscr{S}$, then (SC) recovers the standard solution and coverage 1.

Example. Consider the case of $K = 2$ where $\mathbb{P}_X$ is uniform on $[0, 1]$, $\mathbb{P}(Y = 1|X = x) = x$, and $\mathscr{S}$ consists of single threshold sets $\{x > t\}, \{x \leq t\}$ for $t \in [0, 1]$. The Bayes risk of standard classification is $1/4$. For any $\varepsilon < 1/4$, the coverage at level ε is $C(\varepsilon; \mathscr{S}) = 2\sqrt{\varepsilon}$, which is attained by $\mathcal{S}_1 = \{x > 1 - \sqrt{\varepsilon}\}, \mathcal{S}_2 = \{x \leq \sqrt{\varepsilon}\}$.

Design choices

We outline alternate ways to set up the SC problem that we don't pursue in this paper.

Form of constraints. In (SC), we maximise coverage, while controlling error, which is *error-constrained SC*. Alternately one can pursue the equivalent *coverage constrained SC* problem - minimising $\mathbb{P}(\mathcal{E})$ subject to $\mathbb{P}(\mathcal{R}) \leq \varrho$.

As illustrated in the starting example, our interest in SC is driven by the desire

to attain very small error rates. We thus find the error constrained form of SC more natural, and since we needed to select one of the two for the sake of brevity, we adopt it in the rest of the paper.[2] We note that our method is also effective for coverage-constrained SC, as shown empirically in §11.4.

Error criterion. In (SC), we constrain the raw error $\mathbb{P}(\mathcal{E})$. This has the benefit of being both natural, since it directly controls the standard error metric, and further, simple. Alternate forms of the error metric have been studied in the literature - e.g. (Geifman and El-Yaniv, 2019) condition on acceptance ($P(\mathcal{E}|\mathcal{R}^c)$); (Lei, 2014) separately constrain class conditionals ($P(\mathcal{E}|Y = k) \leq \varepsilon_k$). Most of the development below can be adapted to these settings with minimal changes, and we restrict attention to $\mathbb{P}(\mathcal{E})$ for concreteness.

11.2.2 Relaxation and One-sided Prediction

(SC) couples the $\mathcal{S}_k$s via the $\mathbb{P}$-a.s. disjointness constraint. We now develop a decoupling relaxation.

To begin, note that we may decouple the error constraint by introducing variables that trades off the one-sided error rates as below. This program is equivalent to (SC) in the sense that they have the same optimal value, and the same $\{\mathcal{S}_k\}$ achieve this value.

$$\max_{\{\mathcal{S}_k\}\in\mathscr{S},\{\alpha_k\}\in[0,1]} \sum_{k=1}^{K} \mathbb{P}(\mathcal{S}_k) \tag{SC-expanded}$$
$$\text{s.t.} \quad \forall k : \mathbb{P}(\mathcal{E}^k_{\mathcal{S}_k}) \leq \alpha_k \varepsilon, \quad \sum \alpha_k \leq 1,$$
$$\mathbb{P}(\bigcup_{k,k'\neq k} \mathcal{S}_k \cap \mathcal{S}_{k'}) = 0.$$

Our proposed relaxation is to simply drop the final constraint. The resulting

[2]This is not to imply that the coverage constrained form cannot be more appropriate for some settings. Which one to use in practice is ultimately a problem specific choice.

program may be decoupled, via a search over the variables α_k into K *one-sided prediction* (OSP) problems:

$$L_k(\varepsilon_k; \mathscr{S}) = \max_{\mathcal{S}_k \in \mathscr{S}} \mathbb{P}(\mathcal{S}_k) \text{ s.t. } \mathbb{P}(\mathcal{E}^k_{\mathcal{S}_k}) \leq \varepsilon_k \tag{OSP-k}$$

Notice that the above OSPk problem demands finding the *largest* set $\mathcal{S}_k$ that has a low false alarm probability for the null hypothesis $Y \neq k$. Structurally this is the opposite to the more common Anomaly Detection problem, which demands finding the smallest set with a low missed detection probability.

We note that while we decouple the SC problem completely above, the main benefit is the removal of the intersection constraint, which is the principal difficulty in SC. The sum error constraint is benign, and for reasons of efficiency we will reintroduce it in §11.3.

Continuing, observe that the sets recovered from the above problems may overlap, which introduces an ambiguous region. This overlap region is necessarily of small mass (Prop. 1), and so may be dealt with in any convenient way. Theoretically we break ambiguities in the favour of the smallest label. These sets need not belong to $\mathscr{S}$ anymore, and so this is an (weakly) improper classification scheme.

Overall this gives the following infinite sample scheme:

- For each feasible $\alpha \in [0,1]^K$, solve for $\{L_k(\alpha_k \varepsilon)\}$ for each $k \in [1:K]$. Let $\{\mathcal{T}^\alpha_k\}$ be the recovered sets.
- Let $\mathcal{S}^\alpha_k = \mathcal{T}^\alpha_k \setminus \left(\bigcup_{k'<k} \mathcal{T}^\alpha_{k'}\right)$.
- Return the $\{\mathcal{S}^\alpha_k\}$ that maximises $\sum_k \mathbb{P}(\mathcal{S}^\alpha_k)$ over α.

At small target error levels, which is our intended regime of study, the resulting sets are guaranteed to not be too lossy, as in the following statement. The above is shown (in §H.1.1) by arguing that the mass of the overlap between the OSP solutions (the $\mathcal{T}_k$) is at most 2ε. Empirically this is even lower, see Table 11.4.

Proposition 1. *If $\{\mathcal{S}_k\}$ are the sets recovered by the procedure above, then these are feasible for* (SC)*. Further, their optimality gap is at most 2ε, i.e.*

$$\sum_{k\in[1:K]} \mathbb{P}(\mathcal{S}_k) \geq C(\varepsilon; \mathscr{S}) - 2\varepsilon.$$

11.2.3 Equivalence of SC formulations

We show that the prior gating and confidence frameworks are equivalent to ours, based on transforming feasible solutions of one framework into an other.

Gating: Denote the acceptance set of gating as $\Gamma = \{\gamma = 1\}$, and let the predictions be $\Pi_k = \{\pi = k\}$. Taking $\mathcal{S}_k = \Pi_k \cap \Gamma$ yields disjoint sets that can serve for SC under our formulation that have the same decision regions for each class, and the same rejection region, since $(\bigcup \mathcal{S}_k)^c = \Gamma^c$. Conversely, for disjoint decision sets $\mathcal{S}_k$, the gate $\Gamma = \bigcup \mathcal{S}_k$, and the predictor $\Pi_k = \mathcal{S}_k$ form the corresponding gating solution.

Confidence set: Take confidence sets $\{\mathcal{C}_k\}$ which cover $\mathcal{X}$, and have the rejection set $\mathcal{B} = \bigcup_{k\neq k'} \mathcal{C}_k \cap \mathcal{C}_{k'}$. Then we produce the disjoint sets $\mathcal{S}_k = \mathcal{C}_k \setminus (\bigcup_{k'\neq k} \mathcal{C}_{k'})$, which retain the same decision regions. These also have the same rejection region because we may express $\mathcal{S}_k = \mathcal{C}_k \cap \mathcal{B}^c$, and thus $\bigcap \mathcal{S}_k^c = (\bigcap_k \mathcal{C}_k^c) \cup \mathcal{B}$, and $\bigcap \mathcal{C}_k^c = \varnothing$ since the $\mathcal{C}_k$ cover the space. Conversely, for disjoint $\{\mathcal{S}_k\}$, the sets $\mathcal{C}_k = (\bigcup_{k'\neq k} \mathcal{S}_{k'})^c = \mathcal{S}_k \cup \mathcal{R}$ cover the space, and have the rejection region $\mathcal{R}$ since $\mathcal{C}_k \cap \mathcal{C}_{k'} = \mathcal{R}$ for any pair $k \neq k'$.

Figure 11·1 illustrates these equivalences. Notice that due to the simplicity of the reductions, these equivalences are fine-grained in that the joint complexity of the family of sets used is preserved in going from one to the other. Given these equivalences, we again distinguish our approach from the existing ones.

First, the structure of solutions is markedly different. The gating formulation takes

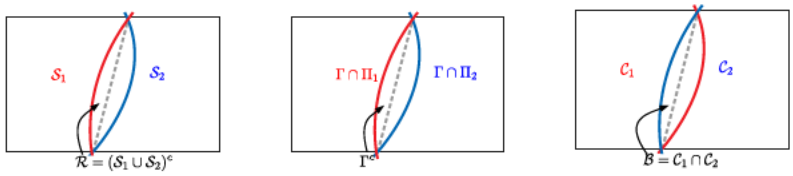


Figure 11·1: An illustration of the equivalence between the three formulations for binary classification. *Top:* our formulation; $\mathcal{S}_i$ denotes disjoint sets; *Bottom Left:* gating with Γ representing gated set; *Bottom Right:* $\mathcal{C}_i$ represents confidence sets, and their intersection representing the rejected set. In each case, the dashed line is the Bayes boundary.

both the rejection and the classification decisions explicitly via the two different sets. The confidence set formulation takes neither explicitly, and instead produces a 'list decoding' type solution. In contrast, we make the classification decisions explicit, and produce the rejection decision implicitly.

Consequently, the salient differences lies in the method. The gating based methods have concentrated on the design of surrogate losses and models, while for the confidence set, methods either go through estimating the regression function, or via a reduction to anomaly detection type problems ((Sadinle et al., 2019; Denis and Hebiri, 2017)). In strong contrast, we develop a new relaxation that allows decoupled learning via 'one-sided prediction' problems. These OSP problems are almost opposite to anomaly detection - instead of finding small sets for each class that do not leave too much of its mass missing, we instead learn large sets that do not admit too much of the complementary class' mass.

11.2.4 Finite Sample Properties of OSP

Thus far we have spoken of the full information setting. This section gives basic generalisation analyses for an empirical risk minimisation (ERM) based finite sample approach. Since the one-sided problems are entirely symmetric, we concentrate only on OSP-1, that is (OSP-k) with $k = 1$, below. Note that the SC problem can directly

be analysed in a similar way (Appx. H.1.2), but we focus on the OSP problem, since this underlies the method we pursue.

We show asymptotic feasibility of solutions, that is, we show that we can, with high probability, recover a set $\mathcal{S}$ for OSP such that $\mathbb{P}(\mathcal{S}) \geq L_1(\varepsilon) - o(1)$ and $\mathbb{P}(\mathcal{E}^1_{\mathcal{S}}) \leq \varepsilon + o(1)$, where the o are as the sample size diverges. This is in contrast to exact feasibility, i.e., insisting on $\mathcal{S}'$ such that $\mathbb{P}(\mathcal{E}^1_{\mathcal{S}'}) \leq \varepsilon$ with high probability. Exactly satisfying constraints via ERM whilst maintaining that the objective is also approaching the optimum is a subtle problem, and was shown to be impossible in certain cases by (Rigollet and Tong, 2011). On the other hand, plug-in methods along with an 'identifiability' condition which imposes that the law of $\eta(X)$ is not varying too fast at any point can be employed to give exact constraint satisfaction along with a small excess risk - the technique was developed by (Tong, 2013), and has been used in SC contexts by, e.g., (Shekhar et al., 2019). However, since the applicability of plug-in methods to large datasets in high dimensions is limited, we do not pursue this avenue here.

One-Sided Learnability

Definition 1. *We say that a class $\mathscr{S}$ is one-sided learnable if for every $\varepsilon \geq 0$ and $(\delta, \sigma, \nu) \in (0,1)^3$, there exists a finite $m(\delta, \sigma, \nu)$ and an algorithm $\mathfrak{A} : (\mathcal{X} \times [1:K])^m \to \mathscr{S}$ such that for any law $\mathbb{P}$, given m i.i.d. samples from $\mathbb{P}$, $\mathfrak{A}$ produces a set $\mathcal{S}_1 \in \mathscr{S}$ such that with probability at least $1-\delta$ over the data,*

$$\mathbb{P}(\mathcal{S}_1) \geq L_1(\varepsilon; \mathscr{S}) - \sigma, \qquad \textit{and} \qquad \mathbb{P}(\mathcal{E}^1_{\mathcal{S}_1}) \leq \varepsilon + \nu.$$

The characterisation we offer is

Proposition 2. *A class $\mathscr{S}$ is one-sided learnable iff it has finite VC dimension. In particular, given n samples, we can obtain a set $\mathcal{S}_1$ that, with probability at least $1-\delta$,*

satisfies

$$\begin{aligned}\mathbb{P}(\mathcal{S}_1) &\geq L_1(\varepsilon;\mathscr{S}) - \sqrt{C_K\frac{(\textsc{vc}(\mathscr{S})\log n + \log(C_K/\delta))}{n}}\\ \mathbb{P}(\mathcal{E}^1_{\mathcal{S}_1}) &\leq \quad \varepsilon \quad + \sqrt{C_K\frac{(\textsc{vc}(\mathscr{S})\log n + \log(C_K/\delta))}{n}},\end{aligned}$$

where C_K is a constant that depends only on the number of classes K.

The proof of the necessity of finite VC dimension is via a reduction to standard learning, while the upper bounds on rates above follow from uniform convergence due finite VC dimension. See §H.1.3. The scheme attaining these is a direct ERM that replaces all $\mathbb{P}$s in (OSP-k) by empirical distributions.

On the whole, applying the above result for each of the K OSP problems tells us that if we can solve the empirical OSP problems for the indicator losses and constraints, then we can recover a SC scheme that, with high probability, incurs error of at most $\varepsilon + O(1/\sqrt{n})$ and has coverage of at least $C(\varepsilon;\mathscr{S}) - 2\varepsilon - O(1/\sqrt{n})$.

11.3 Method

In this section, we derive an efficient scheme, first by replacing indicator losses with two differentiable surrogate variants, and then propose OSP relaxations. A summary of the method expressed as pseudo-code is included in Appx. H.2. Throughout, $\mathscr{S}$ is set to be level sets of the soft output of a deep neural network (DNN), i.e., $\mathscr{S} = \{f(\cdot;\theta) > t\}$, where $f(\cdot;\theta) : \mathcal{X} \to [0,1]$ is a DNN parametrised by θ. The bulk of the exposition concerns learning θs. In this and the following section, $\{(x_i, y_i)\}_{i=1}^n$ refers to a training dataset with n labelled data points.

Relaxed losses. To solve the OSP problem, we follow the standard approach of

replacing indicator losses by differentiable ones. This sets up the relaxed problem

$$\min_{\theta_k} \frac{\sum_i \ell(f(x_i;\theta_k))}{n} \text{ s.t.} \frac{\sum_{i:y_i\neq k} \ell'(f(x_i;\theta_k))}{n_{\neq k}} \leq \varphi_k$$

where θ_k parametrises the DNN, φ_k denote relaxed values of the constraints, and ℓ, ℓ' are surrogate losses that are small for large values of their argument, and $n_{\neq k} = |\{i : y_i \neq k\}|$. In the experiments we use $\ell(z) = -\log(z)$ and $\ell'(z) = -\log(1-z)$, essentially giving a weighted cross entropy loss. We refer to the objective of the above problem as $\widetilde{L}_k(\theta_k)$, and the constraint as $\widetilde{C}_k(\theta_k)$.

A more stable loss. Practically, the loss $\widetilde{L}_k$ suffers from instability due to the fact that the first term sums over all instances. This can seen clearly when $\ell = -\log$, for which the objective includes the sum $\sum_{i:y_i\neq k} -\log(f(x_i;\theta_k))$. Since for negative examples we expect $f(x;\theta_k)$ to be small, this sum is very sensitive to perturbations in these values, which reduces the quality of the solutions. To ameliorate this, we formulate the following 'restricted' loss, where the objective instead sums over only the positively labelled samples

$$\min_{\theta_k} \frac{\sum_{i:y_i=k} \ell(f(x_i;\theta_k))}{n_k} \text{ s.t. } \widetilde{C}_k(\theta_k) \leq \varphi_k. \tag{11.1}$$

Notice that the constraint $\widetilde{C}_k$ is the same as before. We refer to the restricted objective above as $\widetilde{L}_k^{\text{res.}}(\theta_k)$. This loss underlies all further methods, and §11.4.

Note that the above program remains sound w.r.t. the OSP task, since it is a surrogate for the following

$$\max_{\mathcal{S}_k\in\mathscr{S}} \mathbb{P}(X \in \mathcal{S}_k, Y = k) \text{ s.t. } \mathbb{P}(X \in \mathcal{S}_k, Y \neq k) \leq \varepsilon.$$

Comparing (OSP-k) and the above, the constraints are the same, and the objectives differ by $\mathbb{P}(\mathcal{S}_k) - \mathbb{P}(\mathcal{S}_k, Y = k) = \mathbb{P}(\mathcal{S}_k, Y \neq k)$, which, due to the constraint, is at most ε. Thus, the programs are equivalent up to a small gap (that is, optimal solutions for the above attain a value for (OSP-k) that is ε-close to the optimal value for it). For the same reason, we can use the solutions of the above one-sided problem in the scheme of §11.2.2 to yield solutions feasible for (SC) that satisfy an analogue

of Prop. 1 with an optimality gap of 3ε instead of 2ε.

Joint Optimisation and normalisation. A naïve approach with the above relaxations in hand is to optimise the k OSP problems separately. However, this leads to an exponential in K rise in complexity in the model selection process, since different values of $(\varphi_1, \ldots, \varphi_K)$ need to be selected - if Φ such values are searched over for each φ_k, then this amounts to a prohibitive grid search over Φ^K values. In addition, due to class-wise heterogeneity, the values of φ_ks need not be calibrated across programs, and thus simple solutions like pinning all the φ_ks to the same value are not viable. A final issue is that a naïve implementation of this setup results in training K separate DNNs, which leads to a K-fold increase in model complexity.

We make two modifications to handle this situation. First, we normalise function outputs by adopting the following architecture: we consider DNNs with K output nodes, each representing one of the f_k. The backbone layers of the network are shared across all OSP problems. Further, we take

$$f(x) = (f_1(x), \ldots, f_K(x)) = \mathrm{softmax}(\langle w_k, \xi_\theta(x) \rangle),$$

where ξ_θ denotes the backbone's output, and recall that $(\mathrm{softmax}(v))_k = {}^{\exp v_k}/_{\sum \exp v_k}$. This normalisation and restricted model handles both the class-wise heterogeneity, and the blowup in model complexity.

For the sake of succinctness, we define $\mathbf{w} = (w_1, w_2, \ldots, w_K)$, and $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_K)$.

Next, in order to ameliorate the search, we propose jointly optimising the various OSP problems, by enforcing a joint constraint on the sum of the various constraint values via a single value φ. This mimics the structure of (SC), where the constraint limits the sum of the one-sided errors. The relaxation thus amounts to dropping the disjointness constraint, and softening the indicators in (SC). The resulting problem

is

$$\min_{\theta,\mathbf{w},\boldsymbol{\varphi}} \sum_k \widetilde{L}_k^{\text{res.}}(\theta, \mathbf{w}) \tag{11.2}$$
$$\text{s.t.} \quad \forall k : \widetilde{C}_k(\theta, \mathbf{w}) \leq \varphi_k, \quad \sum \varphi_k \leq \varphi,$$

where recall $\widetilde{L}^{\text{res.}}, \widetilde{C}_k$ from above, which are functions of $(\theta, \mathbf{w})$ since the backbone θ is shared, and since all f_k depend on all w_ks due to the softmax normalisation.

Finally, we propose optimising (11.2) via stochastic gradient ascent-descent. We note that one tunable parameter - μ - remains in the problem, corresponding to the sum constraint on the φ_ks, while λ_ks are multipliers for the $\widetilde{C}_k$ constraints. We again denote $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_K)$. The resulting Lagrangian is

$$\widetilde{M}^{\text{res.}}(\theta, \mathbf{w}, \boldsymbol{\varphi}, \boldsymbol{\lambda}, \mu) \tag{11.3}$$
$$= \sum_k \widetilde{L}_k^{\text{res.}}(\theta, \mathbf{w}) + \lambda_k(\widetilde{C}_k(\theta, \mathbf{w}) - \varphi_k) + \mu\varphi_k,$$

and we solve the problem

$$\min_{(\theta,\mathbf{w},\boldsymbol{\varphi})} \max_{\boldsymbol{\lambda}:\forall k, \lambda_k \geq 0} \widetilde{M}^{\text{res.}}(\theta, \mathbf{w}, \boldsymbol{\varphi}, \boldsymbol{\lambda}, \mu), \tag{11.4}$$

treating μ as the single tunable parameter.

We note that the Lagrangian above bears strong resemblance to a one-versus-all (OVA) multiclass classification objective. The principal difference arises from the fact that the losses are weighted by the λ_k terms, and the optimisation trades these off, which are typically not seen in one-versus-all approaches (of course, we also use the resulting functions very differently).

Thresholding and resulting SC solution. The outputs of the classifiers learned with any given μ yield soft signals for the various OSP problems. To harden these

into a decision, we threshold the outputs of the soft classifier at a common level $t \in [0, 1]$. This crucially relies on the earlier normalisation of the soft scores to make them comparable. Finally, to deal with ambiguous regions, we use the soft signals f_k, and assign the label to the one with the largest score. Overall, this leads to the SC solution

$$\begin{aligned} \mathcal{S}_k(\theta, \mathbf{w}, t) =& \{x : f_k(x; \theta, \mathbf{w}) \geq t\} \\ & \cap \ \{x : k = \arg\max_{k'} f_{k'}(x; \theta, \mathbf{w})\}. \end{aligned} \tag{11.5}$$

Model Selection. The above setup has two scalar hyperparameters - μ from (11.4), and threshold t at which hard decisions are produced in (11.5), and each choice of these yields a different solution. Our final model is one that performs the best on the validation dataset among all hyperparameter tuples (μ, t). Concretely, let $\widehat{\mathbb{P}}_V$ denote the empirical law on a validation dataset. Denote the solutions from (11.4) with a choice of μ as $(\theta(\mu), \mathbf{w}(\mu))$. Let $\mathbf{M}, \mathbf{T}$ respectively be discrete sets of μ's and t's. The procedure is

- For each $(\mu, t) \in \mathbf{M} \times \mathbf{T}$, and each k, compute $\mathcal{S}_k(\mu, t) = \mathcal{S}_k(\theta(\mu), \mathbf{w}(\mu), t)$ as defined in (11.5).
- For each $(\mu, t) \in \mathbf{M} \times \mathbf{T}$, evaluate $\widehat{C}_V(\mu, t) = \sum_k \widehat{\mathbb{P}}_V(\mathcal{S}_k(\mu, t))$ and $\widehat{E}_V(\mu, t) = \sum_k \widehat{\mathbb{P}}_V(\mathcal{E}^k_{\mathcal{S}_k})$.
- Let $(\mu^*, t^*) = \arg\max_{\mathbf{M} \times \mathbf{T}} \widehat{C}_V(\mu, t)$ subject to $\widehat{E}_V(\mu, t) \leq \varepsilon$.
- Return $(\theta(\mu^*), \{w_k(\mu^*)\}, t^*)$.

Dataset	Num. of Samples Train.	Test	Val.	Std. Error
CIFAR-10	45K	10K	5K	9.58%
SVHN-10	65.9K	26K	7.3K	3.86%
Cats & Dogs	18K	5K	2K	5.72%

Table 11.1: Dataset sizes and standard classification error

11.4 Experiments

11.4.1 Experimental Setup and Baselines

Datasets and Model Class. We evaluate all methods on three benchmark vision tasks: CIFAR-10 (A.1.4), SVHN-10 (A.1.2) (10 classes), and Cats & DogsA.1.3 (binary). All models implemented below are DNNs with the RESNET-32 architecture ((He et al., 2016)), which is a standard model class in vision tasks. 20% of the training data is reserved for validation in each dataset. All models are implemented in the tensorflow framework. The samples sizes and the best standard classification performance is presented in Table 11.1.

Baselines. We benchmark against three state of the art methods. The 'selective net' and 'deep gamblers' methods also require hyperparameter and threshold tuning as in our setup, and we do this in a brute force way on validation data, as in ours.

Softmax Response Thresholding (SR) involves training a neural network for standard classification, and then thresholding its soft output to decide to reject. More formally, the decision is to reject if $\{\text{softmax}(f_1, \ldots, f_K) < t\}$, where f is the soft output, and t is tuned on validation data. This simple scheme is known to have near-SOTA performance ((Geifman and El-Yaniv, 2017; Geifman and El-Yaniv, 2019)).

Selective Net (SN) is a DNN meta-architecture for SC due to (Geifman and El-Yaniv, 2019). The network provides three soft outputs - (f, γ, π), where f is an auxiliary classifier used to aid featurisation during training, and γ, π is a gate-predictor

pair. Selective net prescribes a loss function that trades off coverage and error via a multiplier c, and by fine-tuning a threshold on γ to reject. We use the publicly available code[3] to implement this, and a comprehensive sweep over the coverage and threshold hyper-parameters. We use 40 valued grid for the parameter c (with 10 equally spaced values in the range $[0.0, 0.65)$ and remaining 30 values in the range $[0.65, 1.0]$). For the gating threshold γ, we use 100 thresholds equally spaced in the range $[0, 1]$, the same as for our scheme.

Deep Gamblers (DG) is a loss function for SC within the gating framework due to (Liu et al., 2019b). The NNs have $K + 1$ outputs - $f_1, \ldots, f_K, f_?$. The cross-entropy loss is modified to $\sum \log\left((f_{y_i}(x_i) + \mathcal{o}^{-1} f_?(x_i)\right)$, where $\mathcal{o} \in [1, K)$ is a hyperparameter that trades-off coverage and accuracy. Hard decisions are obtained by tuning the threshold of $f_?$ on a validation set. We adapt the public torch code[4] for this method to the Tensorflow framework. We used 40 values of $\mathcal{o}$ spaced equally in the range $[1, 2)$[5], and 100 values of thresholds in $[0, 1]$.

11.4.2 Training One-Sided Classifiers

Loss Function. We use the loss function $\widetilde{M}^{\text{res.}}$ developed in §11.3. In particular for $\widetilde{L}_k^{\text{res.}}$, we use $\ell(z) = -\log(z)$, and for $\widetilde{C}_k, \ell'(z) = -\log(1 - z)$.

Training of Backbones. As previously discussed, our models share a common backbone and have a separate output node for each OSC problem. We intialise this backbone with a base network trained using the cross-entropy loss (i.e. a 'warm start'). Note that this typically yields a strong featurisation for the data, and exploiting this

[3] https://github.com/geifmany/selectivenet

[4] https://github.com/Z-T-WANG/NIPS2019DeepGamblers/

[5] We initially made a mistake and scanned $\mathcal{o}$ in $[1, 2)$ instead of $[1, 10)$. We then redid the experiment. with 40 values in $[1, 10)$, and found that performance deteriorated. This is because the optimal $\mathcal{o}$ for these datasets lies in $[1, 2)$, and the wider grid leads to a less refined search in this domain. Thus, values from the original experiment are reported. See Tables H.2, H.3 in §H.3 for the values with a scan over $[1, 10)$.

structure requires us to not move too far away from the same. At the same time, due to the changed objective, it is necessary to at least adapt the final layer significantly. We attain this via a two-timescale procedure: the loss is set to the OSP Lagrangian, and the backbone is trained at a *slower rate* than the last layer. Concretely, the last layer is updated at every epoch, while the backbone is updated every 20 epochs. This stabilises the backbone, while still adapting it to the particular OSP problem that the network is now trying to solve.

Hyper-parameters. All of the methods were trained using the train split and the model selection was performed on the validation set. The results are reported on the separate test data (which is standard for all three of the models considered). The minimax program on the Lagrangian was optimised using a two-timescale stochastic gradient descent-ascent, following the recent literature on nonconvex-concave minimax problems ((Lin et al., 2020b)). In particular, we used Adam optimizer for training with initial learning rates of $(10^{-3}, 10^{-5})$ for the min and the max problems respectively for CIFAR-10 and SVHN-10, and of $(10^{-3}, 10^{-4})$ for Cats & Dogs.[6] These initial rates were reduced by a factor of 10 after 50 epochs, and training was run for 200 epochs. The batch size was set to 128.

We searched over 30 values of μ for each of our experiments - 10 values equally spaced in $[0.01, 1]$, and remaining 20 equally spaced in $[1, 16]$. We further used 100 values of thresholds equally spaced in $[0, 1]$. We have released our implementation at https://github.com/anilkagak2/SelectiveClassification_One_Sided_Prediction.

[6]These rates were selected as follows: the standard classifier was trained with the rate 10^{-4}, which is a typical value in vision tasks. We then picked one value of μ, and trained models using rates in $(10^{-k}, 10^{-j})$ for $(j, k) \in [2:6] \times [2:6]$, tuned thresholds for models at 0.5% target accuracy using validation data, and chose the pair that yielded the best validation coverage. Performance tended to be similar as long as $j \neq k$, and curiously, we found it slightly better to use a smaller rate for the max problem, which goes against the suggestions of (Lin et al., 2020b).

Dataset	Target Error	OSP-based		SR		SN		DG	
		Cov.	Error	Cov.	Error	Cov.	Error	Cov.	Error
CIFAR-10	2%	**80.6**	1.91	75.1	2.09	73.0	2.31	74.2	1.98
	1%	**74.0**	1.02	67.2	1.09	64.5	1.02	66.4	1.01
	0.5%	**64.1**	0.51	59.3	0.53	57.6	0.48	57.8	0.51
SVHN-10	2%	**95.8**	1.99	95.7	2.06	93.5	2.03	94.8	1.99
	1%	**90.1**	1.03	88.4	0.99	86.5	1.04	89.5	1.01
	0.5%	**82.4**	0.51	77.3	0.51	79.2	0.51	81.6	0.49
Cats & Dogs	2%	**90.5**	1.98	88.2	2.03	84.3	1.94	87.4	1.94
	1%	**85.4**	0.98	80.2	0.97	78.0	0.98	81.7	0.98
	0.5%	**78.7**	0.49	73.2	0.49	70.5	0.46	74.5	0.48

Table 11.2: Performance at Low Target Error. The OSP-based scheme is our proposal. SR, SN, DG correspond to softmax-response, selective net, deep gamblers. Errors are rounded to two decimals, and coverage to one.

11.4.3 Results

The key takeaway of our empirical results is the significant increase in performance of our SC scheme when compared to the baselines. We also include some observations about the structure of the solutions obtained.

Performance

Dataset	Target Coverage	OSP-based		SR		SN		DG	
		Cov.	Error	Cov.	Error	Cov.	Error	Cov.	Error
CIFAR-10	100%	100	9.74	99.99	**9.58**	100	11.07	100	10.81
	95%	95.1	**6.98**	95.2	8.74	94.7	8.34	95.1	8.21
	90%	90.0	**4.67**	90.5	6.52	89.6	6.45	90.1	6.14
SVHN-10	100%	100	4.27	99.97	**3.86**	100	4.27	100	4.03
	95%	95.1	**1.83**	95.1	1.86	95.1	2.53	95.0	2.05
	90%	90.1	**1.01**	90.0	1.04	90.1	1.31	90.0	1.06
Cats & Dogs	100%	100	5.93	100	**5.72**	100	7.36	100	6.16
	95%	95.1	**2.97**	95.0	3.46	95.2	5.1	95.1	4.28
	90%	90.0	**1.74**	90.0	2.28	90.2	3.3	90.0	2.50

Table 11.3: Performance at High Target Coverage. Same notation as Table 11.2.

Performance at Low Target Error is presented in Table 11.2, which reports coverage at three (small) targeted values of error - $1/2, 1$, and 2 percent - that are in line with the low target error regime that is the main focus of the paper. Notice that these target error values are far below the best error obtained for standard classification (Table 11.1). We observe that the performance of our SC methods is

Dataset	Target Error	Overlap
CIFAR-10	2%	0.09%
	1%	0.01%
	0.5%	0.00%
SVHN-10	2%	0.05%
	1%	0.01%
	0.5%	0.00%
Cats & Dogs	2%	0.07%
	1%	0.01%
	0.5%	0.00%

Table 11.4: Size of overlap between OSP sets in Table 11.2

significantly higher than the SOTA methods, especially in the case of CIFAR-10 and Cats & Dogs, where we gain over 4% in coverage at the 0.5% design error. The effect is weaker in SVHN, which we suspect is due to saturation of performance in this simpler dataset.

Performance at High Target Coverage is presented in Table 11.3. This refers to the coverage constrained SC formulation discussed in §11.2.1. For these experiments, we use the same μ values (to avoid retraining), but choose thresholds such that the coverage of the resulting model exceeds the stated target, and the models with the lowest error at this threshold are chosen. We observe that at target coverage 100%, the SR solution outperfoms all others. This is expected, since 100% coverage corresponds to standard classification, and the SR objective is tuned to this, while the others are not. Surprisingly, for coverage below 100%, our OSP-based relaxations deliver stronger performance than the benchmarks. Note that this is not due to the low target error performance, because (besides SVHN), the errors attained at these coverage are signficantly above the low target errors investigated in Table 11.2. This shows that our formulation is also effective in the high-coverage regime.

Coverage-Error Curves for the CIFAR-10 dataset are shown in Fig. 11·2. These curves plot the best coverage obtained by training at a given target error level using each of the methods discussed.[7] We find that the coverage obtained by our method

[7]In particular, we train models at target errors $\varepsilon_i = (i/2)\%$ for $i \in [1 : 20]$. We then obtain

uniformly outperform DG and SN, and also outperform SR for the bulk of target errors, except those very close to the best standard error attainable. This illustrates that our scheme is effective across target error levels. We find this rather surprising since we designed our method with explicit focus on the low target error regime. Tables 11.2 and 11.3 can be seen as detailed looks at the left (error < 2) and the upper (coverage > 90) ends of these curves.

Observations regarding baselines. Tables 11.2,11.3, and Figure 11·2 all show that across regimes, DG and SN perform similarly to SR, and are frequently beaten by it. This observation is essentially consistent with the results presented in ((Geifman and El-Yaniv, 2019; Liu et al., 2019b)), and supports our earlier claims that the prior SOTA methods for selective classification do not meaningfully improve on naïve methods. To alleviate concerns about implementation, we emphasise that we performed a comprehensive hyperparameter search for both SN and DG, and the only change is to use RESNETs instead of VGG.

Structure of the Solutions

Overlap of OSP solutions is small. Table 11.4 shows the probability mass of the ambiguous regions for our raw OSP solutions (i.e., the raw sets $\{x : f_k > t\}$ without the max-assignment $\mathcal{S}_k = \{x : f_k > t\} \cap \{x : k = \arg\max f_k\}$) for the models of Table 11.2. We find that this overlap is very small - much smaller than the 2ε bound in Prop. 1. Empirically, these sets are essentially disjoint, and so the training process is close to tight for the SC problem. We believe that this effect is mainly due to the simple tuning enabled by the softmax normalisation of OSP problem outputs described in §11.3.

Consistency of rejection regions. We say that a sequence of models trained

the achieved test error rates $\widehat{\varepsilon}_i$ and coverages c_i for these models. The curves linearly interpolate between $(\widehat{\varepsilon}_i, c_i)$ and $(\widehat{\varepsilon}_{i+1}, c_{i+1})$.

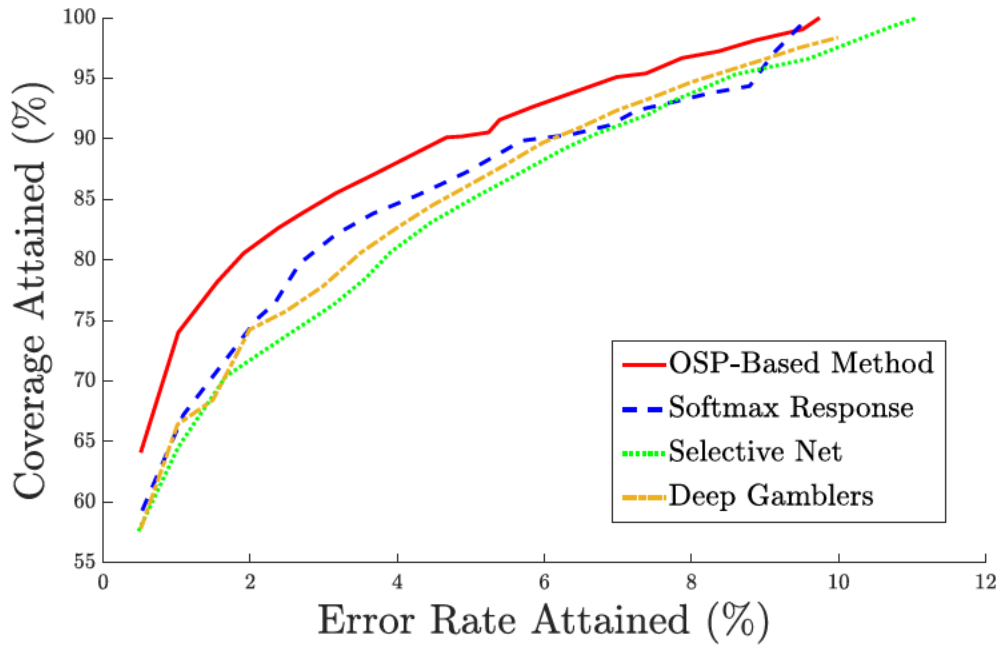


Figure 11·2: Coverage vs Error Curves for the CIFAR-10 dataset. Higher values of coverage are better. Notice the curious behaviour of SR in that the curve's slope sharply changes close to the best standard error rate.

at error levels ε_i have consistent rejection regions if for every $\varepsilon_i < \varepsilon_j$, if $\mathcal{R}_i, \mathcal{R}_j$ are the rejection regions for models trained at these errors, then $\mathbb{P}(\mathcal{R}_i \cap \mathcal{R}_j^c)$ is very small. This means that points that are rejected when designing at a higher error level continue to be rejected for stricter error control. Such consistency may be useful for building cascades of models, or for using the error level at which a point is rejected as a measure of uncertainty.

We found that the models obtained by our procedure are remarkably consistent in the high-accuracy regime. Concretely, for $\varepsilon_i = (i/2)\%$ for $\in [1:5]$, for both CIFAR-10 and Cats & Dogs test sets, the models were entirely consistent, i.e. $\mathcal{R}_j \subset \mathcal{R}_i$ for $j > i$[8], while for SVHN, the only violation was that $|\mathcal{R}_{2.5\%} \cap \mathcal{R}_{2\%}^c| = 2$. Since the test

[8]Due to time constraints we only checked for higher values of i in the CIFAR-10 case, in which the trend continued until $i = 20$, that is, until full coverage. A curious observation in this case was that in all 20 models for the CIFAR-10 dataset, the same value of μ was best, and the models differed

dataset for SVHN has size > 7000, this is a tiny empirical probability of inconsistency of $< 0.03\%$.

11.5 Discussion

One of the drawbacks of the proposed scheme is the mix-n-match of one-sided classifiers with different thresholds after training OSP models for multiple μs. This step is computationally expensive. With some modifications, it should be possible to improve this training cost by only learning one model with one μ and scanning for the same threshold for one-sided classifiers for all classes.

Other works were published on selective classification post our OSP paper. (Huang et al., 2020) proposed self-adaptive training wherein an additional class label represents the abstention signal, and the training proceeds with distillation loss using the exponential moving average (EMA) of the network as the teacher. We can easily apply this strategy to our OSP formulation by incorporating the EMA teacher through knowledge distillation loss functions (such as logit or feature matching). In addition, we can leverage the DiSK (Chapter 13) to replace the distillation loss in such a framework. (Rabanser et al., 2022) captures multiple checkpoints during the training trajectory, and during inference, it computes the agreement between various checkpoints. Their abstention logic is to reject examples with a low agreement between checkpoints. One immediate drawback of such a scheme is the additional storage and compute overhead associated with maintaining many checkpoints from the training trajectory.

only in the thresholds (this did not occur for SVHN and Cats v/s Dogs). While this obviously implies consistency of the rejection regions, it is unexpected, and suggests that there may be room to improve in our training methodology.

Chapter 12

Efficient Edge Inference by Selective Query (Hybrid Models)

12.1 Introduction

We are in the midst of a mobile and wearable technology revolution with users interacting with personal assistants through speech and image interfaces (Alexa, Apple Siri etc.). To ensure an accurate response, the current industrial practice has been to transmit user queries to the cloud-server, where it can be processed by powerful Deep Neural Networks (DNNs). This is beginning to change (see (Kang et al., 2017; Kumar et al., 2020)) with the advent of high-dimensional speech or image inputs. As this interface gains more traction among users, cloud-side processing encumbers higher latencies due to communication and server bottlenecks. Prior works propose a hybrid system whereby the edge and cloud-server share processing to optimize average latency without degrading accuracy.

Proposed Hybrid Learning Method. Our paper focuses on learning aspects of the hybrid system. We propose an end-to-end framework to systematically train hybrid models to optimize average latency under an allowable accuracy-degradation constraint. When a user presents a query, the hybrid learner (see Fig. 12·1) decides whether it can respond to it on-device (eg. "Can you recognize me?") or that the query posed is difficult (eg. "Play a song I would like from 50's"), and needs deeper

Table 12.1: Device & Model Characteristics: Edge (STM32F746 MCU), Cloud (V100 GPU). It takes 2000ms to communicate an ImageNet image from the edge to the cloud (see Appendix §I.1)

Device	Device Characteristics		Model Performance		
	Memory	Storage	Accuracy	Size	Latency
MCU	320KB	1MB	51.1%	0.6MB	200ms
GPU	16GB	1TB	79.9%	9.1MB	25ms

cloud-processing. We emphasize that due to the unpredictable nature (difficulty and timing) of queries coupled with the fact that on-device storage/run-time footprint is relatively small, hard queries inevitably encumber large latencies as they must be transmitted to the cloud.

Fundamental Learning Problem: What queries to cover? While, at a systems level, communications and device hardware are improving, the overall goal of maximizing on-device processing across users (as a way to reduce server/communication loads) in light of unpredictable queries is unlikely to change. It leads to a fundamental learning problem faced by the hybrid learner. Namely, how to train a base, a router, and the cloud model such that, on average, coverage on-device is maximized without sacrificing accuracy. In this context, coverage refers to the fraction of queries inferred by the base model.

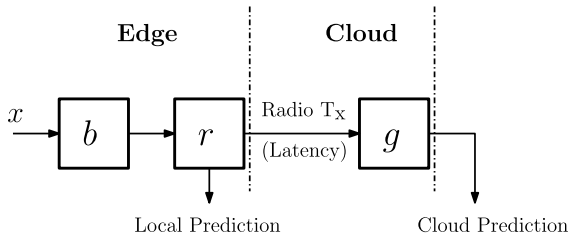


Figure 12·1: Hybrid Model. Cheap base (b) & routing models (r) run on a micro-controller; Expensive global model (g) runs on a cloud. r uses x and features of b to decide if g is evaluated or not.

Novel Proxy Supervision. Training routing models is difficult because we do not a priori know what examples are hard to classify on edge. More importantly, we only benefit from transmitting those hard-to-learn examples that the cloud model correctly predicts. In this context, we encounter three situations, and depending on the operational regime of the hybrid system, different strategies may be required. To expose these issues, consider an instance-label pair (x, y), and the three typical possibilities that arise are: (a) Edge and Cloud are both accurate, $b(x) = g(x) = y$; (b) Edge and Cloud are both inaccurate, $b(x) \neq y$ and $g(x) \neq y$; and (c) Edge is inaccurate but Cloud is accurate $b(x) \neq g(x) = y$.

Our objective is to transmit only those examples satisfying the last condition. *It improves coverage by ceasing data transfers when cloud predictions are bound to be incorrect.* However, the limited capacity of the router (deployed on the MCU) limits how well one can discern which of these cases are true, and generalization to the test dataset is difficult. As such, the routing would benefit from supervision, and we introduce a novel *proxy supervision* (see Sec. 12.2.1) to learn routing models while accounting for the base and global predictions.

Latency. Coverage has a one-to-one correspondence with average latency, and as such our method can be adopted to maximizes accuracy for any level of coverage (latency), and in doing so we characterize the entire frontier of coverage-accuracy trade-off.

Contributions. In summary, we list our contributions below.

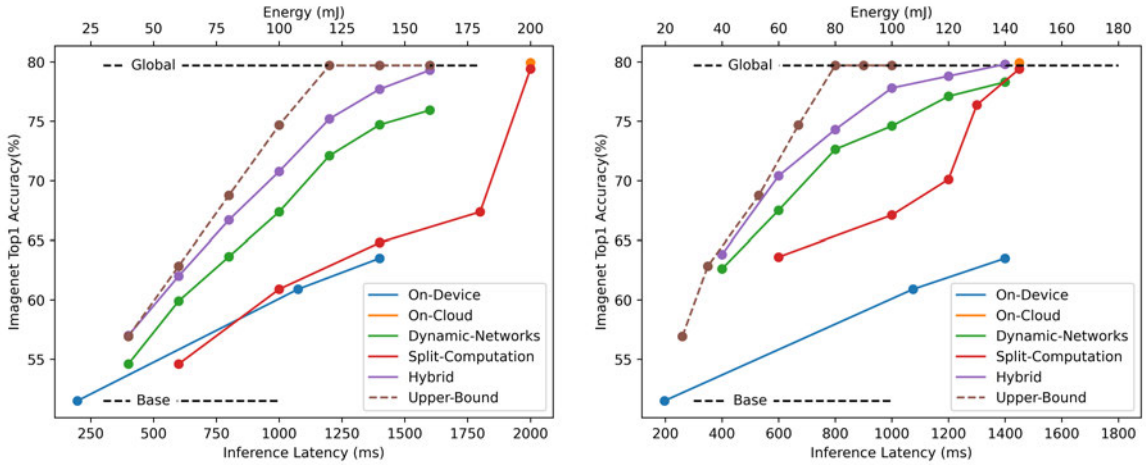
- *Novel End-to-End Objective.* We are the first to propose a novel global objective for hybrid learning that systematically learns all of the components, base, router, global model, and architectures under overall target error or dynamic target latency constraints.
- *Proxy Supervision.* We are the first to provably reduce router learning to binary

classification and exploit it for end-to-end training based on novel proxy supervision of routing models. Our method adapts seamlessly and near optimally across different latency regimes (see knee in Fig. 12·2).

- *Hardware Agnostic.* Our method is hardware agnostic and generalizes to any edge device (ranging from micro-controllers to mobile phones), any server/cloud, and any communication scenarios. Our experiments include (a) MCU and GPU (see Sec. 12.3.1), (b) Mobile Devices and GPUs (see Sec. 12.3.1), (c) on the same device (see Sec. 12.3.2, Appendix I.10.1).
- *SOTA Performance on Benchmark Datasets.* We run extensive experiments on benchmark datasets to show that the hybrid design reduces inference latency as well as energy consumption per inference. Our code is available at `https://github.com/anilkagak2/Hybrid_Models`.

Motivating Example: ImageNet Classification on an MCU. Let us examine a large-scale classification task through the lens of a tiny edge device. This scenario will highlight our proposed on-device coverage maximization problem. We emphasize that the methods proposed in this paper generalize to any hardware and latencies (see Section 12.3). Table 12.1 displays accuracy and processing latencies on a typical edge (MCU) model and a cloud model (see Appendix §I.1). The cloud has GPUs, and its processing speed is 10x relative to MCU processing. In addition, the cloud has a much larger working memory and model storage space compared to the MCU.

Impact of Cloud-Side Bottlenecks. If latency were negligible (either due to communication speed or server occupancy), we would simply transfer every query to the cloud. In a typical system with NB-IoT communication (see Sec. I.1), data transfer to the cloud can take about $2s$—about $100\times$ the processing time on a GPU—and



(a) (b)

Figure 12·2: ImagetNet Classification: Accuracy vs Energy/Latency plot: (a) Constant Communication Latency (2000ms), and (b) Dynamic Communication latency $[200, 2000]$ms. It clearly shows that the proposed hybrid model pareto dominates the baselines while getting significantly closer to the upper-bound in hybrid setup.

Table 12.2: Comparing features of our proposal against baseline. E.-to-E. stands for 'End-to-End', and Arch. for 'Architecture'.

Method	Low Latency	Deploy. on Edge	E.-to-E. Training	Arch. Search	High Accuracy
On-Device	✓	✓	-	✓	✗
On-Cloud	✗	✗	-	✓	✓
Split-Comp.	✗	✗	✓	✗	✗
Dynamic.	✓	✓	✗	✗	✗
Hybrid	✓	✓	✓	✓	✓

thus, to reduce communication, one should maximize MCU utilization for classification. The argument from an energy utilization viewpoint is similar; it costs 20× more for transmitting than processing. In general, communication latency is dynamically changing over time. For instance, it could be as large as 10x the typical rate. In this case, we would want the MCU unit to dynamically adapt its processing so that at fast rates, much of the processing happens on the cloud, and for slow rates, the MCU must still decide what examples to process locally.

12.1.1 Prior Works with Empirical Comparisons on MCU

We use Table 12.1 as a running example to survey prior works, but our inferences here, as seen in Sec. 12.3 hold generally. Our goal is to illustrate key differences through this practical scenario. We refer the reader to Appendix I.1 for more experimental details. We consider two communication contexts: (a) Fixed but large latency (2000ms); (b) Dynamic Latency: A random variable uniformly distributed in the interval $[200, 2000]$ms.

All-Cloud Solution. Here, the edge device transmits all examples to the cloud. It incurs a communication cost in addition to the cloud inference latency. This solution is sub-optimal as seen by the orange circle with 80% accuracy and inference latency of 2025ms in Figure 12·2(a). Our goals are:

(i) To understand the trade-off between latency and accuracy;

(ii) To compare prior methods by the latency level at which cloud-accuracy is achieved.

All-Edge Solution. While we can leverage various methods (Howard et al., 2019; Hinton et al., 2015; Cai et al., 2020; Kag and Saligrama, 2022), MCUnet (Lin et al., 2020a) is the most suitable deployable model for this micro-controller. [1] All-edge baseline is blue curve in Figure 12·2(a). The smallest MCUNet model achieves 51.1% accuracy with 200ms inference latency. In contrast, the largest MCUNet model achieves 63.5% accuracy with 1400ms inference latency; models any larger cannot be deployed due to the MCU hardware limitations listed in Table 12.1. Thus, *cloud-accuracy is unachievable with an all-edge solution for any latency.*

Split-Computation methods. (Kang et al., 2017) splits the model between the edge and cloud device. More specifically, the initial part of the DNN is stored and executed on the edge device. Then, the edge transmits the intermediate features to

[1] Although (Lin et al., 2021) improves upon MCUNet, their pre-trained models are not available publicly. Besides, any improvement in the edge model would yield improvements in all the hybrid baselines.

the cloud, where the remaining computation of the DNN is performed. This split takes into account the latency budget and the edge transmission cost. More recent variants (Li et al., 2021; Odema et al., 2021) add a branch-out stage that allows for classification on the edge device whenever the decision is deemed correct. We employ this method in our comparisons. A fundamental issue is that requiring copies of the same model on server/edge severely constrains practical use cases. For instance, in Table 12.1 server model is 9X of MCU model. In this case, we could either use the maximum-size edge model and copy it to the server or store a partial copy of the server model locally. The former leads to topping accuracy at 63.5% (blue-line) at high latency, while the latter is plotted as the red curves in Figure 12·2. While results improve in the dynamic setting, they are not comparable to the other baselines. This fact is not surprising since the amount of pre-processing gained by having a shared model is more than offset by allowing models optimized to the edge. Since the MCUs are too small, only a tiny part of the large DNN can be stored/executed on the edge device. The resulting intermediate feature size is larger than the input size. Thus, cloud accuracy can only be achieved by transmitting all examples to the cloud at a latency of an all-cloud solution.

Dynamic Neural Network methods are surveyed in (Han et al., 2022). These methods budget more compute for hard instances. It includes (a) cascade-based early exit networks (Park et al., 2015; Bolukbasi et al., 2017; Wang et al., 2018), where the constituents are independently designed without feature sharing; and (b) early exit networks (Teerapittayanon et al., 2017; Dai et al., 2020; Li et al., 2019a) where classifiers are introduced at intermediate layers. These works generally disregard the severely resource-constrained setting such as in Table 12.1. Within this framework, we experimented with different approaches and found that (Bolukbasi et al., 2017; Nan and Saligrama, 2017a; Li et al., 2021) serves as a good baseline. These allow

for different models optimized for the base and global, but the routing scheme is designed post-hoc based on classifying locally depending on prediction uncertainty. This is depicted as green-line in Figure 12·2.

Proposed hybrid method, in contrast to Dynamic Networks, globally optimizes all of the parts (base, router, and cloud) and is the purple line in Figure 12·2. It is evidently close to the upper bound, which is derived in Appendix I.1 based on the practical assumption that the base model is agnostic to what examples are classified by cloud correctly. Prior works including dynamic networks and split-computation methods general miss key details, namely, (a) no explicit accounting number of examples required to be covered locally, (b) no supervision for the router, which makes them myopic, and (c) evaluations only on small datasets (see Appendix §I.6 for details), etc.

Selective Classification. Recently, (Liu et al., 2019b; Gangrade et al., 2021; Geifman and El-Yaniv, 2019) proposed learning with a reject option, wherein a model abstains from predicting uncertain instances. Although we obtain a selective classifier by ignoring the global model (see §12.3.2), we focus on improving the hybrid system, which is a different objective compared to this line of research.

12.2 Method

Notation. Let $\mathcal{X}$ be a feature space and $\mathcal{Y}$ a set of labels. Hybrid design consists of the following:

- A *base model* $b : \mathcal{X} \to \mathcal{Y}$, deployed on an edge device.
- A *global model* $g : \mathcal{X} \to \mathcal{Y}$ deployed on the cloud.
- A *routing model* $r : \mathcal{X} \to \{0, 1\}$ deployed on the edge.

We will treat these models as soft classifiers, outputting $|\mathcal{Y}|$-dimensional scores $\{b_y\}$ and $\{g_y\}$, and two scores r_0 and r_1 for the routing model. The hard output for

the base is the top entry $b(x) = \arg\max_y b_y(x)$, and similarly for g. In this paper, r is realized by the a 2-layer DNN with input $b_y(x)$ - this is intentionally chosen to have minimal implementation complexity. r further admits a scalar tuning parameter t, and assigns x to the global model if $r_1 > r_0 + t$, i.e.

$$r(x; t) = \mathbb{1}\{r_1(x) > t + r_0(x)\}.$$

Varying t (Alg. 12) trades-off accuracy and resource usage, and allows us to avoid retraining r at each resource level by locally tuning a router. By default $t = 0$. The hybrid prediction for an instance x is

$$\hat{y}(x) := (1 - r(x))b(x) + r(x)g(x). \tag{12.1}$$

Evaluation Metrics. *Hybrid accuracy* is the accuracy of the hybrid predictions, defined as

$$\mathcal{A}(r, b, g) = \mathbb{P}(\hat{y}(X) = Y) = \mathbb{P}(r(X) = 0, b(X) = Y) + \mathbb{P}(r(X) = 1, g(X) = Y).$$

This accuracy is fundamentally traded-off with the *coverage* of the hybrid models, which is the fraction of instances that are processed by the cheap base model only, i.e.

$$\mathcal{C}(r, b, g) := \mathbb{P}(r(X) = 0).$$

Modeling Resource Usage. Coverage offers a generic way to model the resource usage of hybrid inference as follows. The resource cost of most models is mainly a function of the architecture. We let α denote a generic architecture and say that $f \in \alpha$ if the function f is realizable by the architecture. Then the resource cost of f is denoted $\mathcal{R}(\alpha)$. Our hybrid design always executes the base and the router, and so the mean resource usage of the hybrid model (r, b, g) with $b \in \alpha_b$ and $g \in \alpha_g$ is

$$\mathcal{R}(r, b, g) := \mathcal{R}_r + \mathcal{R}(\alpha_b) + (1 - \mathcal{C}(r, b, g))\mathcal{R}(\alpha_g), \tag{12.2}$$

where $\mathcal{R}_r$ is a small fixed cost of executing r.

This generic structure can model many resources such as energy usage or total FLOPs used. Our paper is mainly concerned with *inference latency*. To model this, we take $\mathcal{R}(\alpha_b)$ to be the mean time required to execute a base model on the edge, and $\mathcal{R}(\alpha_g)$ to be the *sum of* the mean computational latency of executing g on the cloud and the mean communication latency of sending examples to the cloud. In the previous example of Table 12.1, these numbers would be 200ms and 2025ms respectively.

Overall Formulation. Let $\mathscr{A}_b$ and $\mathscr{A}_g$ be sets of base and global architectures that incorporate implementation restrictions, and ϱ a target resource usage level. Our objective is

$$\max_{\alpha_b \in \mathscr{A}_b, \alpha_g \in \mathscr{A}_g} \max_{r, b \in \alpha_b, g \in \alpha_g} \mathcal{A}(r, b, g) \text{ s.t. } \mathcal{R}(r, b, g) \leq \varrho. \tag{12.3}$$

The outer max over (α_b, α_g) in (12.3) amounts to an architecture search, while the inner max over (r, b, g) with a fixed architecture corresponds to learning a hybrid model.

Below, we describe our method for solving (12.3). Briefly, we propose to decouple the inner and outer optimization problems in (12.3) for efficiency. We train hybrid models by an empirical risk minimisation (ERM) strategy. Furthermore, in Sec. 12.3.3, we perform architecture search using fast proxies for the accuracy attainable by a given pair of architectures without directly training hybrid models.

12.2.1 Learning Hybrid Models

This section trains hybrid models for fixed architectures α_b, α_g, i.e., the inner minimisation problem

$$\max_{r, b \in \alpha_b, g \in \alpha_g} \mathcal{A}(r, b, g) \quad \text{s.t.} \quad \mathcal{R}(r, b, g) \leq \varrho. \tag{12.4}$$

Algorithm 6 Training Hybrid Models

1: **Input:** Training data $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$
2: **Hyper-parameters:** λ_r, Number of epochs E
3: **Initialize:** random r^0, pre-trained b^0, g^0.
4: **for** $e = 1$ **to** E **do**
5: Randomly Shuffle $\mathcal{D}$
6: Generate oracle dataset $\mathcal{D}_{o;(b,g)}$ using Eq. 12.6
7: $r^e = \arg\min_r \mathcal{L}_{\text{routing}}(r, b^{e-1}, g^{e-1})$
8: $g^e = \arg\min_g \mathcal{L}_{\text{global}}(r^e, b^{e-1}, g)$
9: $b^e = \arg\min_b \mathcal{L}_{\text{base}}(r^e, b, g^e)$
10: **Return :** (r^E, b^E, g^E)

Since architectures are fixed in (12.4), the resource constraint amounts to a constraint on the hybrid coverage $\mathcal{C}(r, b, g)$. As is standard, we will approach (12.4) via an ERM over a Lagrangian of relaxed losses. However, several design considerations and issues need to be addressed before such an approach is viable. These include: (a) cyclical non-convexity in (12.4), (b) supervision for the router, and (c) relaxed loss functions. We discuss these below and summarise the overall scheme in Algorithm 7.

Alternating optimisation. Problem (12.4) has a *cyclical non-convexity*. A given r affects the optimal b and g (since these must adapt to the regions assigned by r), and vice-versa. We approach this issue by alternating optimisation. First, we train global and base models with standard methods. Then, we learn a router r under a coverage penalty. The resulting r feeds back into the loss functions of b and g, and these get retrained. This cycle may be repeated.

Modularity of training. The scheme allows modularisation by varying which, and to what extent, steps 7,8,9 in Alg.7 are executed. It helps train a cheap routing model to hybridise a given pair of pre-trained b and g; or in using an off-the-shelf global model that is too expensive to train. Finally, we can learn each component to different degrees, e.g., we may take many more gradient steps on g than r or b in any training cycle.

Learning Routers via Proxy Supervision. Given a base and global pair (b, g), Eqn.(12.4) reduces to

$$\max_r \mathbb{E}[(1 - r(X))\mathbb{1}\{b(X) = Y\} + r(X)\mathbb{1}\{g(X) = Y\}] \qquad \text{s.t.} \qquad \mathbb{E}[r(X)] \leq C_\varrho, \tag{12.5}$$

where C_ϱ is the coverage needed to ensure $\mathcal{R} \leq \varrho$. While a naïve approach is to relax r and pursue ERM, we instead reformulate the problem. Observe that (12.5) demands that

$$r(X) = \begin{cases} 0 & \text{if } b(X) = Y, g(X) \neq Y \\ 1 & \text{if } b(X) \neq Y, g(X) = Y. \end{cases}$$

Further, while $b(X) = g(X)$ is not differentiated, the coverage constraint promotes $r(X) = 0$. Thus, the program can be viewed as a supervised learning problem of fitting the *routing oracle*, i.e.

$$o(x; b, g) = \mathbb{1}\{b(x) \neq g(x) = y\}. \tag{12.6}$$

Indeed, o is the ideal routing without the coverage constraint. For any given (b, g) and dataset $\mathcal{D} = \{(x^i, y^i)\}$, we produce the oracle dataset $\mathcal{D}_{o;(b,g)} := \{(x^i, o(x^i; b, g))\}$. It serves as supervision for the routing model r. This allows us to utilise the standard machine learning tools for practically learning good binary functions, thus gaining over approaches that directly relax the objective of (12.5).

The oracle o does not respect the coverage constraint, and we would need to assign some points from the global to base to satisfy the same. From a learnability perspective, we would like the points flipped to the base to promote regularity in the dataset. However, this goal is ill-specified, and unlikely to be captured well by simple rules such as ordering points by a soft loss of g. We handle this issue indirectly by imposing a coverage penalty while training the routing model and leave it to the

penalised optimisation to discover the appropriate regularity.

Focusing competency and loss functions. To improve accuracy while controlling coverage, we focus the capacity of each of the models on the regions relevant to it - so, b is biased towards being more accurate on the region $r^{-1}(\{0\})$, and similarly g on $r^{-1}(\{1\})$. Similarly, for the routing network r, it is more important to match $o(x)$ on the regions where it is 1, since these regions are not captured accurately by the base and thus need the global capacity. We realise this inductive bias by introducing model-dependent weights in each loss function to emphasise the appropriate regions.

Routing Loss consists of two terms, traded off by a hyperparameter λ_r. The first penalises deviation of coverage from a given target (cov), and the second promotes alignment with o and is biased by the weight $W_r(x) = 1 + 2o(x)$ to preferentially fit $o^{-1}(\{1\})$. Below, ℓ denotes a surrogate loss (such as cross entropy), while $(\cdot)_+$ is the ReLU function, i.e., $(z)_+ = \max(z, 0)$. Sums are over training data of size N. Empirically, we find that $\lambda_r = 1$ yields effective results.

$$\mathcal{L}_{\text{routing}}(r; o) = \lambda_r \left(\text{cov} - \left(1 - \frac{1}{N} \sum_x (r_1(x) - r_0(x))_+\right)\right)_+ + \sum_x W_r(x)\ell(o(x), r(x)). \tag{12.7}$$

Base Loss and *Global Loss* are weighted variants of the standard classification loss, biased by the appropriate weights to emphasise the regions assigned to either model by the routing network - $W_b(x) = 2 - r(x)$ and $W_g(x) = 1 + r(x)$.

$$\mathcal{L}_{\text{base}}(b; r, g) = \sum W_b(x)\ell(y, b(x)); \qquad \mathcal{L}_{\text{global}}(b; r, g) = \sum W_g(x)\ell(y, g(x)).$$

12.3 Experiments

In this section, first, we train hybrid models for resource-constrained MCU devices, thus demonstrating the effectiveness of hybrid training. Next, we show that hybrid

models can be adapted to resource-rich edge devices such as mobile phones. Next, we probe various aspects of our framework through ablations, including (A) validation on other datasets, (B) sensitivity of the solution to small communication latencies, and (C) effectiveness as an abstaining classifier for situations when a global model may be unavailable. Finally, we discuss a simple joint architecture search method for finding hybrid architectures with better performance than off-the-shelf architectures (see Appendix I.2, I.3 for details).

Experimental Setup. We focus on the large-scale ImageNet dataset(Russakovsky et al., 2015), consisting of 1.28M train and 50K validation images. We follow standard data augmentation (mirroring, resize and crop) for training and single crop for testing. We borrow the pre-trained baselines from their public implementations (see Sec. J.3). Appendix I.5.1 lists our hyperparameters settings.

Throughout, we use the largest model in the OFA space (Cai et al., 2020) as our global model and henceforth refer to it as MASS-600M (which needs 600M MACs for evaluation). This model has an accuracy of 80%, and a computational latency of about 25ms on a V100 GPU. There are three main reasons for using this model. Firstly, this model is SOTA at its required computational budget. Secondly, this model is small enough for training to be tractable under our computational limitations (unlike larger models such as EfficientNet-B7, which needs 37B MACs). Finally, since it is the largest model in our architecture search space, it yields a consistent point of comparison across experiments.

To expose gains from the different hybrid components we consider: (a) 'Hybrid-(r)' - training routing models with a pre-trained base and global model, (b) 'Hybrid-(rb)' - training routing and base models with a pre-trained global model, and (c) 'Hybrid-(rbg)' - training all three components. Unless explicitly stated, we create

models by training routing and base, i.e., 'Hybrid-rb' since training the global model is computationally expensive. We train hybrid models to achieve a similar coverage as the oracle. Post-training, we tune the threshold t to generate r with varying coverage (Alg. 12).

Baseline Algorithms. We investigated prior methods (Kang et al., 2017; Nan and Saligrama, 2017a; Bolukbasi et al., 2017; Li et al., 2021; Teerapittayanon et al., 2017), and the entropy thresholding, which we report here, dominates these methods across all datasets. This fact has also been observed in other works (Gangrade et al., 2021; Geifman and El-Yaniv, 2019). It is not surprising since they lack proxy supervision (2nd term in Eq. 12.7), and the routing model does not get instance-level supervision.

Coverage as Proxy. As discussed in Sec. 12.2, for simplicity, we use coverage as the proxy for communication latency, the major contributor to the inference latency. We explicitly model latency in Sec. 12.3.1, where we define the latency of the hybrid system as the sum of three latencies: (a) on-device inference, (b) communication for the examples sent to the cloud, and (c) on-cloud inference.

12.3.1 Hybrid Models for Resource-Limited Edge Devices

In this section, we train hybrid models with off-the-shelf architectures using Algorithm 7. First, we delve into the illustrative experiment in Figure 12·2. Next, we study a similar problem for a resource-rich edge device, specifically mobile phones, and train hybrid models at various base complexities.

Resource Constrained MCU. *Proposed hybrid method with MCUs for ImageNet task is near optimal (close to upper bound); realizes 25% latency/energy reduction while maintaining cloud performance, and at accuracy loss of less than 2% realizes 30% latency/energy reduction.*

Recall from Sec. 12.1 that we deployed an ImageNet classifier on a tiny STM32F746

MCU (320KB SRAM and 1MB Flash), the same setting as MCUNets (Lin et al., 2020a). Using their TFLite model (12.79M MACs, 197ms latency, 51.1% accuracy) as the base, we create a hybrid model by adding the MASS-600 as the global model. We provide additional setup details in the Appendix §I.1.

Figure 12·2 shows the accuracy vs latency and energy. Table 12.3 shows the latency and energy of the hybrid approach against baselines. Deploying hybrid model on an MCU results in following benefits:

- *Pareto Dominance over on-device model.* Hybrid model provides 10% accuracy gains over the best on-device model with similar latency. It achieves the best on-device accuracy with half the latency.
- *Pareto Dominance over other baselines.* Hybrid model achieves 5% higher accuracy than the dynamic networks that use the entropy thresholding baseline (see Figure 12·2). In passing we recall from our earlier discussion that entropy thresholding dominates prior methods in this context.
- *Significant latency reduction to achieve SOTA accuracy.* The hybrid model achieves near SoTA accuracy with 25% reduction in latency and energy consumption.
- *Micro-controller Implementation.* We deployed base and router on the MCU with negligible ($\sim$ 2%) slowdown (see Appendix I.9 for details).

Although our setup in Figure 12·2(a) assumes a constant communication latency of 2000ms, we can easily modify it to incorporate dynamic communication latency shown in Figure 12·2(b). We refer the reader to Appendix I.11 for the exact formulation.

Resource Constrained Mobile Device. *Gains of hybrid method are consistent, near-optimal, and dominate prior methods, across diverse devices, including large footprint devices.*

To show this, we choose the popular MBV3 (Howard et al., 2019) architectures as the base since they have been heavily deployed on mobile devices with resource constraints

Table 12.3: Hybrid models on STM32F746 MCU: Accuracy achieved by different methods at various latency.

	Accuracy (%) v/s Resource Usage					
Latency (ms)	200	600	1000	1400	1600	2000
Energy (mJ)	11	53	96	140	161	216
On-Cloud	-	-	-	-	-	79.9
On-Device	51.1	-	60.9	63.5	-	-
Entropy	-	59.9	67.4	74.7	76.93	-
Hybrid-r	-	61.1	69.5	76.3	78.68	-
Hybrid-rb	-	62.0	70.8	77.7	79.3	-
Hybrid-rbg	-	62.3	71.2	77.9	79.5	-
Upper-Bound	-	62.8	74.7	79.9	79.9	-

such as storage and compute. We create hybrid models using the following base models in the MBV3 family: MBV3-48 (2.54M params, 48M MACs), MBV3-143 (4M params, 143M MACs), and MBV3-215 (5.48M params, 215M MACs). We use MASS-600 as the global model and operate the base model at a fixed coverage level.

Table 12.4: Results for hybrid models with base at various coverages. MASS-600 model achieving $\approx 80\%$ Top1 accuracy is used as global model. Base model belongs to MBV3 space. Upper bounds (Appendix §I.1) are also reported and nearly match hybrid.

Base MACs	Base Acc.(%)	Method	Acc. (%) at Cov.		
			90%	80%	70%
48M	67.6	Entropy	70.7	73.3	74.9
		Hybrid	71.6	74.6	76.8
		Upper Bnd	72.0	75.2	78.9
143M	73.3	Entropy	75.1	76.8	77.6
		Hybrid	75.9	77.8	79.0
		Upper Bnd	75.9	78.2	79.9
215M	75.7	Entropy	77.1	78.3	78.9
		Hybrid	77.6	79.0	79.6
		Upper Bnd	77.6	79.1	79.9

Table 12.4 shows the hybrid accuracy at three coverage levels: 90%, 80% and 70%. Hybrid models operating at a fixed coverage provide the following benefits:

- *Up to 70% latency reduction to achieve SOTA.* Hybrid model with MBV3-215M base achieves 79.59% (near SOTA) with 70% coverage. It communicates with global

Table 12.5: Joint evolutionary search for hybrid models base constraints: 75M, 150M, & 225M. Table shows hybrid and base accuracies at different coverages. Upper bounds are reported in Appendix §I.1. Excess gains represent improved neural-network architecture.

Base MACs	Base Acc.(%)	Method	Acc. (%) at Cov.		
			90%	80%	70%
74M	70.8	Entropy	73.2	75.3	76.8
		Hybrid	74.0	76.4	78.2
		Upper Bnd	73.9	77	79.9
149M	74.5	Entropy	76.1	77.6	78.2
		Hybrid	76.9	78.5	79.4
		Upper Bnd	76.6	78.7	79.9
225M	76.5	Entropy	77.4	78.3	79.1
		Hybrid	78.3	79.6	80.2
		Upper Bnd	77.8	79.4	79.9

model for only 30% inputs.

- *Hybrid models outperform the entropy thresholding.* Hybrid model with MBV3-48M achieves nearly 2% higher accuracy than entropy thresholding baseline.
- *Hybrid model with a smaller base model achieves performance of larger models at various coverage levels.* Using MBV3-48M base, the hybrid model exceeds the accuracy of MBV3-143M at 80% coverage. Similarly, it exceeds the accuracy of MBV3-215M at 70% coverage.

12.3.2 Ablative Experiments.

We refer the reader to Appendix I.10 for the ablation setup. Below we summarize our findings.

(A) **Other Datasets.** We train hybrid models on CIFAR-100 (Krizhevsky and Hinton, 2009) and IMDb (Maas et al., 2011) (reported in Appendix), and observe similar gains.

CIFAR-100. We use the efficient version of the Resnet-32 (He et al., 2016) from (Dong and Yang, 2019) as the global model (35M MACs, 0.5M params, 67.31%

accuracy). The base is a tiny pruned Resnet model (2.59M MACs, 0.02M params, 52.56% accuracy). Table 12.6 shows the performance of the hybrid models and the entropy thresholding baseline at various coverage levels. It shows that hybrid models provide similar benefits on the CIFAR-100 dataset.

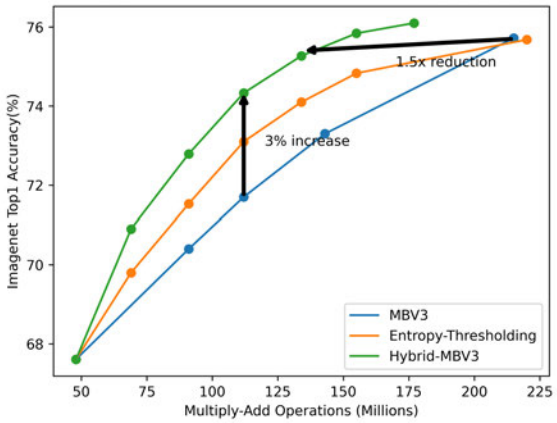


Figure 12·3: Plot for hybrid MACs vs accuracy. Base & Global models on same device.

(B) **Other communication regimes.** We train hybrid models in a setup with no communication delay. It arises when both the base and global models reside on the same hardware. In such a setup, we use a simple evaluation metric for inference latency, i.e., hybrid MACs, i.e., the amount of multiply-add operations required to execute the hybrid model. We pick up an architecture family and create a hybrid model using the smallest and largest architecture. For convenience, we use MobileNetV3 (MBV3) (Howard et al., 2019) family. From MBV3, we pick the smallest model (48M MACs, 67.6% accuracy) as the base and largest model (215M MACs, 75.7% accuracy) as global to create the Hybrid-MBV3 model. Figure 12·3 shows the hybrid model achieves SOTA accuracy at 30% lower latency than entropy thresholding.

(C) **Abstaining classifier.** When a global model is unavailable, the hybrid system can serve as an abstaining classifier on the device, i.e., it rejects a few inputs

and provides predictions on the rest. We create abstaining classifiers with the hybrid setup in Sec. 12.3.1. We show the accuracy of the base model at various coverage levels in Table 12.7. It shows that the abstaining base significantly outperforms the base at full coverage and the abstaining classifier from the entropy thresholding baseline, which demonstrates that our method is competitive against prior works on classification with abstention.

Table 12.6: Hybrid models for CIFAR-100 at various coverages.

Base MACs	Base Acc.(%)	Accuracy(%) at Cov.		
		80%	70%	40%
Entropy	52.56	57.77	59.95	65.1
Hybrid-r	52.56	58.43	61.18	66.9
Hybrid-rb	52.56	59.32	62.48	67.4

Table 12.7: Abstaining Classifier with hybrid models from Sec. 12.3.1. Results for hybrid models with base at various coverage levels.

Base MACs	Base Acc.(%)	Accuracy(%) at Cov.					
		90%		80%		70%	
		Base	Entropy	Base	Entropy	Base	Entropy
48M	67.6	73.3	71.7	78.6	75.6	83.4	80.5
143M	73.3	79.0	77.6	83.9	81.7	88.4	85.4
215M	75.7	81.3	78.6	86.1	82.0	90.1	86.6

(D) **Router validation.** We evaluate the performance of the router against the oracle supervision and show that the router learnt using the Algorithm 7 generalizes well on the test dataset.

12.3.3 Joint Neural Architecture Search for Hybrid Models.

This section proposes a joint neural architecture search (NAS) scheme to resolve the outer max problem of (12.3). NAS methods are strongly dependent on two aspects

Architecture Search Space. Our method is designed for implementation on a *Marked Architecture Search Space* (MASS). It is a set of architectures $\mathscr{A}$ such that

each architecture $\alpha \in \mathscr{A}$ is associated with a known set of canonical parameters θ_α, which are known to be representative of the models under this architecture. That is, for most tasks, fine-tuning θ_α returns a good model with this architecture. Such search spaces have been proposed by, for instance, (Cai et al., 2020) and (Yu et al., 2019). We use (Cai et al., 2020) as search space as it contains a range of architectures of varying capacities.

Proxy Score Function. Given a pair of base and global architectures (α_b, α_g), the search process requires access to the value of the program (12.4). The score function estimates the accuracy of the hybrid model under a given resource constraint. Since training a hybrid model using these architectures would be slow, we use the proposed oracle supervision as a proxy for the router. Thus, the routing oracle $o(\cdot, \alpha_b, \alpha_g)$ in conjunction with the canonical parameters $(\theta_{\alpha_b}, \theta_{\alpha_g})$ serves as an efficient proxy score function for evaluating the architecture pair (α_b, α_g).

Search Algorithm. Given these above two components, NAS reduces to a combinatorial optimization and can be approached by any standard heuristic. Due to its simplicity and prevalence, we use a simple evolutionary algorithm for this (Elsken et al., 2019; Liu et al., 2021) to yield a concrete method. Algorithm 10 summarizes the search scheme (see Appendix I.2 for details).

Empirical Validation. We evaluate the proposed architecture search scheme against the off-the-shelf architectures in Table 12.5 as well as Appendix I.3. It shows that evolutionary search yields higher accuracy hybrid models than off-the-shelf classifiers.

12.4 Discussion

Edge devices offer a range of predictive services to end users. These services require access to cloud resources to guarantee state-of-the-art performance since the edge-

deployable model is very restrictive. Such a cloud interaction has many downsides, including (a) increased inference latency, (b) increased battery usage on edge, (c) additional cost for cloud resources, (d) end-user privacy concerns, and (e) increased carbon footprint due to cloud usage.

The proposed hybrid design aims to address these issues in the existing on-cloud solutions by only selectively querying the cloud on inputs that are hard to classify locally. Our proposal engages the cloud only when it is sufficiently confident that the cloud usage would result in correct classification. As a result, it increases the coverage, i.e., the number of inputs locally predicted by the base model without invoking the cloud model. Thus, the overall cloud utilization is reduced significantly as seen from our empirical evaluations. Note that, in the worst case, when the input is hard to predict locally, the router engages the cloud and gets hit with on-cloud latency.

Our hybrid design, as much as possible, helps the existing predictive services to selectively querying the cloud. As a result, we get the following benefits as compared to the existing on-cloud solutions:

1. Decreased inference latency on the edge device due to communication delay
2. Decreased battery/energy usage on the edge device due to data transmission
3. Reduced cost for access to cloud resources
4. Reduced carbon footprint due to less resource usage
5. End-user privacy concerns are addressed in two ways. First, by selectively querying the cloud, only a small fraction of the data is sent to the cloud. Second, the hybrid design allows the base model to operate as the abstaining classifier, and it enables the practitioners to ask for user consent by showing the base model prediction and router confidence in those predictions. Such a design gives the users informed consent on whether they are satisfied with the inference results or if they would like to send the data to the cloud as their local models cannot correctly classify the inputs.

Chapter 13

Distilling Selective/Scaffolded Knowledge (DiSK)

13.1 Introduction

A fundamental problem in machine learning is to design efficient and compact models with near state-of-the-art (SOTA) performance. Knowledge Distillation (KD) (Zeng and Martinez, 2000; Bucila et al., 2006; Ba and Caruana, 2014; Hinton et al., 2015) is a widely used strategy for solving this problem wherein the knowledge from a large pre-trained teacher model with SOTA performance is distilled onto a small student network.

Vanilla KD. (Hinton et al., 2015) proposed the popular variant of KD by matching the student soft predictions, $\mathbf{s}(\mathbf{x})$ with that of the pre-trained teacher, $\mathbf{t}(\mathbf{x})$ on inputs $\mathbf{x}$. Informally, during student training, an additional loss term, $D_{KL}(\mathbf{t}(\mathbf{x}), \mathbf{s}(\mathbf{x}))$ is introduced that penalizes the difference between student and teacher predictive distributions using Kullback-Leibler (KL) divergence. This promotes inter-class knowledge learned by the teacher. We will henceforth refer to Vanilla KD as KD.

Capacity mismatch between student and teacher. One of the primary issues in Vanilla KD is that the loss function is somewhat blind to the student's capacity to interpolate. In particular, when the student's capacity is significantly lower than the teacher's, we expect the student to follow the teacher only on those inputs realizable

by the student.

We are led to the following question: *What can the teacher provide by way of predictive hints for each input, so that the student can leverage this information to learn to its full capacity?*

Our Proposal: Scaffolding a Student to Distill Knowledge (DiSK). To address this question, we propose that the teacher, during training, not only set a predictive target, $\mathbf{t}(\mathbf{x})$, but also provide hints on hard to learn inputs. Specifically, the teacher utilizes its model to output a guide function, $g(\mathbf{x})$, such that the student can selectively focus only on those examples that it can learn.

- *if* $g(\mathbf{x}) \approx 1$*,* teacher discounts loss incurred by the student on the input $\mathbf{x}$.
- *if* $g(\mathbf{x}) \approx 0$*,* teacher signals the input $\mathbf{x}$ as learnable by student.

With this in mind we modify the KL distance in the KD objective and consider, $D_{KL}(\mathbf{t}(\mathbf{x}), \phi(\mathbf{s}(\mathbf{x}), g(\mathbf{x})))$, where $\phi(\mathbf{s}, g)$, which will be defined later, is such that, $\phi(\mathbf{s}, 0) = \mathbf{s}$ if g offers no scaffolding. We must impose constraints on the guide function g to ensure that only hard-to-learn examples are scaffolded. In the absence of such constraints, the guide can declare all examples to be hard, and the student would no longer learn. We propose to do so by means of a budget constraint $B(\mathbf{s}, g) \leq \delta$ to ensure that the guide can only help on a small fraction of examples.

While more details are described in Sec. 13.3, we note that, in summary, our proposed problem is to take the empirical linear combination of the aforementioned KL distance and a cross-entropy term as the objective, and minimize it under the empirical budget constraint.

We emphasize that $g(\mathbf{x})$ is used only during training. The inference logic for the student remains the same as there is no need for $g(\mathbf{x})$ during inference. The guide function supported student training has three principal benefits. The benefits are explored in Sec. 13.2.

• **Censoring Mechanism.** Our guide function censors examples that are hard to learn for the student. In particular, when there is a large capacity gap, it is obvious that the student cannot fully follow the teacher. For this reason, the teacher must not only *set an expectation* for the student to predict, but also selectively gather examples that the student has the *ability to predict.*

• **Smoothen the Loss landscape.** We also notice in our synthetic experiments that whenever scaffolding is powerful, and can correct student's mistakes, the loss landscape undergoes a dramatic transformation. In particular, we notice fewer local minima in the loss viewed by the guided student.

• **Good Generalization.** The solution to our constrained optimization problem in cases where the guide function is powerful can ensure good student generalization. Specifically, we can bound the statistical error in terms of student complexity and not suffer additional complexity due to the teacher.

Contributions. We summarize our main results.

• We develop a novel approach to KD that exploits teacher representations to adjust the predictive target of the student by scaffolding hard-to-learn points. This novel scaffolding principle has wider applicability across other KD variants, and is of independent interest.

• We design a novel response-matching KD method (Gou et al., 2021) which is particularly relevant in the challenging regime of large student-teacher capacity mismatch. We propose an efficient constrained optimization approach that produces powerful training scaffolds to learn guide functions.

• Using synthetic experiments, we explicitly illustrate the structural benefits of scaffolding. In particular, we show that under our approach, guides learn to censor difficult input points, thus smoothening the student's loss-landscape and often eliminating suboptimal local minima in it.

• Through extensive empirical evaluation, we demonstrate that the proposed DiSK method;

– yields large and consistent accuracy gains over vanilla KD under large student-teacher capacity mismatch (upto 5% and 2% on CIFAR-100 and Tiny-ImageNet).
– produces student models that can get near-teacher accuracy with significantly smaller model complexity (e.g. 8× computation reduction with ∼ 2% accuracy loss on CIFAR-100).
– improves upon KD even under small student-teacher capacity mismatch, and is even competitive with modern feature matching approaches.

13.2 Illustrative Examples

We present two synthetic examples to illustrate the structural phenomena of the censoring mechanism and smoothening of student's loss landscape enabled by the scaffolding approach DiSK, which lead to globally optimal test errors. We defer exact specification of the algorithm to Sec.13.3.

Example 1 (1D Intervals). Consider a toy dataset with one dimensional features $x \in [0, 9]$ and binary class labels $y \in \{\text{Red}, \text{Blue}\}$ as shown in Figure 13·1. There are two Blue labelled clusters as in $[2, 3]$ and in $[5, 7]$. The remaining points are labelled as Red. We sample 1000 i.i.d. data points as the training set and 100 data points as the test set with balanced data from both classes. We describe the details of the experiment setup such as models and learning procedure in Appx. J.1.

Teacher T belongs to the 2-interval function class, and the capacity-constrained student S belongs to the 1-interval function class. Since teacher capacity is sufficient to separate the two classes without error, it learns the correct classifier (see Figure 13·1). In contrast, the best possible student hypothesis cannot correctly separate the two classes. Hence, the student will have to settle onto one of the many local

Table 13.1: The number of times each method lands on various local minima in two toy problems for 100 runs.

Dataset	**1D Intervals**			**2D Gaussians**			
Minima	**A**	**B**	**C (Global)**	**A**	**B**	**C**	**D (Global)**
Accuracy	**67%**	**83%**	**87%**	**70%**	**80%**	**90%**	**100%**
Cross-Entropy	35	64	1	73	12	9	6
KD	30	67	3	1	11	31	57
DiSK	9	1	90	0	0	3	97

minima. We show these minima and the contour plot for the student in Figure 13·1. We present the results of training student models with different initializations in Table 13.1.

KD suffers from bad local minima. KD loss landscape contains many local minima (see Figure 13·1). Due to a big gap between student and teacher capacity, it is unable to help the student discern between these minima. Hence, KD fails to distinguish between the different minima (see Table 13.1).

DiSK censors interval $[2, 3]$ *and in addition focuses training on learnable datapoints.* If we analyze the guide function at the end of the training, we see that it covers (censors) the first Blue cluster. Indeed, both clusters are not simultaneously *l*earnable with the available student capacity. Once we censor the interval $[2, 3]$, then the problem becomes realizable for the student model. The guide function thus captures the excess capacity of the data.

DiSK smoothens loss landscape. The guide function and the budget constraint enable our method to have a smooth loss landscape thanks to the guide-function censoring points, which eliminate the local minima. Hence, DiSK solution lands in the global minimum with high probability.

Example 2 (2D Gaussians). Consider another toy dataset with two dimensional features $\mathbf{x} \in \mathbb{R}^2$ and three class labels $y \in \{\text{Red}, \text{Green}, \text{Blue}\}$. Here we wish to show that DiSK can allow for *globally optimal solutions reaching 100% accuracy,*

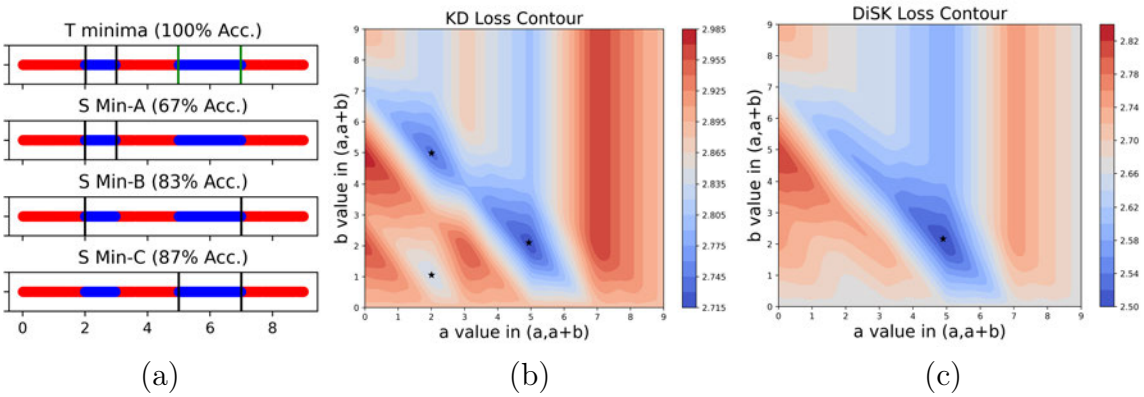


Figure 13·1: (a): 1D Intervals. Data distribution on x-axis $[0, 9]$. Teacher T learns the correct decision boundary with 2-intervals and it is the global minima for this binary classification task. Student S has many bad local minima, and one global minima that best describes the decision boundary with 1-interval. **(b)**: KD training. Loss contour plot shows the various local minima exist. **(c)**: DiSK training. Loss contour plot shows the bad local minima no longer exist.

which appears unachievable with cross-entropy minimization regardless of data size.

Figure 13·2.a shows the labelled data. There are six cluster centers, two with each class label. Data points are drawn using Gaussian balls around the cluster centers with small radii. We sample 1000 i.i.d. data points as the training set and 1000 data points as the test set with equal representation from all three classes. We provide details (hypothesis classes, learning procedure, etc.) in Appx. J.2.

The teacher is a 3-layer neural network with 8, 16, and 3 neurons. The student is a 2-layer neural network with 2 and 3 neurons. We point out that the teacher being an over-parameterized network in this feature space, easily learns the correct decision boundary. While the student being severely constrained network suffers in learning the task. Different training runs lead to different local minima. We show teacher solution and student local minima in Figure 13·2.a. The contour plots for the student models under KD loss and DiSK loss are shown in Figure 13·2.b-13·2.c using (Li et al., 2018a).

The results are similar to the 1D example - KD converges to a poor local minimum

with at least 43% of the initializations, while in contrast, DiSK escapes these by focusing on the learnable part of the input space (Fig. 13·2.c), converging to the global minimum nearly always (Table 13.1).

To conceptualize our findings in these examples, let us attempt to intuitively infer the example-censoring, landscape-smoothening, and good generalization, by utilizing the following conditions that appear to be satisfied for these synthetic examples.

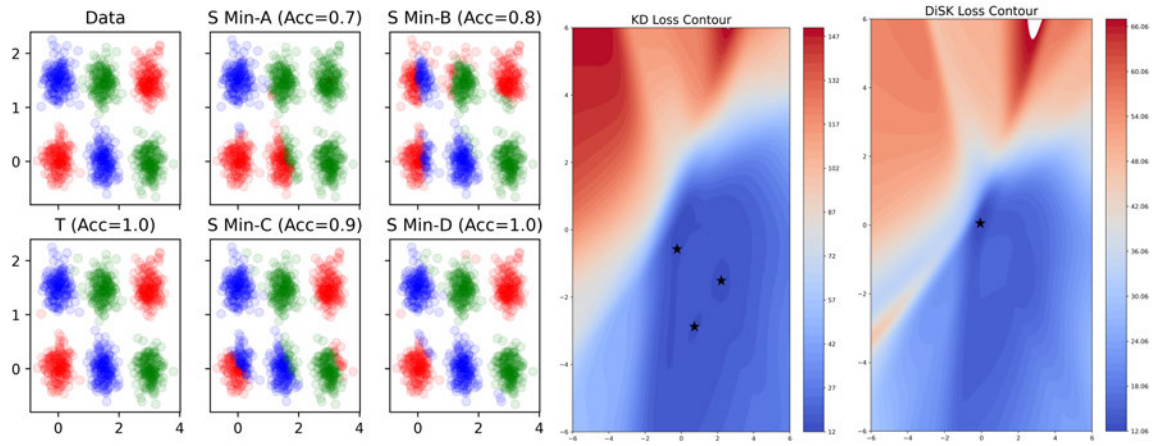
Realizability. Suppose we are in a situation where the guide function $g \in \mathcal{G}$ is sufficiently powerful that there is a student and guide function capable of interpolation, i.e., predictions supported by the guide function, $\phi(\mathcal{S}, g)$, interpolates to mimic the labels.

Example: For instance, consider a binary classification problem with the labels $y \in \{-1, 1\}$. Let $\phi(s, g) = y(s + g)$ with $s(\mathbf{x}) \in [-1, 1]$. Our realizability condition is that we always satisfy $y(s(\mathbf{x}) + g(\mathbf{x})) > 0$. As such, this leads to the condition that if $ys(\mathbf{x}) \leq 0$, then $yg(\mathbf{x}) > 0$. Therefore, $\mathbb{E}[\mathbb{1}_{[ys(\mathbf{x})<0]}] \leq \mathbb{E}[\mathbb{1}_{[ys(\mathbf{x})<0, yg(\mathbf{x})>0]}] \leq \mathbb{E}[\mathbb{1}_{[yg(\mathbf{x})>0]}]$.

Small Guide Function Capacity. In addition to realizability suppose the class of guide functions $g \in G$ has a small capacity (for instance, small VC dimension). For our case this condition is satisfied because our guidance function is obtained by using an MLP on teacher's last layer features.

Example: Continuing with the example above, say we now have m training instances, $(\mathbf{x}_i, y_i)$, $i \in [m]$, $\hat{g}(\mathbf{x})$ is guide function output of DiSK. We can infer by standard statistical learning results (Shalev-Shwartz and Ben-David, 2014) that, for the estimated function $\hat{g} \in G$, it follows with probability greater than $1 - \eta$ that $\mathbb{E}[\mathbb{1}_{[y\hat{g}(\mathbf{x})>0]}] \leq \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{[y_i\hat{g}(\mathbf{x}_i)>0]} + \sqrt{\frac{VC(G)+\log\frac{1}{\eta}}{m}}$. As a result, we can say that if there is a student, $s(\mathbf{x})$ (not necessarily that output by DiSK), which complements $\hat{g}(\mathbf{x})$ and satisfies realizability, then with probability greater than $1 - \eta$:

$$\mathbb{E}[\mathbb{1}_{[ys(\mathbf{x})<0]}] \leq \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{[y_i \hat{g}(\mathbf{x}_i)>0]} + \mathcal{O}\left(\sqrt{\frac{VC(G)+\log\frac{1}{\eta}}{m}}\right).$$



(a) (b) (c)

Figure 13·2: **(a)**: 2D Gaussians. Data distribution on $\mathbb{R}^2$. Teacher T learns the correct decision boundary with 3 layer NN and it is the global minima for this three-way classification task. While student S has many bad local minima, and one global minima that best describes the decision boundary with 2 layer NN. **(b)**: KD training. Loss contour plot shows the various local minima in the loss landscape. **(c)**: DiSK training. Loss contour plot shows the bad local minima no longer exist (wider minima, join two adjust minima, remove bad local minima).

Remarks. The key point is that the student capacity is considerably larger since we typically train an entire DNN, and student complexity-based bound can be vacuous. While the guidance function does bound the student generalization error in terms of guide function complexity, there are strong caveats— we require the strong assumption of realizability on the entire domain, and additionally, while the guide function can witness student error, we are not in a position to precisely estimate it without additional training data. Furthermore, the RHS is a relaxed bound on the student training error. This motivates having a budget constraint to ensure that student learns with small training error.

13.3 Definitions and Formulations

Notation. Let $\mathcal{X}$ and $\mathcal{Y} = \{1, \ldots, C\}$ be the feature and label spaces respectively, focusing on a C-class classification task. We assume that we have a training set of N i.i.d. data points $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We use symbols S and T to denote the student and teacher models respectively. Let $\mathbf{l}_S(\mathbf{x}) \in \mathcal{R}^{|\mathcal{Y}|}$ and $\mathbf{l}_T(\mathbf{x}) \in \mathcal{R}^{|\mathcal{Y}|}$ be the score vector, logits, predicted by S and T on input $\mathbf{x}$. We use τ as the temperature used to soften the probability distribution. We write the resulting softened student and teacher probabilities as $\mathbf{s}^\tau(\mathbf{x})$ and $\mathbf{t}^\tau(\mathbf{x})$, i.e.,

$$\mathbf{s}^\tau(\mathbf{x}) = \text{softmax}\left(\frac{\mathbf{l}_S(\mathbf{x})}{\tau}\right); \quad \mathbf{t}^\tau(\mathbf{x}) = \text{softmax}\left(\frac{\mathbf{l}_T(\mathbf{x})}{\tau}\right)$$

The standard prediction probabilities correspond to $\mathbf{s}^1(\mathbf{x})$ and $\mathbf{t}^1(\mathbf{x})$. We will use $\mathbf{s}_y^\tau(\mathbf{x})$ to denote the yth coordinate in $\mathbf{s}^\tau(\mathbf{x})$, and similarly $\mathbf{t}_y^\tau(\mathbf{x})$. The hard prediction of the student is $p_S(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \mathbf{s}_y^1(\mathbf{x})$, and similarly $p_T(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \mathbf{t}_y^1(\mathbf{x})$ for the teacher.

We use $g(\mathbf{x}) \in [0, 1]$ to denote the helper guide function for the student and teacher pair (S, T). Guide takes input $\mathbf{x}$ and any other feature processed by (S, T) pair and decides whether or not the student needs help on the input $\mathbf{x}$. Finally, we define ReLU activation as $(\cdot)_+ = \max(0, \cdot)$.

13.3.1 Vanilla Knowledge Distillation

KD relaxes the $0-1$ error between the student predictions and the true labels y using the cross-entropy loss $\mathcal{L}_{CE}$. Then, KD denotes the distance between the student and teacher softened probability distributions using the KL divergence. We summarize the corresponding losses as,

$$\mathcal{L}_{CE}(\mathbf{s}) = -\frac{1}{N}\sum_{i=1}^{N} \log \mathbf{s}_{y_i}^1(\mathbf{x}_i); \quad \mathcal{L}_{KL}^\tau(\mathbf{s}) = -\frac{1}{N}\tau^2 \sum_{i=1}^{N}\sum_{y} \mathbf{t}_y^\tau(\mathbf{x}_i) \log \frac{\mathbf{s}_y^\tau(\mathbf{x}_i)}{\mathbf{t}_y^\tau(\mathbf{x}_i)}$$

For hyperparameters $\alpha \in [0,1], \tau > 0$, KD minimizes a mixture of the above losses, as shown below

$$\mathcal{L}_{KD}^{\tau,\alpha}(\mathbf{s}) = \alpha \mathcal{L}_{CE}(\mathbf{s}) + (1-\alpha)\mathcal{L}_{KL}^{\tau}(\mathbf{s}). \tag{13.1}$$

13.3.2 Selective Knowledge Distillation.

KD attempts to transfer the knowledge from the teacher to the student on all training data points, which is a sub-optimal objective when there is a capacity mismatch between the student and the teacher. Instead, we propose distilling selective knowledge (DiSK) to allow the student to selectively ignore some hard-to-learn data points during training, transferring the teacher's knowledge only on easy-to-learn inputs, and matching the learning to student capacity. Our objective is to minimize

$$\min_{\mathbf{s},g,\delta} \underbrace{\frac{1}{N}\sum_{i=1}^{N} \text{distance}(\mathbf{t}(\mathbf{x}_i); \phi(\mathbf{s}, g)(\mathbf{x}_i))}_{\text{Distance between } T \text{ and } S \text{ with help of } g} \quad \text{subject to.} \quad \underbrace{\frac{1}{N}\sum_{i=1}^{N} g(\mathbf{x}_i)\mathbb{1}_{\{y_i \neq \arg\max_y \mathbf{s}_y(\mathbf{x}_i)\}} \leq \delta}_{\text{Support budget constraint on } g} \tag{13.2}$$

where ϕ interpolates student predictions based on the guide's help. The divergence term helps in minimizing the distributional distance between the teacher and student probabilities after the guide g is included. While the budget term in the optimization constrains the helper g to provide help only when necessary, the amount of help given to the student should be within the budget $\delta \in [0,1]$.

Function g Construction. As previously stated, we use the teacher's last layer features and soft predictions as input to the guide g. The guide is structured as a light-weight three-layer neural network with these inputs, with a sigmoid activation at the last layer. We re-emphasise that g is not used at inference time, and only aids training. More details are left to Appx.J.3.

Relaxed Losses, Lagrangian & Optimization Algorithm. We relax Eq.

13.2 and construct a Lagrangian by integrating the constraint into the minimization.

Budget constraint relaxation. We relax the indicator loss in the budget to a cross-entropy, and treat δ as a hyperparameter to get

$$\mathcal{L}^{\delta}_{budget}(\mathbf{s}, g) = \left[-\frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}_i) \log s^1_{y_i}(\mathbf{x}_i) - \delta \right]_+ \tag{13.3}$$

We view the scaffold as a way for the student to interpolate the uncensored data. It suggests that a good initialization for the budget is the error of cross-entropy trained model when the student does not have the teacher supervision. Thus, we scan the budget in a small interval around this initialization.

Distillation objective. Motivated from KL loss, we construct a distance loss with guide function as,

$$\mathcal{L}^{\tau,K}_{dist}(\mathbf{s}, g) = -\frac{1}{N} \tau \tau_{\mathbf{t},\mathbf{s},\mathcal{D}} \sum_{i=1}^{N} \sum_{y} t^{\tau}_{y}(\mathbf{x}_i) \log \left(s^{\tau_{\mathbf{t},\mathbf{s},\mathcal{D}}}_{y}(\mathbf{x}_i) + 1_{y \in \text{top}_K(t^{\tau}(\mathbf{x}_i))} g(\mathbf{x}_i) \right) \tag{13.4}$$

We point out two modifications in the distillation loss. First, $\mathcal{L}_{dist}$ explicitly adds guide value to softened student probabilities in selected class indices. The class indices guide function adds value are picked as top K classes based on the teacher probabilities for any input $\mathbf{x}_i$ where K is a hyperparameter of our method. The rest of the class indices do not get any value from g. Second, we use different temperature parameters for teacher and student. Temperature parameter for teacher, τ, is a hyperparameter. The student temperature is found by minimizing the KL loss between teacher softened probabilities and the student softened probabilities over the training dataset, .i.e $\tau_{\mathbf{t},\mathbf{s},\mathcal{D}} = \arg\min_{\tau'} \sum_i KL(\mathbf{t}^{\tau}(\mathbf{x}_i), \mathbf{s}^{\tau'}(\mathbf{x}_i))$.

Similar to KD, we incorporate standard cross entropy loss between student model predictions and the ground truth labels for stability. We construct our Lagrangian

Algorithm 7 DiSK: Distilling Selective Knowledge.

Input: Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, Teacher $\mathbf{t}$,
Parameters: τ, K, α, $\lambda_{\min}$, $\lambda_{\max}$, Number of iterations R, λ_T cosine period, Budget δ,
Initialize: $\mathbf{s}$, randomly initialize g, $\lambda = \lambda_{\min}$,
for $r = 1$ **to** R **do**
  Randomly Shuffle Dataset $\mathcal{D}$
  $g \leftarrow \arg\min_g \mathcal{L}_{DiSK}^{\tau,K,\delta,\alpha}(\mathbf{s}, g, \lambda)$
  $\mathbf{s} \leftarrow \arg\min_{\mathbf{s}} \alpha\mathcal{L}_{CE}(\mathbf{s}) + (1-\alpha)\mathcal{L}_{dist}^{\tau,K}(\mathbf{s}, g)$
  $\lambda \leftarrow \lambda_{\min} + (\lambda_{\max} - \lambda_{\min}) \times \frac{(1-\cos\frac{r \mod \lambda_T}{\lambda_T}\pi)}{2}$
Return : $\mathbf{s}$

by combining Eq. 13.3 and 13.4 as,

$$\mathcal{L}_{DiSK}^{\tau,K,\delta,\alpha}(\mathbf{s}, g, \lambda) = \alpha\mathcal{L}_{CE}(\mathbf{s}) + (1-\alpha)\mathcal{L}_{dist}^{\tau,K}(\mathbf{s}, g) + \lambda\mathcal{L}_{budget}^{\delta}(\mathbf{s}, g) \tag{13.5}$$

where α is a hyper-parameter and λ is the dual parameter of DiSK.

We optimize Obj. 13.5 using a primal dual update scheme as explained in Algorithm 7.

Primal Parameter Updates $(\mathbf{s}, g)$*.* We learn the student $\mathbf{s}$ and the guide function g using alternating minimization. We approximate $\arg\min$ with running SGD for a small number of epochs on the full dataset. In each iteration, we first learn the guide function g to select the data partition from which the knowledge needs to be distilled. Next, given the function g, we learn the student using the help g. We empirically found that not optimizing the student model on budget loss gives more stable results. Hence, we minimize the student model only on the distillation and cross-entropy losses.

Dual Parameter Update (λ) *Intuition.* Although it is tempting to optimize the above via a dual ascent and primal descent scheme (wherein the dual parameter λ is increased by residual term in the constraint until constraint satisfaction), recent

Table 13.2: Model Statistics. We compute the storage (number of parameters) and computational requirements (number of multiply-addition operations) of the models used in this work.

Architecture		CIFAR-100		Tiny-ImageNet		Architecture	CIFAR-100	
		MACs	Params	MACs	Params		MACs	Params
Teacher	ResNet10-ℓ	64M	1.25M	255M	1.28M			
	ResNet10	253M	4.92M	1013M	5M	ResNet32x4	1083M	7.4M
	ResNet18	555M	11.22M	2221M	11.27M	Wide-ResNet-40-2	327M	2.25M
	ResNet34	1159M	21.32M	4637M	21.38M			
Student	ResNet10-xxs	2M	13K	8M	15K	ResNet8x4	177M	1.2M
	ResNet10-xs	3M	28K	12M	31K	ShuffleNetV2	44.5M	1.4M
	ResNet10-s	4M	84K	16M	90K	Wide-ResNet-16-2	101M	700K
	ResNet10-m	16M	320K	64M	333K	Wide-ResNet-40-1	83M	570K
						MobileNetV2x2	22M	2.4M

work, (Sun and Sun, 2021), has proposed to decrease the λ in the non-convex regime. Inspired by this, we update the dual parameter by a fixed schedule between $[\lambda_{\min}, \lambda_{\max}]$. $\lambda_{\min}$ encourages exploration and allows student model to distill knowledge from all points. On the other hand, $\lambda_{\max}$ enforces the constraint and forces the student model to learn on uncensored inputs. We choose $R \approx 4\lambda_T$, so that the algorithm is exposed to a few exploratory periods. For the final period, we increase λ monotonically so that budget is more strictly enforced at termination.

Computational Efficiency. Algorithm 7 trains both student and guide networks. The guide network being small (three-layer MLP) relative to the student (CNN model), the additional cost in training the guide is relatively insignificant, and as such DiSK efficiency is similar to KD.

13.4 Experiments

We evaluate DiSK in various capacity mismatch scenarios on benchmark datasets. Our code is available at `https://github.com/anilkagak2/DiSK_Distilling_Scaffolded_Knowledge`

Datasets. We use publicly available CIFAR-100 (Krizhevsky and Hinton, 2009), Tiny-ImageNet (Le and Yang, 2015) datasets. CIFAR-100 contains 50K training and 10K test images from 100 classes with size $32 \times 32 \times 3$. While Tiny-ImageNet contains

100K training and 10K test images from 200 classes with size $64 \times 64 \times 3$. We provide the dataset setup and data augmentations used in detail in Appx. A.1.

Models. We evaluate standard convolutional models on these datasets including ResNet(He et al., 2016), Wide-ResNet(Zagoruyko and Komodakis, 2016), MobileNet(Sandler et al., 2018), and ShuffleNet(Ma et al., 2018). Table 13.2 shows the storage and computational requirements of all the models used in this work. We provide explicit model configurations in Appx. J.3, including the tiny models we generate from the ResNet architectures.

Methods. We study performance against standard cross-entropy (CE) based learning and the vanilla KD methods. For each method, we train models for 200 epochs using SGD as the optimizer with 0.9 momentum and 0.1 learning rate. See Appx. J.4 for more training details. We have recorded the mean in our results as the variance of 3 trials in our experiments is not larger than 0.1 in most cases.

We perform evaluations in different settings. Below, we explain individual setups.

Large Capacity Mismatch Setting. We distill knowledge from a teacher model into a student model where the student has much less capacity compared to the teacher model. We use four large capacity ResNet teachers and five tiny ResNet students and train these students using CE, KD, and DiSK methods. Performances, and the gains of DiSK are reported in Table 13.3.

Small Capacity Mismatch Setting. While DiSK has been designed for the scenario when student capacity is very low, we further evaluate it in the setting where teacher and student capacities are similar, to probe how far the power of the method extends. The model classes used are the standard choice for this scenario (Chen et al., 2022; Tung and Mori, 2019). Performance is reported in Table 13.4.

Table 13.4 further reports the results of the feature matching distillation methods: FitNets (Romero et al., 2015), SemCKD (Chen et al., 2021a), and SimKD (Chen et al.,

2022). Such methods can often outperform response matching KD on large students, due to student representations that are more aligned with the teacher, but typically at an increased training cost. While feature matching methods are not the main focus of our work (and in principle scaffolding idea can be extended to them), we observe that DiSK often improves upon their performance without any direct feature matching.

Experiment results. Below, we highlight salient features of DiSK based on empirical data.

DiSK outperforms the baselines uniformly across all datasets and student sizes. As shown by Table 13.3, DiSK significantly improves the student performance in CIFAR-100 and the (more challenging) Tiny-ImageNet dataset, respectively showing accuracy gains of up to 5% and 2% compared to KD. These gains are consistent across a wide range of student and teacher capacities.

DiSK achieves better performance with worse teachers than KD does with even the best teachers. In Table 13.3, we point out that the student performance increases for KD as the teacher complexity is increased for a given student. But note that for the same student, DiSK achieves much better performance with even worse teacher. For instance, for the 'ResNet10-m' student, KD accuracy increases from 66.96% to 68.09% by using high capacity teachers. But 'ResNet10-m' trained with even the worst teacher ('ResNet10-ℓ') achieves 70.03% accuracy. This saves a lot of resources in any application as large teacher requires more training time, and larger compute resources.

DiSK is competitive even in small capacity difference setting. As shown by Table 13.4, DiSK does not loose its competitive edge over the KD even when the student is relatively similar sized as teachers, and shows gains of up to 2.5% relative to KD. We conjecture that the observed gains arise from the fact that DiSK provides scaffolding for hard points to the student in initial training stages, which promotes the student

to learn easy examples first. As training progresses, DiSK removes the discounted help from hard inputs. As a result, the student evolves from simpler hypothesis to the ones consistent with both easy and hard inputs. This justifies our dual parameter (λ) update in Algorithm 7, wherein we periodically increase and decrease λ to enforce and relax the budget constraint.

DiSK students achieve near teacher accuracy while saving up to $8\times$ *MACs &* $5\times$ *Params.* As reported in Table 13.3, student ('ResNet10-m') trained with the teacher ('ResNet10-ℓ') achieves close to the teacher accuracy of 71.99%. In this process, it saves $4\times$ compute and requires $4\times$ less parameters. Similarly, student ('ShuffleNetV2') trained with the teacher ('ResNet32x4') achieves close to the teacher accuracy of 81.45%. In this process, it saves $24\times$ compute and requires $5\times$ less parameters.

DiSK cleverly selects a subset of datapoints and smoothens the loss landscape. As illustrated in Figure 13·1 and 13·2, DiSK judiciously selects a subset of hard-to-learn data points for the students and provide discounted help to the student focus on easily learnable inputs. As a result, it eliminates some bad local minima in the student loss-landscape, and smoothens tihs surface.

DiSK enables the student to reach saturation capacity. In Table 13.3, the performance of KD often suffers as the teacher size is increased, e.g., student ('ResNet10-s') accuracy decreases substantially with the teacher 'ResNet34' versus the teacher 'ResNet18'. In contrast, DiSK saturates the student performance across different teachers. For instance, student ('ResNet10-m') accuracy is $\approx 70\%$ for all the teachers. Thus, we point out that DiSK enables the student to reach saturation. This may be due to the fact that guide g identifies the same set of 'easy' points across different teachers.

Table 13.3: DiSK performance under large capacity mismatch on CIFAR-100 & Tiny-ImageNet: We draw mismatched teachers and students from the ResNet family, and report accuracy of CE trained teachers and students, performance of students distilled using KD and DiSK, and gains of the latter relative to KD.

Architecture		CIFAR-100					Tiny-ImageNet				
		Accuracy (%)					Accuracy (%)				
Teacher	**Student**	**Teacher**	**CE**	**KD**	**DiSK**	**Gain**	**Teacher**	**CE**	**KD**	**DiSK**	**Gain**
ResNet10-ℓ	ResNet10-xxs	71.99	32.05	32.64	**37.56**	**4.92**	52.14	17.44	17.59	**18.62**	**1.03**
	ResNet10-s		52.16	54.92	**58.14**	**3.22**		34.65	35.77	**37.43**	**1.66**
	ResNet10-m		65.24	66.96	**70.03**	**3.07**		44.74	46.01	**48.03**	**2.02**
ResNet10	ResNet10-xxs	75.25	32.05	34.25	**37.84**	**3.59**	56.04	17.44	17.96	**18.55**	**0.59**
	ResNet10-s		52.16	54.95	**58.36**	**3.41**		34.65	36.11	**37.37**	**1.26**
	ResNet10-m		65.24	67.27	**70.15**	**2.88**		44.74	46.08	**48.19**	**2.11**
ResNet18	ResNet10-xxs	76.56	32.05	34.16	**37.8**	**3.64**	62.48	17.44	17.47	**18.53**	**1.06**
	ResNet10-s		52.16	55.76	**58.11**	**2.35**		34.65	35.59	**37.5**	**1.91**
	ResNet10-m		65.24	68.09	**69.86**	**1.77**		44.74	45.91	**47.7**	**1.79**
ResNet34	ResNet10-xxs	80.46	32.05	33.93	**37.78**	**3.85**	63.06	17.44	17.67	**18.91**	**1.24**
	ResNet10-s		52.16	54.19	**58.02**	**3.83**		34.65	35.43	**37.68**	**2.25**
	ResNet10-m		65.24	66.78	**69.89**	**3.11**		44.74	45.89	**47.6**	**1.71**

13.5 Exploratory Experiments

Scaling upto ImageNet setting. We show that DiSK scales easily to the large-scale ImageNet-1K dataset. We train three configurations with DiSK, namely (a) ResNet18(He et al., 2016) student and ResNet50 teacher, (b) ViT-Tiny(Dosovitskiy et al., 2021) student and ViT-Large teacher, and (c) DeiT-Tiny(Touvron et al., 2021) student and ViT-Large teacher. We borrow these models from the timm(Wightman, 2019) library. Table 13.5 shows the DiSK performance on these configurations along with the baseline. It clearly shows that DiSK scales well to ImageNet-1K setup and achieves significant improvements over the baselines.

Less Labelled Data Regime. We explore the performance of all three methods (CE, KD, DiSK) when the amount of labelled data is low. We pick up a teacher (ResNet18 model) trained on full CIFAR-100 data, i.e., 50K data points, and use this to train the different distillation setups. We use four data configurations (number of labelled data: 50K, 37.5K, 25K, 12.5K). Figure 13·3 show the two student models

Table 13.4: DiSK performance with small capacity mismatch on CIFAR-100. We pick standard student and teacher configurations used in the KD literature, and report accuracies and gains similarly to Table 13.3. Feature matching KD baselines are due to (Chen et al., 2022).

Architecture		CIFAR-100							
		Response Matching KD					Feature Matching KD		
		Accuracy (%)					Accuracy (%)		
Teacher	**Student**	**Teacher**	**CE**	**KD**	**DiSK**	**Gain**	**FitNet**	**SemCKD**	**SimKD***
ResNet32x4	ResNet8x4	81.45	73.89	76.25	**76.92**	**0.67**	74.32	76.23	78.08
	ShuffleNetV2		73.74	79.13	**80.23**	**1.1**	75.82	77.62	78.39
	Wide-ResNet-16-2		74.26	76.28	**77.67**	**1.39**	74.70	75.65	77.17
	MobileNetV2x2		69.24	76.05	**77.24**	**1.19**	73.09	73.98	75.43
Wide-ResNet-40-2	ResNet8x4	78.41	73.89	75.15	**76.05**	**0.9**	75.02	75.85	76.75
	ShuffleNetV2		73.74	75.81	**78.33**	**2.52**	-	-	-
	Wide-ResNet-40-1		72.81	74.44	**75.92**	**1.48**	74.17	74.4	75.56
	MobileNetV2x2		69.24	73.92	**76.32**	**2.40**	-	-	-

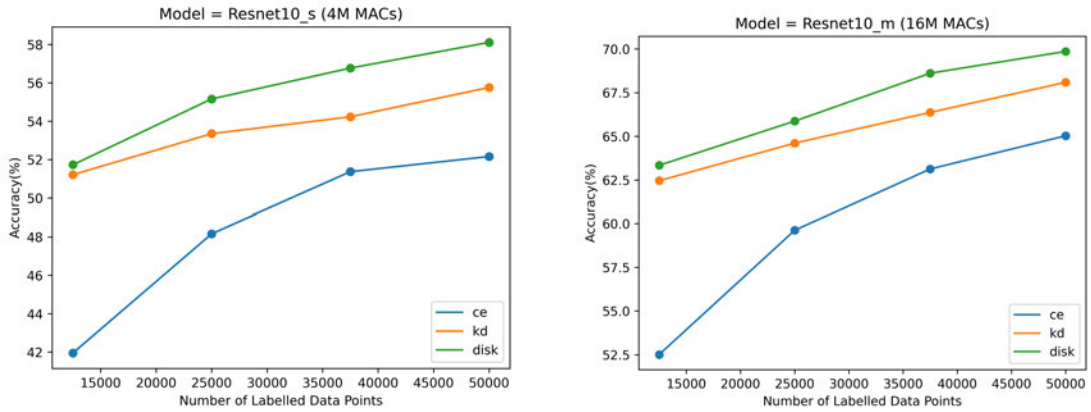
*SimKD accuracy is not emphasized as it employs additional layers beyond the given student architecture and thus not directly comparable to other methods.

Table 13.5: ImageNet-1K: We pick some student and teacher configurations to show that we can scale DiSK to the ImageNet dataset with significant improvements in Top-1 accuracy. We borrow model definitions from timm(Wightman, 2019) repository including the convolutional and transformer vision models.

Teacher	Student	CE	KD	DiSK
ResNet50	ResNet18	69.73	71.29	72.35
ViT-Large (Patch 16, Res. 384)	ViT-Tiny (Patch 16, Res. 224)	75.45	76.61	77.86
ViT-Large (Patch 16, Res. 384)	DeiT-Tiny (Patch 16, Res. 224)	72.2	74.5	75.59

(ResNet10-s, ResNet10-m) trained in this setup. It shows that the gap between DiSK and KD starts to bridge when there are few labelled data points, but the gap starts to increase between KD and DiSK as soon as the number of labelled data points increases. It is due to the fact that when more data is available, the likelihood of hard data points increases for a model.

Self-Distillation. In this setup, we use the same teacher and student architectures (referred to as the self-distillation in the literature). We want to analyze how KD and DiSK fare when the student and teacher are same capacities. We train four different models (ResNet18, Wide-ResNet-40-2, ShuffleNetV2, and MobileNetV2x2)



(a) **(b)**

Figure 13·3: CIFAR-100 Less Labelled Data. Comparing CE, KD, and DiSK, all trained on the same amount of labelled data points (50K, 37.5K, 25K, 12.5K). The teacher is the ResNet18 model trained with CE loss on all labelled data. (a) ResNet10-s (4M MACs) student, and (b) ResNet10-m (16M MACs).

on the full CIFAR-100 dataset. Table 13.6 shows the performance of student models trained using the same teacher architecture. Note that the CE column denotes the teacher's performance (since this checkpoint is used as the teacher's). This experiment shows that DiSK still achieves better performance than KD in the self-distillation setup.

Table 13.6: Self-Distillation (CIFAR-100 dataset): We pick the same student and teacher configurations to show that we can utilize DiSK even in the self-distillation literature. The teacher model is the cross-entropy checkpoint in both KD and DiSK.

Teacher=Student	CE	KD	DiSK
ResNet18	76.56	78.78	80.02
ShuffleNetV2	73.74	74.61	76.03
MobileNetV2x2	69.23	70.63	71.1
Wide-ResNet-40-2	78.42	78.72	80.41

Low-Capacity Teacher + High-Capacity Student. Conventionally, in a knowledge distillation setup, the teacher has a higher model capacity than the student. In this experiment, we probe this aspect and choose the student network to be much larger compared to the teacher. We pick ShuffleNetV2 as the teacher and four

student models (ResNet18, Wide-ResNet(40-2, 16-2), ResNet32x4). We train these models on the CIFAR-100 dataset using CE, KD, and DiSK schemes. We report their performance in Table 13.7. It shows that, in most cases, the gains are marginal compared to the cross-entropy method (except ResNet18 architecture). Further, DiSK has a much better performance than KD in this setup. Concretely, KD does not surpass the CE accuracy, while DiSK surpasses the CE performance.

Table 13.7: Low-Capacity Teacher + High-Capacity Student: We pick the student model to be larger than the teacher network. We use the ShuffleNetV2 model as the teacher. It achieves 73.74% accuracy on the CIFAR-100 dataset.

Student	CE	KD	DiSK
ResNet18	76.56	75.94	80.79
Wide-ResNet-16-2	74.26	73.45	74.67
Wide-ResNet-40-2	78.42	76.33	79.57
ResNet32x4	81.52	80.21	82.39

DiSK + Hybrid-Models. We apply DiSK as the training scheme while learning hybrid models. We pick the student network as ResNet10-s (4M MACs) model that achieves close to 52% accuracy with cross-entropy training. We use the teacher as a $ResNet10-\ell$ (64M MACs) model that achieves close to 72% accuracy with cross-entropy training. We train a hybrid model using these two models (where the student routes low-confidence predictions to the teacher). We use the training procedure described in Chapter 12. Table 13.8 shows the performance of traditional hybrid architecture and the impact of the DiSK procedure in such training setup. Thus, DiSK improves the performance of models in the hybrid setup as well.

13.6 Discussion

In this section, we discuss potential extension of our method beyond Vanilla KD to feature based distillation as well as self-supervised distillation.

Feature based KD methods. Let f_s and f_t denote the features for the student

Table 13.8: Hybrid models (trained with the DiSK objective) for CIFAR-100 at various coverages. Note that Entropy and Hybrid methods are borrowed from Chapter 12, we add the other methods by training Hybrid models with the DiSK objective. Since DiSK has a guide installed during training that decides the hard input instances during training for the student network, we have two guide functions one utilizing student and the other utilizing teacher features.

Algorithm	Base Acc.(%)	Accuracy(%) at Cov.	
		80%	70%
Entropy	52.16	57.77	59.95
Hybrid	52.16	59.32	62.48
Hybrid+DiSK (student guide)	57.16	62.04	64.58
Hybrid+DiSK (teacher guide)	58.14	62.54	64.83

and teacher respectively and Ψ denote the operator such that $\Psi(f_s)$ lies in the same feature space as f_t . Commonly used feature transfer strategy is to minimize the distance between these two representations via loss function such as mean-squared error as shown below by the loss $\mathcal{L}_{ft}$.

$$\mathcal{L}_{ft} = -\frac{1}{N}\sum_{i=1}^{N}\|\Psi(f_s(\mathbf{x}_i))-f_t(\mathbf{x}_i)\|^2; \quad \mathcal{L}_{ft-g} = -\frac{1}{N}\sum_{i=1}^{N}(1-g(\mathbf{x}_i))\|\Psi(f_s(\mathbf{x}_i))-f_t(\mathbf{x}_i)\|^2 \tag{13.6}$$

A simple extension of this feature alignment loss to the selective distillation is shown by the loss $\mathcal{L}_{ft-g}$ that weighs each data point with the helper function decision. Table 13.9 shows this feature matching extension for the SimKD(Chen et al., 2022) scheme. We leave question of finding better selective distillation losses in feature alignment to future work.

Self-Supervision. We can substitute the teacher with a moving average of the student to help guide in the distillation process (a surrogate that helps in learning the hints).

Other Domains. Although our work focuses mainly on models in the vision domain. We point out that conceptually the function g is applicable on models in other domains albeit with some modifications. For instance, we can utilize the censoring

Table 13.9: DiSK performance against feature matching KD on CIFAR-100: Similar setup as in Table 13.4. We integrate DiSK within SimKD (Chen et al., 2022). The gains of using DiSK over KD and using SimKD + DiSK over SimKD are reported. Feature matching KD baselines are due to (Chen et al., 2022).

Architecture		Response Matching KD					Feature Matching KD				
		Accuracy (%)					Accuracy (%)				
Teacher	Student	Teacher	CE	KD	DiSK	Gain	FitNet	SemCKD	SimKD	SimKD + DiSK	Gain
Wide-ResNet-40-2	ResNet8x4	78.41	73.89	75.15	**76.05**	**0.9**	75.02	75.85	76.75	**77.13**	**0.38**
	Wide-ResNet-40-1		72.81	74.44	**75.92**	**1.48**	74.17	74.4	75.56	**76.21**	**0.65**

mechanism in sequential decision making with recurrent neural networks (Kag et al., 2020; Kag and Saligrama, 2021a). We already show that DiSK can be applied to the vision transformers in the Table 13.5. We leave the application and modifications to DiSK for transformers in language domain for future research.

Part IV

Conclusion & Future Directions

Chapter 14

Conclusions and Future Directions

14.1 Conclusion

In this thesis, we studied various ways to reduce the resource footprint of deep neural networks. The central theme revolved around addressing challenges in existing architecture and algorithmic design.

In the first part (Chapter 2-5), we designed low-complexity RNNs by addressing various challenges such as vanishing/exploding gradients, noise amplifications, and BPTT complexity.

First, drawing inspiration from Continuous RNNs, we developed discrete time incremental RNN (iRNN) (Chapter 3). Leveraging the equilibrium properties of CTRNN, iRNN solves exploding/vanishing gradient problem. We show that iRNN improved gradients directly correlate with improved test accuracy. Several experiments demonstrate iRNN's responsiveness to long-term dependency tasks. In addition, due to its smooth low-dimensional trajectories, it has a lightweight footprint that can be leveraged for IoT applications.

Next, we designed time adaptive RNN (TARNN) (Chapter 4) for learning complex patterns in sequential data. By modifying the time constants of an ODE-RNN, it learns to skip uninformative inputs while focusing on informative input segments. Additionally, we develop parameter constraints, which lead to lossless information propagation from informative inputs by mitigating gradient explosion or decay. Our

empirical evaluation validates our approach against competitors with similar complexity. Indeed, we realize competitive performance with a lighter memory footprint and faster training time without suffering performance degradation or increased inference time.

Finally, we proposed a novel forward-propagation-through-time (FPTT) (Chapter 5) algorithm for training RNNs based on sequentially updating parameters forward through time. At a time t, it involves taking a gradient step of an instantaneously constructed regularized risk, where the regularizer evolves dynamically based on the history. Our method exhibits a lightweight footprint and improves LSTM trainability for benchmark long-term dependency tasks, bypassing vanishing/exploding gradient issues encountered while training LSTMs with BPTT. As a result, we show that LSTMs have sufficient capacity and often realize results competitive to much higher capacity models.

In the second part (Chapter 6-9), we propose new convolutional residual layers and a new training algorithm to improve CNN architectures.

First, we developed a novel feature layer that couples the input and output feature map with PDE constraints. The proposed Global layer (Chapter 7) is readily deployable across many existing architectures. We show that the architectures with Global layers are more compact, shallower, and require less compute for inference and training. Empirical evaluations demonstrate that the proposed layer provides $2-5\times$ storage and computational savings.

Next, we constructed a novel spatial interpolation (SI) (Chapter 8) scheme that exploits spatial smoothness in convolutional residual blocks. We show that it can be easily integrated with popular convolutional blocks, resulting in novel architectures with significantly reduced computational footprint despite little loss in performance. Our SI-MobileNetV3 and SI-EfficientNet architectures can get nearly the same accu-

racy as the baseline architectures while reducing the computational cost by up to 40%. Also, crucially, our architecture can be seamlessly implemented using the standard set of operators/hardware. Our empirical evaluations demonstrated the effectiveness of the proposed scheme on object classification and semantic segmentation tasks.

Finally, we proposed Distributionally Constrained Learning (DCL) (Chapter 9), a novel training method for improving DNN generalization. DCL enforces data-dependent constraints using a cosine-scheduled multi-phase constrained training. We propose two distributional constraints, an input-variability constraint, which penalizes the KL divergences of model outputs on different input examples, and a model-variability constraint that de-sensitizes model variations arising from randomly sampled batches. We conduct experiments on several benchmark datasets (CIFAR-100, Tiny-ImageNet, ImageNet-1k) and diverse architectures (ShuffleNet, Resnet-18, Resnet-50, and ViT-Tiny) and demonstrate SOTA performance. On CIFAR-100, our method surprisingly matches ImageNet pre-trained networks.

In the last part (Chapter 10-13), we explore the notion of input hardness and design input adaptive architectures and algorithms.

First, we presented a novel method for selective classification that was motivated by relaxing a new formulation (Chapter 11) of the problem. The formulation is natural, equivalent to prior proposals, and amenable to standard statistical analyses. The OSP-based relaxation is theoretically efficient in the low target error regime, and the resulting method is efficiently trainable via standard techniques and outperforms SOTA methods across target error levels. Further, it is the first method to non-trivially outperform naïve post-hoc solutions and thus represents a significant step in the practical approaches to selective classification.

Next, we proposed a novel hybrid design (Chapter 12) where an edge-device selectively queries the cloud only on those hard instances that the cloud can classify

correctly to optimize accuracy under latency and edge-device constraints. We propose an end-to-end method to train neural architectures, base predictors, and routing models. We use novel proxy supervision for training routing models. Our method adapts seamlessly and near optimally across different latency regimes. Empirically, on the large-scale ImageNet classification dataset, our proposed method, deployed on an MCU, exhibits a 25% reduction in latency compared to cloud-only processing while suffering no excess classification loss.

Finally, we developed a new knowledge distillation algorithm (Chapter 13) that utilizes teacher predictions in novel ways combining predictive targets with scaffolding the student on hard-to-learn points through a guide function. Our method is particularly relevant when there is a large gap between student and teacher capacities. We show that it allows convergence to better minima based on two key properties. Our guide function allows for censoring hard-to-learn examples, and the predictive targets set by the teacher on remaining points allow for eliminating bad local minima and smoothening the resulting student loss landscape. Against vanilla KD, we achieve improved performance and are comparable to more intrusive techniques that leverage feature matching on benchmark datasets.

To summarize, we explored two avenues to improve DNN resource efficiency. In the first direction, we designed new architectures and training algorithms to improve performance at a fixed budget. These low-complexity DNNs (both convolutional and recurrent) achieve superior performance. Since these architectures are static in nature (i.e., they utilize the same amount of resources irrespective of the input hardness), they give us the second direction to explore. In this direction, we introduced the notion of input hardness and enabled the abstention capabilities in neural networks. These networks allow sending complex examples to expert models as well as designing training algorithms that incorporate input hardness.

14.2 Future Research Directions

Before discussing future research work, let us compare GPT-3(Brown et al., 2020), a SOTA DNN, and the human brain. (Huang, 2022) notes that GPT-3 has twice the number of neurons compared to the human brain, yet, it can only solve language processing tasks compared to the myriad tasks performed by the human brain (audio, video, language, sentiment, creativity, etc.). In addition, training this network alone requires 50× more energy than an average human consumes over their entire lifespan. Thus, current SOTA DNNs are inefficient, unspecialized, and underperforming compared to the human brain. It should put in perspective how far behind the field is in the efficiency and specialization of the DNNs. Below we lay down a few future research directions to improve this efficiency and specialization viewpoint. Below, I have listed some concrete ideas that tackle this issue.

- **Efficient Transformers.** Transformers(Vaswani et al., 2017) have emerged as a one-stop solution for many learning tasks in language, vision, and speech domains. They evolved from their simple siblings like RNNs, and do not inherit long-range dependency learning issues. But, this improvement comes with many challenges: (a) large model size and inference complexity as it requires access to dependency calculation that scales quadratically in sequence length, and (b) long training times and high working memory. Ideas from low-complexity RNNs can be extended to generate efficient transformers with significantly less resource consumption. In addition, the FPTT algorithm for RNNs can be modified to reduce transformer training time. Similarly, input hardness can be incorporated into the transformer architectures by adaptively invoking a low-capacity transformer for simple inputs and routing to a high-capacity transformer for complex inputs.

- **Context Aware DNNs.** Traditional DNNs rely on a single feature generation

pipeline that focuses on the input and outputs the predictions. In addition, the human brain relies on tangential observations to infer the context, speeding up the inference and predictive performance. DNNs should follow a similar strategy for the low-complexity and input hardness-aware architectures. Side information should help improve the DNN generalization. In the visual or textual domain, a context can be as simple as the prominent feature locations, objects, or actor information. It will enable a DNN to parse useful features before delving into the details or effectively censoring unnecessary information. I believe some logic that efficiently infers contextual information will benefit any neural architecture.

- **Enforcing Constraints During Training.** In this thesis, we proposed selective distillation (DiSK) and distributionally constrained learning (DCL), wherein we designed a simple primal-dual scheme that gradually enforces constraints during training. We have only explored the tip of the iceberg. In the DNN literature, a better generalizing minimum has various characteristics. I believe many characterizations should be enforced as constraints during the training to yield efficient DNNs.

- **Applications.** We can deploy abstaining and hybrid models in many mobile applications, such as Android/iOS cameras, Alexa/Siri, Oculus, etc. In this setup, the low-capacity abstaining model resides on the device and infers as many queries as it can confidently. Once the edge model decides the uncertainty about user input, it can provide the user with its low-confident prediction and an option for the user to consent to the cloud model in case the user wants high-quality predictions. It includes image classification, sentiment prediction, object detection, semantic segmentation, conversations, etc. Similarly, the proposed low-complexity architectures improve computational and storage requirements for all resource constraints. Thus, they can be easily scaled and deployed in existing machine learning applications, including cloud devices with lots of resources and edge devices with limited resources.

Part V

Appendix: Datasets, Experiment Details and Proofs

Appendix A

Datasets

In this chapter, we consolidate all the publicly available datasets used in this thesis since many of these datasets are used across different chapters and parts. We defer the toy examples discussion to the individual chapters for self containment.

A.1 Vision Datasets

In this section, we list all the vision datasets used in this work. We will include the train/test split as well as the data augmentation strategies used in various experiments.

A.1.1 MNIST-10 (LeCun et al., 2010)

This dataset consists of 10 classes with grayscale images of 28×28 pixels. There are $60,000$ images in the training set and $10,000$ images in the test set. We normalize the data to be mean 0 and variance 1.

A.1.2 SVHN-10 (Netzer et al., 2011)

This is a real-world image dataset consisting of 10 digits (digits $1 \rightarrow 9$ are labelled from $1 \rightarrow 9$ and digit 0 is labelled as 10). It has RGB images of size $32 \times 32 \times 3$. This dataset is similar to MNIST-10(A.1.1), but the data is more realistic since the

images are captured from natural scene images. The training set consists of $73,257$ images and the test set consists of $26,032$ images.

A.1.3 Cats & Dogs

This is binary classification dataset for distinguishing dogs and cats. It is available at https://www.kaggle.com/c/dogs-vs-cats. This dataset consists of $25,000$ images in the training set.

A.1.4 CIFAR-10/100 (Krizhevsky and Hinton, 2009)

This dataset consists of RGB images of 32×32 pixels. It contains $50,000$ training and $10,000$ test images. It has two variants: (a) CIFAR-10 images are drawn from 10 classes, and (b) CIFAR-100 images are drawn from 100 classes. Unless explicitly stated, we follow standard data augmentation techniques (mirroring/shifting) used in earlier works (He et al., 2016; Huang et al., 2017), followed by normalization to a standard gaussian across channels.

A.1.5 ImageNet-1K (Russakovsky et al., 2015)

It is the popular ILSVRC 2012 classification dataset. This 1000 way classification dataset consists of 1.28 million training and $50,000$ validation images. We follow the standard data augmentation (mirroring, resize and crop to shape 224×224) for training and single crop for testing. Similar to previous works, we report results on the validation set.

A.1.6 Tiny-ImageNet (Le and Yang, 2015)

It is a trimmed down version of the ImageNet-1K dataset. It contains 100K training and 10K test images from 200 classes with size $64 \times 64 \times 3$. We use the standard

Table A.1: Dataset Statistics & Long Term Dependence

Dataset	Avg. Activity Time	Input Time	Sequence Ratio	#Train	#Fts	#Steps	#Test
Google-30	25ms	1000ms	3/99	51,088	32	99	6,835
HAR-2	256ms	2560ms	13/128	7,352	9	128	2,947
Noisy-MNIST	28	1000	7/250	60,000	28	1000	10,000
Noisy-CIFAR	32	1000	4/125	60,000	96	1000	10,000
Pixel-MNIST				60,000	1	784	10,000
Permuted-MNIST				60,000	1	784	10,000

data augmentations including 'RandomCrop', 'RandomHorizontalFlip', 'AutoAugment'(Cubuk et al., 2019), 'Cutout' (DeVries and Taylor, 2017), and 'Mean-Std-Normalization'.

A.1.7 Cityscapes (Cordts et al., 2016)

This dataset corresponds to a semantic segmentation task. It consists of images of size 1024×2048 pixels, along with high-quality pixel-level annotations for 19 foreground classes and one background class in the city street setup. It consists of 5000 images split into the train (2975 images), validation (500 images), and test (1525 images) splits. Additionally, it consists of 20K coarsely annotated images. In our experiments, we only use the fine annotations and report the mIoU (mean intersection over union) metric on the test set.

A.2 Sequential & Long Range Dependency Datasets

In this section, we list all the sequential datasets used in this work, including the long range dependency datasets. Table A.1 provides the statistics for these datasets.

A.2.1 Google-12 & Google-30 (Warden, 2017)

Google Speech Commands dataset contains 1 second long utterances of 30 short words (30 classes) sampled at 16KHz. Standard log Mel-filter-bank featurization with 32 filters over a window size of 25ms and stride of 10ms gave 99 timesteps of 32 filter responses for a 1-second audio clip. For the 12 class version, 10 classes used in Kaggle's Tensorflow Speech Recognition challenge[1] were used and remaining two classes were noise and background sounds (taken randomly from remaining 20 short word utterances). Both the datasets were zero mean - unit variance normalized during training and prediction.

A.2.2 HAR-2 (Anguita et al., 2013)

[2] Human Activity Recognition (HAR) dataset was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. The features available on the repository were directly used for experiments. The 6 activities were merged to get the binarized version. The classes Sitting, Laying, Walking_Upstairs and Standing, Walking, Walking_Downstairs were merged to obtain the two classes. The dataset was zero mean - unit variance normalized during training and prediction.

A.2.3 Permute-Pixel MNIST and Pixel-CIFAR-10

These are sequential variants of the popular image classification datasets: MNIST (Lecun et al., 1998) and CIFAR-10 (Krizhevsky and Hinton, 2009). MNIST consists of images of 10 digits with shape $28 \times 28 \times 1$, while CIFAR-10 consists of images with shape $32 \times 32 \times 3$. The input images are flattened into a sequence (row-wise). At each time step, 1 and 3 pixels are presented as the input for MNIST and CIFAR

[1] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

[2] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

datsets respectively. This construction results in Pixel MNIST and CIFAR datasets with 784 and 1024 length sequences respectively. While Permute-MNIST is obtained by applying a fixed permutation on the Pixel MNIST sequence. This creates a harder problem than the Pixel setting since there are no obvious patterns to explore. MNIST dataset has $60,000$ training and $10,000$ test images while CIFAR-10 dataset has $50,000$ training and $10,000$ test images.

A.2.4 PTB-300

It is a word level language modelling task with the difficult sequence length of 300 and has been studied in many previous works to study long range dependencies in language modeling (Zhang et al., 2018a; Kusupati et al., 2018). This dataset consists of $929K$ training words, $73K$ validation words, and $82K$ test words with $10K$ vocabulary. An example in this dataset consists of a sentence, where the RNN receives an input at timestep t and has to predict the next word (which will be available on the next timestep). Perplexity is used as the evaluation metric on this dataset, where lower values corresponds to better performance. Validation set is used for tuning the hyper-parameters and once the right set of parameters have been found, evaluation is performed on the test set.

A.2.5 PTB-w (McAuley and Leskovec, 2013)

It is the traditional word level language modelling variant of the PTB dataset. It uses 70 as sequence length and we follow (Yang et al., 2018) to setup this experiment. This dataset consists of $929K$ training words, $73K$ validation words, and $82K$ test words with $10K$ vocabulary. Although this is a small scale dataset for studying language modelling, it has been extensively used by previous works (Merity et al., 2018; Yang et al., 2018; Bai et al., 2019b). We follow (Merity et al., 2018) to setup

our experiments for BPTT and FPTT.

A.2.6 PTB-c (McAuley and Leskovec, 2013)

It is the character level modelling task that uses 150 sequence length. It contains 5M characters for training, $396K$ for validation, and $446K$ for testing, with an alphabet size of 50. The evaluation metric used for this dataset is bits per character (bpc). Similar to perplexity, the lower value of bpc corresponds to better performance.

A.2.7 NTU RGB-d Skeleton based Action Recognition (Shahroudy et al., 2016)

Skeleton based action recognition is performed on the NTU RGB-d dataset with 60 action classes. We follow (Li et al., 2019b; Li et al., 2018b) in order to create the cross-subject (CS) and cross-view (CV) datasets. After eliminating the spurious entries, CS dataset contains 40,091 train and 16,487 test samples, while CV dataset contains $37,646$ train and 18,932 test samples. In this dataset, 5% of the train data is used for hyper-parameter selection.

A.2.8 Noisy-MNIST

To introduce more long-range dependencies to the Pixel-MNIST task, we define a more challenging task called the Noisy-MNIST, inspired by the noise padded experiments in (Chang et al., 2019). Instead of feeding in one pixel at one time, we input each row of a MNIST image at every time step. After the first 28 time steps, we input independent standard Gaussian noise for the remaining time steps. Since a MNIST image is of size 28 with 1 RGB channels, the input dimension is m = 28. The total number of time steps is set to T = 1000. In other words, only the first 28 time steps of input contain salient information, all remaining 972 time steps are merely

random noise. For a model to correctly classify an input image, it has to remember the information from a long time ago. This task is conceptually more difficult than the pixel-by-pixel MNIST, although the total amount of signal in the input sequence is the same.

A.2.9 Noisy-CIFAR

This is exactly replica of the noise paded CIFAR task mentioned in (Chang et al., 2019). Instead of feeding in one pixel at one time, we input each row of a CIFAR-10 image at every time step. After the first 32 time steps, we input independent standard Gaussian noise for the remaining time steps. Since a CIFAR-10 image is of size 32 with three RGB channels, the input dimension is m = 96. The total number of time steps is set to T = 1000. In other words, only the first 32 time steps of input contain salient information, all remaining 968 time steps are merely random noise. For a model to correctly classify an input image, it has to remember the information from a long time ago. This task is conceptually more difficult than the pixel-by-pixel CIFAR-10, although the total amount of signal in the input sequence is the same.

A.2.10 Addition Task (Hochreiter and Schmidhuber, 1997b)

We closely follow the adding problem defined in (Arjovsky et al., 2016; Hochreiter and Schmidhuber, 1997b) to explain the task at hand. Each input consists of two sequences of length T. The first sequence, which we denote x, consists of numbers sampled uniformly at random $\mathcal{U}[0, 1]$. The second sequence is an indicator sequence consisting of exactly two entries of 1 and remaining entries 0. The first 1 entry is located uniformly at random in the first half of the sequence, whilst the second 1 entry is located uniformly at random in the second half. The output is the sum of the two entries of the first sequence, corresponding to where the 1 entries are located

in the second sequence. A naive strategy of predicting 1 as the output regardless of the input sequence gives an expected mean squared error of 0.167, the variance of the sum of two independent uniform distributions.

A.2.11 Copying Task (Hochreiter and Schmidhuber, 1997b)

Following a similar setup to (Arjovsky et al., 2016; Hochreiter and Schmidhuber, 1997b), we outline the copy memory task. Consider 10 categories, $\{a_i\}_{i=0}^{9}$. The input takes the form of a $T + 20$ length vector of categories, where we test over a range of values of T. The first 10 entries are sampled uniformly, independently and with replacement from $\{a_i\}_{i=0}^{7}$, and represent the sequence which will need to be remembered. The next $T - 1$ entries are set to a_8, which can be thought of as the 'blank' category. The next single entry is a_9, which represents a delimiter, which should indicate to the algorithm that it is now required to reproduce the initial 10 categories in the output. The remaining 10 entries are set to a_8. The required output sequence consists of $T + 10$ repeated entries of a_8, followed by the first 10 categories of the input sequence in exactly the same order. The goal is to minimize the average cross entropy of category predictions at each time step of the sequence. The task amounts to having to remember a categorical sequence of length 10, for T time steps.

A simple baseline can be established by considering an optimal strategy when no memory is available, which we deem the memoryless strategy. The memoryless strategy would be to predict a_8 for $T + 10$ entries and then predict each of the final 10 categories from the set $\{a_i\}_{i=0}^{7}$ i=0 independently and uniformly at random. The categorical cross entropy of this strategy is $\frac{10\log(8)}{T+20}$

A.2.12 DSA-19

[3] This dataset is based on Daily and Sports Activity (DSA) detection from a resource-constrained IoT wearable device with 5 Xsens MTx sensors having accelerometers, gyroscopes and magnetometers on the torso and four limbs. The features available on the repository were used for experiments. The dataset was zero mean - unit variance normalized during training and prediction.

A.2.13 Yelp-5

Sentiment Classification dataset based on the text reviews[4]. The data consists of 500,000 train points and 500,000 test points from the first 1 million reviews. Each review was clipped or padded to be 300 words long. The vocabulary consisted of 20000 words and 128 dimensional word embeddings were jointly trained with the network.

A.2.14 IMDb (Maas et al., 2011)

This is a sentiment classification dataset with two classes (positive and negative). It consists of raw text of movie reviews. There are 25K train and 25K test data points. We borrow text parsers and tokenizers from the models available in the HuggingFace[5] library.

[3] https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities
[4] https://www.yelp.com/dataset/challenge
[5] https://huggingface.co

Appendix B

Appendix to iRNNs

B.1 Multi-Layer Deep RNN Networks.

We point out in passing that our framework readily admits deep multi-layered networks within a single time-step. Indeed our setup is general; it applies to shallow and deep nets; small and large time steps. As a case in point, the Deep Transition RNN (Pascanu et al., 2013a):

$$h_{m+1} = f_h(h_m, x_{m+1}) = \phi_h(W_L \phi_{L-1}(W_{L-1} \dots W_1 \phi_1(U h_m + W x_{m+1}))$$

is readily accounted by Theorem 1 in an implicit form:

$$h_{m+1} = f_h(h_{m+1} + h_m, x_{m+1}) - h_m.$$

So is Deep-RNN (Hermans and Schrauwen, 2013). The trick is to transform $h_m \to h_m + h_{m+1}$ and $h_{m+1} \to h_m + h_{m+1}$. As such, all we need is smoothness of f_h, which has no restriction on # layers. On the other hand, that we do not have to limit the number of time steps is the *point* of Theorem 1, which asserts that the partial differential of hidden states (which is primarily why vanishing/exploding gradient arises (Pascanu et al., 2013b) in the first place) is identity!!

Algorithm 8 Pseudo Code for computing iRNN hidden states for one input sequence

Input: Sequence $\{x_m\}_{m=1}^T$
Input: Number of recursion steps K
Initialize: Model parameters $\big(U, W, b, \alpha, \{\eta_m^k\}\big)$
Initialize: Hidden state $h_0 = 0$
for $m = 1$ **to** T **do**
  Initialize: g_0 to zero or h_{m-1}
  for $k = 1$ **to** K **do**
    Update : $g_k = g_{k-1} + \eta_m^k\big(\phi(U(g_{k-1} + h_{m-1}) + Wx_m + b) - \alpha(g_{k-1} + h_{m-1})\big)$
  Set: : $h_m = g_K$
Return : Hidden states $\{h_m\}_{m=1}^T$

B.2 Pseudo Code and Implementation

Given an input sequence and iRNN model parameters, the hidden states can be generated with the help of subroutine 8. This routine can be plugged into standard deep learning frameworks such as Tensorflow/PyTorch to learn the model parameters via back-propagation.

B.3 Convergence Guarantees for General Learning Rates.

Theorem 3 (Local Convergence with Linear Rate)**.** *Assume that the function* $F(g_i) \triangleq \phi(U(g_i + h_{k-1}) + Wx_k + b) - (g_i + h_{k-1})$ *and the parameter* $\eta_k^{(i)}$ *in Eq. 3.5 satisfies*

$$[\eta_k^{(i)}]^2 \|\nabla F(g_i)F(g_i)\|^2 + 2\eta_k^{(i)} F(g_i)^\top \nabla F(g_i)F(g_i) < 0, \forall k, \forall i. \tag{B.1}$$

Then there exists $\epsilon > 0$ *such that if* $\|g_0 - h_{eq}\| \leq \epsilon$ *where* h_{eq} *denotes the fixed point, the sequence* g_i *generated by the Euler method converges to the equilibrium solution in* $\mathcal{M}_{eq}(h_{k-1}, x_k)$ *locally with linear rate.*

The proof is based on drawing a connection between the Euler method and inexact Newton methods, and leverages Thm. 2.3 in (Dembo et al., 1982). See appendix

Sec. B.7.1 Thm. 4 and Sec. B.6.5 (for proof, empirical verification).

Corollary 1. *If* $\|\mathbf{I} + \eta_k^{(i)} \nabla F(g_i)\| < 1, \forall k, \forall i$, *the forward propagation (Eq. B.4) is stable and the sequence* $\{g_i\}$ *converges locally at a linear rate.*

The proof is based on Thm. 2.3 in (Dembo et al., 1982), Thm. 3 and Prop. 2 in (Chang et al., 2019). See appendix B.7.1 Corollary. 2

B.4 Baseline Justification

In our experiments section, we stated that some of the potential baselines were removed due to experimental conditions enforced in the setup. Here we clearly justify our choice. Mostly the reasoning is to avoid comparing complementary add-ons and compare the bare-bone cells.

- (Cooijmans et al., 2017) is removed since its an add-on and can be applied to any method. Besides its pixel-mnist results involve dataset specific heuristics.
- (Gong et al., 2018) is also an add-on and hence can be applied to any method.
- (Zilly et al., 2017; Pascanu et al., 2013a; Mujika et al., 2017) denote deep transitioning methods. They are add-ons for any single recurrent block and hence can be applied to any recurrent cell.
- Gating variants of single recurrent cells (Chang et al., 2019; Kusupati et al., 2018) have also been removed. Since iRNN can be extended to a gating variant and hence its just an add-on.

B.5 Hyper-parameters for reproducibility

We report various hyper-parameters we use in our experiments for reproduciblity. As mentioned earlier we mainly use 'ReLU' as the non-linearity and Adam as the

Table B.1: Various hyper-parameters to reproduce results

Dataset	Hidden Units
Google-30	80
HAR-2	80
Pixel-MNIST	128
Permuted-MNIST	128
Noisy-MNIST	128
Noisy-CIFAR	128
Addition Task	128
Copying Task	128
PTB	256

optimizer. Apart from this, other hyper-parameters are mentioned in table C.1.

B.6 Additional Experiments

Table B.2: Other Dataset Statistics & Long Term Dependence

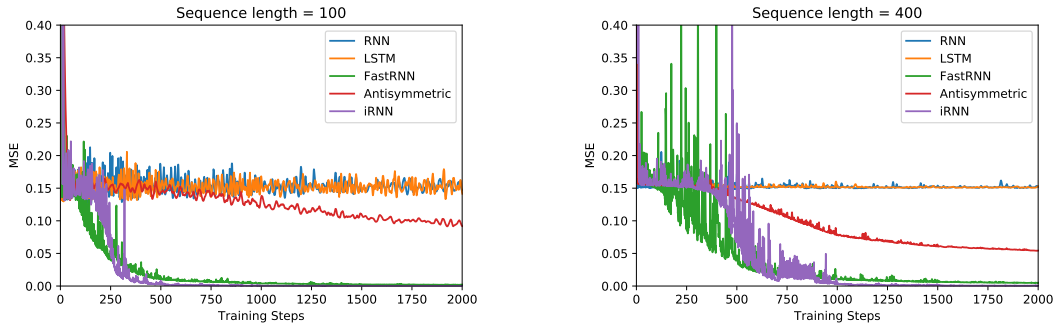
Dataset	Avg. Acitivity Time	Input Time	Sequence Ratio	#Train	#Fts	#Steps	#Test
Google-12	25ms	1000ms	3/99	22,246	32	99	3,081
DSA-19	500ms	5000ms	13/125	4,560	45	125	4,560
Yelp-5	20	300	1/15	500,000	128	300	500,000
PTB				929,589	300	300	82,430

B.6.1 Copying and Addition Tasks

Figure B·1 shows the results for remaining experiments for the addition task for length $100, 400$.

B.6.2 Traditional Datasets

Table B.3 shows the results including left out baselines for Pixel-MNIST and permute-MNIST task. Here we also include star rating prediction on a scale of 1 to 5 of Yelp reviews (Yelp, 2017). Table B.4 shows the results for this dataset.



(a) **(b)**

Figure B·1: Mean Squared Error shown for the Add Task (Sequence Length) : (c) 100 (d) 400

B.6.3 Activity Recognition Datasets

We also include activity recognition tasks: (a)Google-12 (Warden, 2018) , *i.e.* detection of utterances of 10 commands plus background noise and silence and (b) DSA-19 (Altun et al., 2010), Daily and Sports Activity (DSA) detection from a resource-constrained IoT wearable device with 5 Xsens MTx sensors having accelerometers, gyroscopes and magnetometers on the torso and four limbs. Table C.3 shows results for these activities along with some other baselines for activity recognition tasks mentioned in Sec. 3.3.3 and described in Sec. A.2.

B.6.4 PTB Language Modelling

We follow (Kusupati et al., 2018; Zhang et al., 2018a) to setup our PTB experiments. We only pursue one layer language modelling, but with more difficult sequence length (300). Table B.6 reports all the evaluation metrics for the PTB Language modelling task with 1 layer as setup by (Kusupati et al., 2018), including test time and number of parameters.

Table B.3: Results for Pixel-by-Pixel MNIST and Permuted MNIST datasets. K denotes pre-defined recursions embedded in graph to reach equillibrium.

Data set	Algorithm	Accuracy (%)	Train Time (hr)	#Params
Pixel-MNIST	FastRNN	96.44	15.10	33k
	FastGRNN-LSQ	**98.72**	12.57	14k
	RNN	94.10	45.56	14k
	SpectralRNN	97.7		6k
	LSTM	97.81	26.57	53k
	URNN	95.1		16k
	Antisymmetric	98.01	8.61	14k
	iRNN (K=1)	97.73	**2.83**	**4k**
	iRNN (K=2)	98.13	3.11	**4k**
	iRNN (K=3)	98.13	2.93	**4k**
Permute-MNIST	FastRNN	92.68	9.32	8.75k
	SpectralRNN	92.7		8.5k
	LSTM	92.61	19.31	35k
	URNN	91.4		12k
	Antisymmetric	93.59	4.75	14k
	iRNN (K=1)	**95.62**	**2.41**	**8k**

B.6.5 Linear Rate of Convergence to Fixed Point

Empirically we verify the local convergence to a fixed point with linear rate by comparing the Euclidean distance between the approximate solutions, $\mathbf{h}_t^{(k)}$, using Eq. B.2 with $g_0 = 0$ and the fixed points, $\mathbf{h}_t$, computed using FSOLVE from SCIPY. The learnable parameters are initialized suitably and then fixed. We illustrate our results in Fig. B·2, which clearly demonstrates that the approximate solutions tend to converge with linear rate.

$$g_i = g_{i-1} + \eta_t^i(\phi(U(g_{i-1} + h_{t-1}) + Wx_t + b) - \alpha(g_{i-1} + h_{t-1})) \tag{B.2}$$
$$h_t^K = g_K$$

B.6.6 Theoretical Verification

Here we include some experiments to show that our theoretical assumptions hold true.

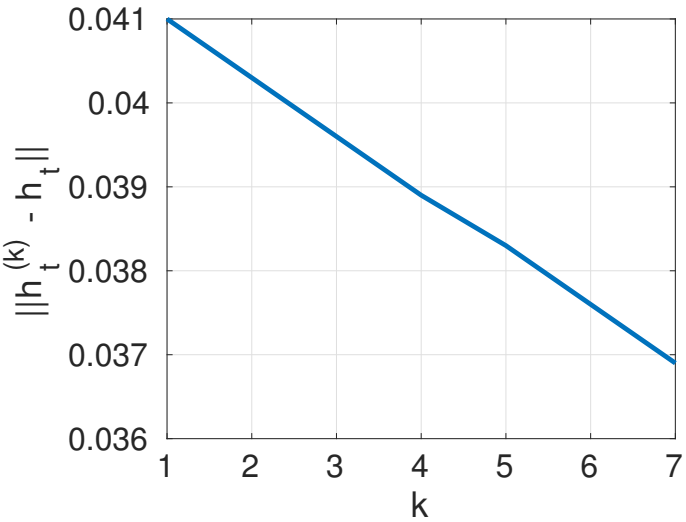
Table B.4: Results for Yelp Dataset.

Data set	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)	Test Time (ms)	#Params
Yelp-5	FastRNN	55.38	130	3.61	**0.4**	32.5k
	FastGRNN-LSQ	**59.51**	130	3.91	0.7	32.5k
	FastGRNN	59.43	8	4.62		
	RNN	47.59	130	3.33	**0.4**	32.5k
	SpectralRNN	56.56	**89**	4.92	0.3	**22k**
	EURNN	59.01	122	72.00		
	LSTM	59.49	516	8.61	1.2	129k
	GRU	59.02	388	8.12	0.8	97k
	Antisymmetric	54.14	130	2.61	**0.4**	32.5k
	UGRNN	58.67	258	4.34		
	iRNN(K=1)	58.16	97.67	**0.31**	**0.4**	25k
	iRNN(K=2)	59.01	98.84	**0.31**	0.7	25k
	iRNN(K=3)	59.34	100	1.16	1.0	25k

Non-Singularity of the matrix D For our iRNN parametrization to satisfy the conditions of having equillibrium points to be locally asymptotically stable, the eigen values of the matrix $D = (\nabla\phi(\cdot)U - \gamma I)$ should be negative. We plot a histogram of the eigenvalues of D for all the points in the HAR-2 dataset. As illustrated in the figure B·3, all the eigenvalues are negative.

B.6.7 Identity Gradient comparison iRNN vs RNN

To verify Theorem. 1 empirically, we train RNN and iRNN on the HAR-2 data set (see more details in Sec. 3.3), respectively, and plot in Fig. B·4 the magnitude of gradient of the last layer $\mathbf{h}_T$ *w.r.t.* the first layer $\mathbf{h}_1$ in log scale to confirm that our approach leads to no vanishing or exploding gradients when the error is back-propagated through time. We also conducted experiments to verify that the gradient of iRNN is norm preserving (see Sec. B.6.8 and Figure 3·3). As we see clearly, RNN suffers from serious vanishing gradient issue in training, while iRNN's backpropagated gradients is close to 1, and the variance arises mainly our approximation of fixed points and stochastic behavior in training networks, demonstrating much better training



(a)

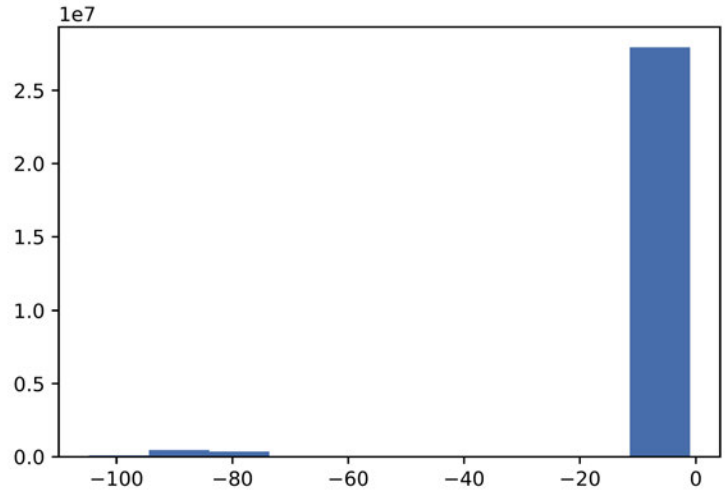
Figure B·2: Linear convergence in iRNN.

stability of iRNN.

B.6.8 Gradient norm w.r.t. loss $\|\frac{\partial L}{\partial h_1}\|$

In addition to the gradient ratio we plot in Sec.3.3.2, we also show in figure B·5, the more popular quantity captured in earlier works (Arjovsky et al., 2016; Zhang et al., 2018a), i.e. the gradient norm w.r.t. loss $\|\frac{\partial L}{\partial h_1}\|$. We emphasize that this quantity alone is misleading in the context of resolving the issue of vanishing/exploding gradients. Since $\|\frac{\partial L}{\partial h_1}\| = \|\frac{\partial L}{\partial h_T}\| * \|\frac{\partial h_T}{\partial h_1}\|$. The long term component controlling the gradients is $\|\frac{\partial h_T}{\partial h_1}\|$, but the other component, $\|\frac{\partial L}{\partial h_T}\|$ could become zero by the virtue that the loss is nearly zero. This happens in our addition task experiment, because MSE is close to zero, we experience nearly 0 value for this quantity. But this is clearly because the MSE is 0. Also note that none of our graphs have log scale, which is not the case in earlier works. The conclusion that can be drawn from the loss-gradient is that it is somewhat stable, and can inform us about quality of convergence.

We also plot $\|\frac{\partial h_T}{\partial h_{T-1}}\|$ in figure B·5 in order to show that indeed iRNN achieves



(a)

Figure B·3: Histogram of the eigenvalues of $\nabla\phi\mathbf{U} - \mathbf{I}$ for iRNN on HAR-2 dataset.

identity gradients everywhere in the time horizon, since fig. 3·3 had shown that the ratio of $\|\frac{\partial h_T}{\partial h_1}\|$ and $\|\frac{\partial h_T}{\partial h_{T-1}}\|$ equals 1 for iRNN.

B.6.9 Different Activation Function

We also performed some experiments for sigmoid activation on HAR-2 dataset. The results for this variant also follow similar pattern as we saw in ReLU variant.

B.7 Proofs

B.7.1 Local Convergence with Linear Rate

Recall that we rewrite the fixed-point constraints in our iRNN as the following ODE:

$$g_k'(t) = F(g_i) \stackrel{def}{=} \phi(U(g_i + h_{k-1}) + Wx_k + b) - (g_i + h_{k-1}); \;\; g(0) = 0. \tag{B.3}$$

Then based on the Euler method, we have the following update rule for solving

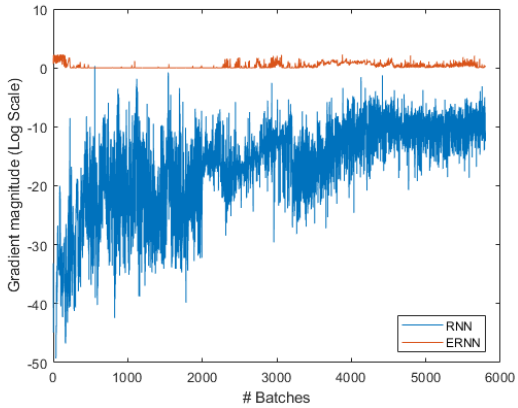
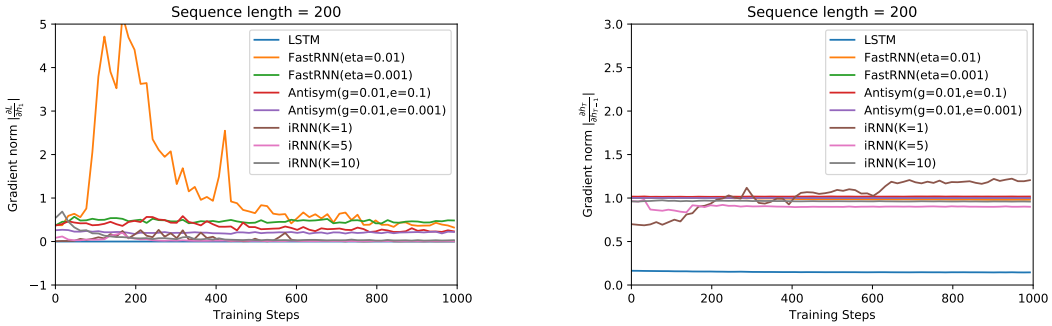


Figure B·4: Comparison between RNN and iRNN on the magnitudes of gradients.



(a) **(b)**

Figure B·5: Exploratory experiments for the Add task : (a) Gradient norms w.r.t. loss $\|\frac{\partial L}{\partial h_1}\|$, (b) Gradient norms $\|\frac{\partial h_T}{\partial h_{T-1}}\|$. This together with Figure 3·3 shows that the gradients are identity everywhere for $K = 10$

fixed-points:

$$g_{i+1} = g_i + \eta_k^{(i)} F(g_i) \tag{B.4}$$

$$= g_i + \eta_k^{(i)} [\phi(U(g_i + h_{k-1}) + Wx_k + b) - (g_i + h_{k-1})]. \tag{B.5}$$

Inexact Newton methods (Dembo et al., 1982) refer to a family of algorithms that aim to solve the equation system $F(z) = 0$ approximately at each iteration using the

following rule:

$$z_{i+1} = z_i + s_i,\ r_i = F(z_i) + \nabla F(z_i) s_i, \tag{B.6}$$

where ∇F denotes the (sub)gradient of function F, and r_i denotes the error at the i-th iteration between $F(z_i)$ and 0.

By drawing the connection between Eq. B.4 and Eq. B.6, we can set $z_i \equiv g_i$ and $s_i \equiv \eta_k^{(i)} F(g_i)$. Then based on Eq. B.6 we have

$$r_i = F(g_i) + \eta_k^{(i)} \nabla F(g_i) F(g_i). \tag{B.7}$$

Lemma 2 (Thm. 2.3 in (Dembo et al., 1982))**.** *Assume that*

$$\frac{\|r_i\|}{\|F(z_i)\|} \leq \tau < 1, \forall k, \tag{B.8}$$

where $\|\cdot\|$ denotes an arbitrary norm and the induced operator norm. There exists $\varepsilon > 0$ such that, if $\|z_0 - z_*\| \leq \varepsilon$, then the sequence of inexact Newton iterates $\{z_i\}$ converges to z_*. Moreover, the convergence is linear in the sense that $\|z_{i+1} - z_*\|_* \leq \tau \|z_i - z_*\|_*$, where $\|y\|_* = \|\nabla F(z_*) y\|$.

Theorem 4 (Local Convergence with Linear Rate)**.** *Assume that the function F in Eq. B.3 and the parameter $\eta_k^{(i)}$ in Eq. B.4 satisfy*

$$[\eta_k^{(i)}]^2 \|\nabla F(g_i) F(g_i)\|^2 + 2\eta_k^{(i)} F(g_i)^\top \nabla F(g_i) F(g_i) < 0, \forall i, \forall k. \tag{B.9}$$

Then there exists $\epsilon > 0$ such that if $\|g_0 - h_{eq}\| \leq \epsilon$ where h_{eq} denotes the fixed point, the sequence $\{g_i\}$ generated by the Euler method converges to the equilibrium solution in $\mathcal{M}_{eq}(h_{k-1}, x_k)$ locally with linear rate.

Proof. By substituting Eq. B.7 into Eq. B.8, to prove local convergence we need to guarantee

$$\|F(g_i) + \eta_k^{(i)} \nabla F(g_i) F(g_i)\| < \|F(g_i)\|. \tag{B.10}$$

By taking the square of both sides in Eq. B.10, we can show that Eq. B.10 is equivalent to Eq. B.9. We then complete our proof.

□

Corollary 2. *Assume that* $\|\mathbf{I} + \eta_k^{(i)} \nabla F(g_i)\| < 1, \forall i, \forall k$ *holds. Then the forward propagation using Eq. B.4 is stable and our sequence* $\{g_i\}$ *converges locally with linear rate.*

Proof. By substituting Eq. B.7 into Eq. B.8 and based on the assumption in the corollary, we have

$$\begin{aligned} \frac{\|r_i\|}{\|F(g_i)\|} &= \frac{\|F(g_i) + \eta_k^{(i)} \nabla F(g_i) F(g_i)\|}{\|F(g_i)\|} \\ &\leq \frac{\|\mathbf{I} + \eta_k^{(i)} \nabla F(g_i)\| \|F(g_i)\|}{\|F(g_i)\|} < 1. \end{aligned} \tag{B.11}$$

Further based on Prop. 2 in (Chang et al., 2019) and Thm. 3, we then complete our proof. □

Table B.5: Results for Activity Recoginition Datasets.

Data set	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)	Test Time (ms)	#Params
HAR-2	FastRNN	94.50	29	0.063	**0.01**	7.5k
	FastGRNN-LSQ	95.38	29	0.081	0.03	7.5k
	FastGRNN	95.59	3	0.10		
	RNN	91.31	29	0.114	**0.01**	7.5k
	SpectralRNN	95.48	525	0.730	0.04	134k
	EURNN	93.11	**12**	0.740		
	LSTM	93.65	74	0.183	0.04	16k
	GRU	93.62	71	0.130	0.02	16k
	Antisymmetric	93.15	29	0.087	**0.01**	7.5k
	UGRNN	94.53	37	0.120		
	iRNN(K=1)	95.32	**17**	0.061	**0.01**	**4k**
	iRNN(K=3)	95.52	**17**	0.081	0.02	**4k**
	iRNN(K=5)	**96.30**	18	**0.018**	0.03	**4k**
DSA-19	FastRNN	84.14	97	0.032	**0.01**	17.5k
	FastGRNN-LSQ	85.00	208	0.036	0.03	35k
	FastGRNN	83.73	3.25	2.10m		
	RNN	71.68	20	0.019	**0.01**	**3.5k**
	SpectralRNN	80.37	50	0.038	0.02	8.8k
	LSTM	84.84	526	0.043	0.06	92k
	GRU	84.84	270	0.039	0.03	47k
	Antisymmetric	85.37	32	0.031	**0.01**	8.3k
	UGRNN	84.74	399	0.039		
	iRNN(K=1)	**88.11**	**19**	0.015	**0.01**	**3.5k**
	iRNN(K=3)	85.20	**19**	0.020	0.02	**3.5k**
	iRNN(K=5)	87.37	20	**0.005**	0.03	**3.5k**
Google-12	FastRNN	92.21	56	0.61	**0.01**	12k
	FastGRNN-LSQ	93.18	57	0.63	0.03	12k
	FastGRNN	92.10	5.5	0.75		
	RNN	73.25	56	1.11	**0.01**	12k
	SpectralRNN	91.59	228	19.0	0.05	49k
	EURNN	76.79	210	120.00		
	LSTM	92.30	212	1.36	0.05	45k
	GRU	93.15	248	1.23	0.05	53k
	Antisymmetric	89.91	57	0.71	**0.01**	12k
	UGRNN	92.63	75	0.78		
	iRNN(K=1)	93.93	**36**	0.20	**0.01**	**8.1k**
	iRNN(K=3)	94.16	37	0.33	0.03	**8.1k**
	iRNN(K=5)	**94.71**	38	**0.17**	0.05	**8.1k**
Google-30	FastRNN	91.60	96	1.30	**0.01**	18k
	FastGRNN-LSQ	92.03	45	1.41	**0.01**	**8.5k**
	FastGRNN	90.78	6.25	1.77		
	RNN	80.05	63	2.13	**0.01**	12k
	SpectralRNN	88.73	128	11.0	0.03	24k
	EURNN	56.35	135	19.00		
	LSTM	90.31	219	2.63	0.05	41k
	GRU	91.41	257	2.70	0.05	48.5k
	Antisymmetric	90.91	64	0.54	**0.01**	12k
	UGRNN	90.54	260	2.11		
	iRNN(K=1)	93.77	**44**	0.44	**0.01**	**8.5k**
	iRNN(K=3)	91.30	**44**	0.44	0.03	**8.5k**
	iRNN(K=5)	**94.23**	45	**0.44**	0.05	**8.5k**

Algorithm	Test Perplexity	Model Size (KB)	Train Time (min)	Test Time (ms)	#Params
FastRNN	127.76	513	11.20	1.2	52.5k
FastGRNN-LSQ	115.92	513	12.53	1.5	52.5k
FastGRNN	116.11	39	13.75		
RNN	144.71	**129**	9.11	**0.3**	**13.2k**
SpectralRNN	130.20	242	-	0.6	24.8k
LSTM	117.41	2052	13.52	4.8	210k
UGRNN	119.71	256	11.12	0.6	26.3k
iRNN(K=1)	**115.71**	288	**7.11**	0.6	29.5k

Table B.6: PTB Language Modeling: 1 Layer. To be consistent with our other experiments we used a low-dim $\mathbf{U}$; For this size our results did not significantly improve with K. This is the dataset of (Kusupati et al., 2018) which uses sequence length 300 as opposed to 30 in the conventional PTB.

Data set	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)	Activation	#Params
HAR-2	iRNN(K=1)	95.32	**17**	0.061	ReLU	**4k**
	iRNN(K=3)	95.52	**17**	0.081	ReLU	**4k**
	iRNN(K=5)	**96.30**	18	**0.018**	ReLU	**4k**
	iRNN(K=1)	92.16	**17**	0.065	Sigmoid	**4k**
	iRNN(K=3)	93.35	**17**	0.078	Sigmoid	**4k**
	iRNN(K=5)	95.30	18	0.020	Sigmoid	**4k**

Table B.7: HAR-2 dataset (Sigmoid, ReLU activations): K denotes pre-defined recursions embedded in graph to reach equillibrium.

Appendix C

Appendix to TARNNs

C.1 Proofs

Proposition 6. *Consider the ODE in Eq. 4.5 and assumptions on* $\mathbf{A}$ *described above. Suppose we have* $\|U\| < \alpha/2$, *and* $\phi(\cdot)$ *is 1-Lipshitz function, it follows that, for any given,* $\boldsymbol{\beta}$, $\mathbf{u}_m$, *an equilibrium point exists and is unique.*

To prove the proposition, we must find a solution to the non-linear equation $\mathbf{Az}+\mathbf{Bu}_m+\phi(\mathbf{Uz}+\mathbf{Wu}_m) = 0$ and show that it is unique. We do this by constructing a fixed-point iterate, and show that the iteration is contractive.

To this end, define $\Gamma(\mathbf{z}) = \mathbf{z} + \eta(\mathbf{Az} + \mathbf{Bu}_m + \phi(\mathbf{Uz} + \mathbf{Wu}_m))$, and note that for any two $\mathbf{z}, \mathbf{z}' \in \mathbb{R}^D$, we find,

$$
\begin{aligned}
\|\Gamma(\mathbf{z}) - \Gamma(\mathbf{z}')\| &\leq \|(I + \eta\mathbf{A})(\mathbf{z} - \mathbf{z}')\| + \eta\|\phi(\mathbf{Uz} + \mathbf{Wu}_m)) - \phi(\mathbf{Uz}' + \mathbf{Wu}_m))\| \\
&\leq \|(I + \eta\mathbf{A})(\mathbf{z} - \mathbf{z}')\| + \eta\|\mathbf{U}(\mathbf{z} - \mathbf{z}')\| \\
&\leq \sigma_{\max}(I + \eta\mathbf{A})\|(\mathbf{z} - \mathbf{z}')\| + \eta\|\mathbf{U}(\mathbf{z} - \mathbf{z}')\| \\
&\leq \sigma_{\max}(I + \eta\mathbf{A})\|\mathbf{z} - \mathbf{z}'\| + \eta\|\mathbf{U}\|\|\mathbf{z} - \mathbf{z}'\| \\
\implies \|\Gamma(\mathbf{z}) - \Gamma(\mathbf{z}')\| &\leq \Big(\sigma_{\max}(I + \eta\mathbf{A}) + \eta\|\mathbf{U}\|\Big)\|\mathbf{z} - \mathbf{z}'\| = \gamma\|\mathbf{z} - \mathbf{z}'\| \qquad \text{(C.1)}
\end{aligned}
$$

If the constant $\gamma < 1$, then the above inequality proves that Γ is a contraction. The result then follows by invoking the Banach fixed point theorem (contraction-mapping

theorem). All that remains to show is that $\gamma = \sigma_{\max}(I + \eta\mathbf{A}) + \eta\|\mathbf{U}\| < 1$. From assumptions we have, $\sigma_{\max}(I + \eta\mathbf{A}) \leq 1 - \alpha\eta$ and $\|\mathbf{U}\| < \alpha$, where $\alpha > 0$; $\implies \gamma < 1 - \alpha\eta + \alpha\eta \leq 1$.

Proposition 7. *With the setup in Proposition 3, and regardless of $\boldsymbol{\beta}$, the equilibrium point is globally asymptotically stable, and the discrete Euler recursion converges to the equilibrium solution at a linear rate.*

Let $\mathbf{z}^*$ be the equilibrium solution, i.e. $\mathbf{A}\mathbf{z}^* + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{u}_m) = 0$. We consider the Lyapunov function $V(\mathbf{z}(t)) = \|\mathbf{z}(t) - \mathbf{z}^*\|^2$ and show that it is monotonically decreasing along the ODE system trajectories. Observe that, as per our setup, components where $(\boldsymbol{\beta})_j = 0$ does not pose a problem, because those states remain frozen, and serve as an additional exogenous input in our ODE. Consequently, we can assume without loss of generality that $(\boldsymbol{\beta})_j = 1$ for all $j \in [D]$. The gradient of the Lyapunov function along the ODE system trajectories can be written as

$$
\begin{aligned}
&\frac{dV(z(t))}{dt} = (\dot{\mathbf{z}}(t))^\top(\mathbf{z}(t) - \mathbf{z}^*) + (\mathbf{z}(t) - \mathbf{z}^*)^\top\dot{\mathbf{z}}(t) \\
=&(\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
+&(\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m)) \\
=&(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \mathbf{B}(\mathbf{u}_m - \mathbf{u}_m) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) - \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{x}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
&+ (\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \mathbf{B}(\mathbf{u}_m - \mathbf{u}_m) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) - \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{u}_m)) \\
=&(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) - \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{u}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
&+ (\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) - \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{u}_m)) \\
=&(\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A} + \mathbf{A}^\top)(\mathbf{z}(t) - \mathbf{z}^*) + 2(\phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) - \phi(\mathbf{U}\mathbf{z}^* + \mathbf{W}\mathbf{u}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*)
\end{aligned}
$$

We now invoke Cauchy-Schwartz inequality to bound the second term, namely,

$$
\begin{aligned}
&|(\phi(\mathbf{U}\mathbf{z}(t)+\mathbf{W}\mathbf{u}_m)-\phi(\mathbf{U}\mathbf{z}^*+\mathbf{W}\mathbf{u}_m))^\top(\mathbf{z}(t)-\mathbf{z}^*)| \\
&\leq \|(\phi(\mathbf{U}\mathbf{z}(t)+\mathbf{W}\mathbf{u}_m)-\phi(\mathbf{U}\mathbf{z}^*+\mathbf{W}\mathbf{u}_m))\|\|\mathbf{z}(t)-\mathbf{z}^*\| \\
&\leq \|\mathbf{U}\|\|\mathbf{z}(t)-\mathbf{z}^*\|\|\mathbf{z}(t)-\mathbf{z}^*\| < \|\mathbf{z}(t)-\mathbf{z}^*\|^2
\end{aligned}
$$

where in the last inequality we used the fact that $\phi(\cdot)$ is 1-Lipshitz and $\|\mathbf{U}\| < \alpha \leq 1$. As a result, we have,

$$
\frac{dV(z(t))}{dt} < (\lambda_{\max}(\mathbf{A}+\mathbf{A}^T)+1)\|\mathbf{z}-\mathbf{z}^*\|^2 \leq 0
$$

where the last inequality follows because, we have $(\lambda_{\max}(\mathbf{A}+\mathbf{A}^T) \leq -1)$. This shows that the ODE is globally asymptotically stable and converges to a unique equilibrium point. To show a linear rate of convergence we note that K-fold iterations of the Euler method (see Prop 3), $\mathbf{z}^k = \Gamma(\mathbf{z}^{k-1}) = \mathbf{z}^{k-1} + \eta(\mathbf{A}\mathbf{z}^{k-1} + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}^{k-1} + \mathbf{W}\mathbf{u}_m))$, results in,

$$
\|\mathbf{z}^K - \mathbf{z}^*\| \leq \gamma^K \|\mathbf{z}^0 - \mathbf{z}^*\|
$$

which follows directly from the fact that $\mathbf{z}^K = \Gamma(\mathbf{z}^{K-1})$, $\mathbf{z}^* = \Gamma(\mathbf{z}^*)$, and Γ is a contraction as obtained by Eq. C.1. This establishes the linear-rate of convergence.

Proof of Theorem 2

Note that, when $\beta_i = 0$, $\mathbf{s}_m(i) = \mathbf{s}_{m-1}(i)$. On the other hand when $\beta_i > 0$, the system is in equilibrium, and for those components, j, we have

$$
(\dot{\mathbf{z}}(t))_j = (F(\mathbf{z}(t), \mathbf{u}_m))_j = 0, \text{ where } F(\mathbf{z}(t), \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))
$$

Now $(F(\mathbf{s}_m, \mathbf{u}_m))_k = 0$ regardless of β_k. This is because if $\beta_k(\mathbf{u}_m) > 0$ we reach equilibrium, and $\dot{\mathbf{z}}(t) = 0$, and on the other have if $\beta_k = 0$ then $(F(\mathbf{s}_m, \mathbf{u}_m))_k = 0$ in

any case. With this in mind, define $D = \text{diag}[\mathbf{1}_{\beta_j(\mathbf{u}_m)>0}]$. We then write the vector $\mathbf{s}_m = D\mathbf{s}_m + (I - D)\mathbf{s}_{m-1}$. Let $J_{m,m-1}$ denote the Jacobian of $\mathbf{s}_m$ with respect to $\mathbf{s}_{m-1}$. Taking derivatives we get,

$$\begin{aligned} 0 = \nabla F(\mathbf{s}_m, \mathbf{u}_m) =& \beta(\mathbf{u}_m) \circ (\mathbf{A}(DJ_{m,m-1} + (I - D)) + \mathbf{B}_2) \\ &+ \beta(\mathbf{u}_m) \circ (\nabla\phi(\mathbf{U}(DJ_{m,m-1} + (I - D)) + \mathbf{W}_2)) \\ &+ D\nabla\sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x\mathbf{u}_m)(\mathbf{A}\mathbf{s}_m + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{s}_m + \mathbf{W}\mathbf{u}_m)) \end{aligned}$$

First, note that the third term is always zero, due to the fact we noted earlier, namely, if a component is active, then the corresponding state reaches equilibrium, and there is nothing to do if the component is otherwise inactive. Now noting that $\mathbf{A} = \mathbf{B}_2$ and $\mathbf{U} = \mathbf{W}_2$, we get,

$$\nabla F(\mathbf{s}_m, \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A}D)(J_{m,m-1} - I) + \nabla\phi(\cdot)\mathbf{U}D(J_{m,m-1} - I)$$

Collecting the common terms, we have,

$$\nabla F(\mathbf{s}_m, \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A} - \nabla\phi U)D(J_{m,m-1} - I)$$

Now for the case in hand, $\|\nabla\phi U\| < 1$, and since $\|\mathbf{A}\| \geq 1$, the middle term is non-zero. This implies that for all the active components, $(J_{m,m-1})_{kk} = 1$.

For the other case, the proof follows in an identical manner. Specifically, for the non-zero rows of $\mathbf{B}$ the proof is identical, and the claims hold for those associated state components. For the rows with zero rows since,

C.2 Implementation Details

We acquired the publicly available code for the baselines except Antisymmetric RNN (Chang et al., 2019) and Incremental RNN(Kag et al., 2020). We write the RNN cell implemntation for Antisymmetric RNN and Incremental RNNs from the pseudo code provided in their papers. Before running our grid search, we ensured that we were able to reproduce the publicly reported results. Following which we run our experiments for suggested hidden states as per the previous works for each dataset.

In order to avoid non-determinism in the experiments, we initialize both the numpy and tensorflow random library with the same seed number, 1234. Our parameter matrices are initialized with a random normal initializer with mean 0 and standard deviation 0.1 while our time-constant biases are initialized with -3.0 and remaining biases are initialized with 0.

We provide the pseudo code in Algorithm 1 to generate the hidden states of the TARNN. In order to implement this routine on a deep learning framework, we need to elaborate a bit more about the ODESolve function. We implement the Euler iterations described in the practical implementations in the method section. Following the recommendation from (Kag et al., 2020) and the fact that many of these datasets are slowly time varying, we use the $K = 5$ in the Euler recursions to reach the equilibrium. Table C.1 provides the number of hidden units used for different datasets.

Our experiments use hidden size as suggested by (Kusupati et al., 2018; Chang et al., 2019) i.e. 128. We point out that this is not the setting used by (Kerg et al., 2019; Lezcano-Casado and Martínez-Rubio, 2019) as their best results are achieved with much larger state space i.e. 512 state dimension, thus requiring much larger models. Thus, in order to provide fair comparison we only allow state space as 128 dimensions.

In order to enable grid search on the baseline methods, we use the method specific

Table C.1: Various hyper-parameters to reproduce results

Dataset	Hidden Dimension	Learning Rate	L2 regularization	Init η	Epochs τ		Batch Size
Pixel-MNIST	128	$1e^{-2}$	$4.5e^{-6}$	0.08	30	5	128
Permuted-MNIST	128	$1e^{-2}$	$4.5e^{-6}$	0.0008	30	5	128
Noisy-MNIST	128	$1e^{-2}$	$4.5e^{-5}$	0.0008	30	5	512
Noisy-CIFAR	128	$1e^{-2}$	$4.5e^{-5}$	0.001	30	5	256
Addition Task	128	$1e^{-2}$	$1.0e^{-5}$	0.001	2	-	128
Copying Task	128	$1e^{-2}$	$1.0e^{-6}$	0.45	-	-	128
PTB	256	-	-	0.001	100	-	

hyper-parameter values suggested in the respective baselines. We allow the methods to pick the non-linearity from the set { ReLU, tanh, sigmoid }. For Antisymmetric RNN, as per their recommendation we step size from the set $\{0.01, 0.1, 1\}$ and diffusion parameter $\gamma \in \{0.001.0.01, 0.1, 1.0\}$. For nnRNN and expRNN methods, we follow the hyper-parameter search grid as suggested in (Kerg et al., 2019).

We use grid search for tuning the hyper-parameters for the methods. We used the values $[4.5E-6, 4.5E-5, 4.5E-4, 1E-6, 1E-5, 1E-4]$ for L2 regularization. We searched over $[1e-2, 1e-3, 1e-4]$ as the base learning rates which are halved after each $\tau = [5, 10, 20]$ epochs have passed. We allowed the methods to train for $[30, 50, 100, 300]$ epochs. We use ReLU as the non-linearity for all of our experiments except in Copy and PTB tasks where we use tanh as the non-linearity (performs better than ReLU).

We point out that we set $\mathbf{A} = -I$ for all our experiments except Pixel-MNIST and Permute-MNIST tasks where we use $\mathbf{A}$ to be the blocked triangular identity matrix as mentioned in the analysis Section 4.2.1. This allows us to couple the linear part resulting in better performance on these tasks in comparison to the $\mathbf{A} = -I$ configuration.

Note that the settings used for PTB dataset corresponds to the small configuration with 300 as the sequence length. We piggy back on the configuration changes used

in (Kusupati et al., 2018; Kag et al., 2020; Zhang et al., 2018a) which describes the learning rate along with the learning rate schedule and the number of epochs all the methods are trained. Thus, we do not list these hyper-parameters in the table C.1.

C.3 Unitary RNNs do not solve vanishing gradients.

(Lezcano-Casado and Martínez-Rubio, 2019; Kerg et al., 2019) and others propose to "cheaply" design orthonormal transition matrices (OTM), appealing to (Arjovsky et al., 2016) for justification. (Arjovsky et al., 2016) (Eq. 4) only shows an upper-bound with ReLU + OTM. This solves exploding gradients, but the more pernicious vanishing gradients remains (RELU+OTM is discussed in (Pennington et al., 2017) [PSG17]). In (Arjovsky et al., 2016)'s notation with D_k binary diagonal arising from ReLU activations, W unitary, we would need, $\|\partial C/\partial h_T(\prod_{s=t}^{T-1} D_s W^\top)\| \geq \|\partial C/\partial h_T\|$. This is generally not true due to matrix non-commutativity. E.g. for $t = T-2$, this is possible if $\|D_{T-1}W^\top D_{T-2}W^\top\| = \|D_{T-1}W^\top D_{T-2}\| \geq 1$. Unless, $D_{T-1} = D_{T-2}$ is identity, $D_{T-1}WD_{T-2}$ is a submatrix of W, and generically has norm less than one.

C.4 Relationship to existing Recurrent architectures.

We will now briefly discuss other recurrent architectures in the literature to gain intuition into our framework. We will refer to the ODE Eq. 4.5

(a) Vanilla RNNs: Setting $\beta = 1$, $\mathbf{A} = -\mathbf{I}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{0}$, results in the ODE, $\dot{\mathbf{z}}(t) = -\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Euler discretization of this ODE with only one step results in Vanilla RNNs.

(b) Fast/Antisymmetric RNNs: Setting $\beta = 1$, $\mathbf{A} = \mathbf{0}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{0}$, results in the ODE, $\dot{\mathbf{z}}(t) = \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Euler discretization of this ODE with only one step results in (Kusupati et al., 2018; Chang et al., 2019).

(c) Incremental RNNs: Setting $\beta = 1$, $\mathbf{A} = -\mathbf{I}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{I}$, results in the ODE, $\dot{\mathbf{z}}(t) = -\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Since the initial state of the ODE, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$, we can write it into $\dot{\mathbf{z}}(t) = -(\mathbf{z}(t) - \mathbf{s}_{m-1}) + \phi(\mathbf{U}(\mathbf{z}(t) - \mathbf{s}_{m-1}) + \mathbf{W}\mathbf{x}_m)$ with $\mathbf{z}(t_0) = 0$. This ODE is equivalent to (Kag et al., 2020).

C.5 Additional plots for Toy Example.

We add additional figures for the toy example in order to describe the following properties: (a) TARNN achieves faster convergence than the baselines, (b) TARNN time constants activate at the correct locations where the markers are placed and hence we get the hidden state transitions at these locations, and finally (c) we plot a the hidden state norms in order to demonstrate that SkipLSTM does focus at the input markers while TARNN ends up changing the hidden states at these locations.

C.6 Toy Example with larger state space.

In our main experiments with Toy example we have used a very small state space to demonstrate that TARNN outperforms the baselines even with such small state space. Note that forcing training and inference on a small state dimension leads to a difficult problem. This is because in our setup we have 16-length real-valued input traces. As such a large state dimension could in principle commit the entire input trace to memory, and good performance would not be surprising (indeed as Table C.2 shows). The purpose of our example was to motivate our key intuition (a) that trainability of an RNN is limited by vanishing/exploding gradients; and (b) Sequential data consists of uninformative/noisy data segments, which if not suppressed can lead to performance degradation. See Caption of Fig 4·1 main text. For good performance we need both lossless hidden-state gradients (for informative input segments), and

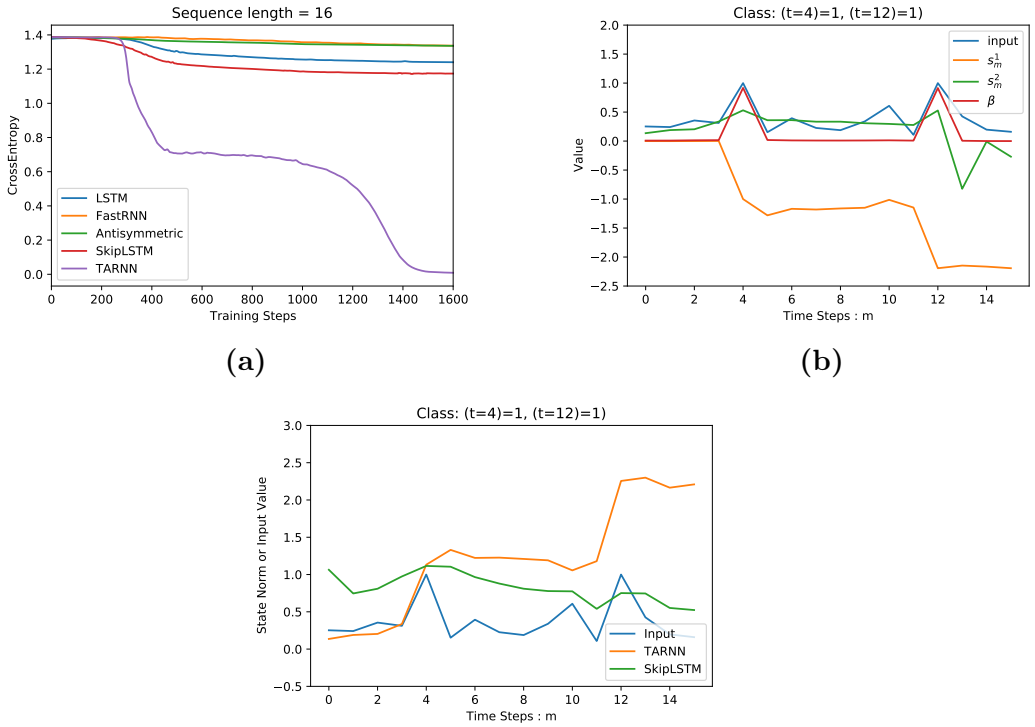


Figure C·1: Toy Example. (a) TARNN converges quickly to the 0.0 cross-entropy error. (b) shows time constant β along with the input, at locations $t = 4, 12$, both the input and time constants are in sync resulting in the state update while everywhere else the time constant does not allow the state to update (see s_m^1 state which captures the update or skip state part). (c) shows the norm of the hidden state for SkipLSTM and TARNN.

skipping (uninformative inputs).

C.7 Gradient Norm Plot for Add-Task.

TARNN works exactly as in the toy example, on other datasets as well. As evidence, we plot gradient norm for the add-task in Figure C·2, and as expected TARNN is able to better maintain gradient norms near unity. The plots for other datasets follow a similar trend.

Table C.2: Toy Example: Accuracy for various hidden state sizes.

Algorithm	Hidden Dimension					
	2	4	8	16	32	64
FastRNN	45	47	52	69	82	96
Antisymmetric	37	39	41	59	73	90
LSTM	45	54	67	82	96	100
SkipLSTM	60	66	72	91	98	100
TARNN	100	100	100	100	100	100

C.8 Google-30, HAR-2 datasets

In order to verify that our method works well for IoT tasks, we use popular datasets from previous works ((Kusupati et al., 2018)). These datasets primarily focus on detecting activitiy embedded in a longer sequence. We pick two datasets namely: (a) HAR-2 (Anguita et al., 2012), *i.e.* Human Activity Recognition from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone, and (b) Google-30 (Warden, 2018), *i.e.* detection of utterances of 30 commands plus background noise and silence. For these tasks, light footprint of the model also becomes extremely important given that these models are deployed on resource constrained IoT devices.

Table C.3 shows accuracy, model size, training time, inference time, and the number of parameters. TARNN beats the baselines in terms of test accuracy. TARNN has smaller model size, while its inference time comparable to iRNN and hence well suited for IoT tasks.

Table C.3: Results for Activity Recoginition (IoT) Datasets.

Data set	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)	Test Time (ms)	#Params
HAR-2	FastRNN	94.50	29	0.063	**0.01**	7.5k
	LSTM	93.65	74	0.183	0.04	16k
	Antisymmetric	93.15	29	0.087	**0.01**	7.5k
	iRNN	96.30	18	**0.018**	0.03	4k
	TARNN	**96.59**	**17**	0.03	0.02	**3.7k**
Google-30	FastRNN	91.60	96	1.30	**0.01**	18k
	LSTM	90.31	219	2.63	0.05	41k
	Antisymmetric	90.91	64	0.54	**0.01**	12k
	iRNN	94.23	45	0.44	0.05	**8.5k**
	TARNN	**94.93**	**20**	**0.38**	0.01	9k

C.9 Inference time

As the table C.3 shows that the inference time for TARNN is similar to FastRNN and about at least one-half of the inference time for the LSTMs.

C.10 Impact of larger K on the results

Our choice of K is inspired by previous ODE discretization works (Kag et al., 2020). Small K suffices for many datasets because inputs are slowly varying (small drift). Furthermore, our dynamical system is exponentially stable (Proposition 2), allowing for rapid convergence to equilibrium. As shown in Table D.2, larger values of K lead to increased inference time, and there is an inherent tradeoff between seeking exact equilibria and inference time. We will elaborate this in the revision.

Table C.4: PTB Language Modeling: Larger K values.

Algorithm	Hidden Dimension	K	Test Perplexity	Train Time (min)	Inference Time (ms)
TARNN	128	1	104.15	27	1
TARNN	128	3	102.42	40	1.7
TARNN	128	5	101.21	65	3.2
TARNN	128	7	101.01	91	5.3
TARNN	128	10	100.91	123	8.1

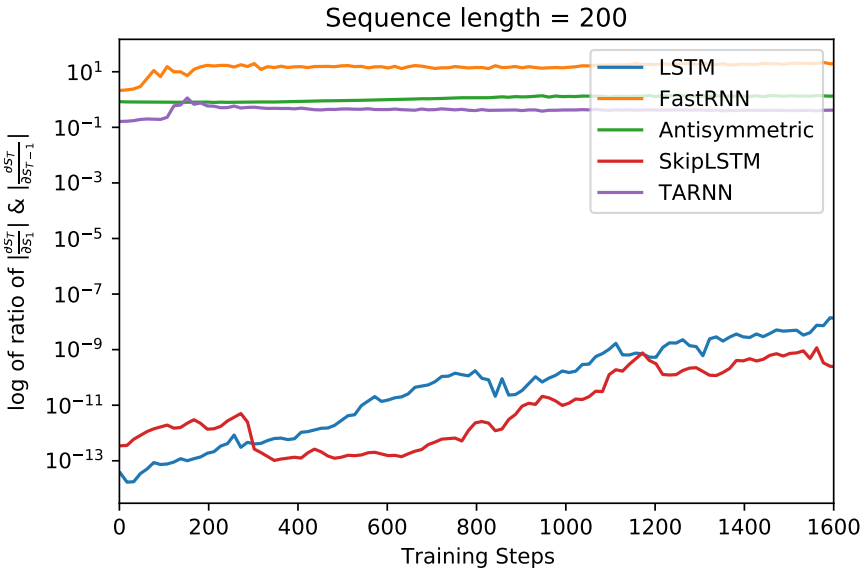


Figure C·2: Add Task Gradient Norm for 200 length sequences.

Appendix D

Appendix to FPTT

D.1 Experiment Details.

We refer the reader to Appendix A.2 for dataset details. Below we describe the details of the various experiments along with hyper-parameter tuning. We use a mixture of language modelling tasks (corresponds to sequence modelling regime) and terminal prediction tasks (corresponds to long range dependency tasks). Language modelling tasks include variants of the popular Penn Tree Bank (McAuley and Leskovec, 2013) dataset. Terminal prediction tasks include the synthetic Add-Task along with Sequential vision tasks. These together corresponds to the long range dependency datasets.

Add-Task A.2.10. For both the algorithms (BPTT and FPTT), we use episodic training where a train batch size of 128 is presented to the RNN to update its parameters and evaluated using an independently drawn test set.

Our main text uses four different sequence lengths : (a) $T = 200$ used in the ablative experiment (see Figure 5·2), (b) $T = 500$ used in the Toy example (see Figure 5·1), (c) the difficult sequence lengths $T = 750$ and $T = 1000$ shown in Figure 5·3. As used in earlier works (Kag et al., 2020), our experiments set hidden state size as 128. Our architecture is one-layer LSTM followed by one classifier layer. We use $1e-3$ as the learning rate for this experiment. We have used Adam optimizer for this task. For smaller sequence lengths $T = 200, 500$, we run the experiment for 5000 training iterations while for the larger sequence lengths $T = 750, 1000$ we run

the experiments for 10000 training iterations. We use $K = 10$ in this task for the proposed algorithm and the default $\alpha = 0.01$ was used in this experiment.

Permute-Pixel MNIST and Pixel-CIFAR-10 A.2.3. Neural network used for this task is one-layer LSTM followed by a classifier layer. Our LSTMs use the hidden state size as 128 and are trained for 200 epochs. We set aside 10% training data for hyper-parameter tuning and once the hyper-parameters are fixed, we use the full data for training and report the performance on the test set. We use the the grid with learning rate choices : $\{0.01, 0.005, 0.001, 0.0001, 0.0005\}$, batch sizes $\{B = 64, 128, 256\}$ and for FPTT we have α choices $\{0.1, 0.5, 0.05, 0.01, 0.001\}$. For the proposed algorithm we use $K = 20$ for this experiment. Note that we use the learning rate schedule defined in (Bai et al., 2019b), i.e. we decay the learning rate by a factor of $\frac{1}{2}$ at fixed intervals, i.e. $\{60, 90, 120\}$ epochs. Our ablative experiment which use the CIFAR-10 dataset keep the same experimental setup described here. The corresponding hyper-parameter tuning details are same as the main experiment.

PTB-300 A.2.4. We used the known small configuration to setup this task for PTB word level task except that sequence length has been changed to 300 to model long range dependencies. We only use $1-$layer LSTM with hidden state size of 256. The embedding dimension has been set to the hidden state in accordance to earlier works (Kusupati et al., 2018; Zhang et al., 2018a). We train both BPTT and FPTT to 100 epochs and learning rate is decayed by half at every epoch where validation perplexity plateaus. We use SGD optimizer as per the configuration and use initial learning rate of 20. We use $K = 10$ for FPTT. Note that α values are tuned using the grid search with values $\{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$.

PTB-w A.2.5. We follow (Merity et al., 2018) to setup our experiments for BPTT and FPTT. We use three layer LSTM model for this task with embedding dimensions 280. As recommended our hidden states for three layers are 1150. All the

other hyper-parameter settings (learning rate, batch size, dropout, other regularizers, dynamic evaluation parameters, etc.) have been borrowed from (Yang et al., 2018) and they are ideal for the BPTT LSTM experiment since this has been tuned by previous works. We report the results with dynamic evaluation(Krause et al., 2018) on the trained model. We use the same architecture and training setup to train LSTMs with both BPTT and FPTT. We use $K = 10$ in our experiments and tune the α values in the grid $\{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ using the validation dataset.

PTB-c A.2.6. We utilize (Merity et al., 2018) to setup the character level task. We use 3-layer LSTM models as recommended with hidden size 1000 and embedding dimension 200. Remaining hyper-parameters have been kept as it is and they are well tuned for BPTT training as per previous work. We train this model with both BPTT and FPTT with the same setting. We perform similar parameter tuning as in the PTB-w experiment to find α values and set $K = 10$ for this dataset.

D.2 Training Time Comparison.

In this section, we demonstrate that training recurrent neural networks with the proposed method is not computationally expensive than back-propagation through time. Table D.1 shows the training time in minutes as measured by the `time` utility in the python language. Note that this refers to the wall-clock time and we ensure that the system is running only one experiment during this process in order to perform fair comparison. As evident from the table, the proposed method fares well in training time when compared to BPTT.

Table D.1: Training time comparison (reported in hours).

Dataset	BPTT	FPTT
PTB-300	1.01	1.12
Pixel-MNIST	3.29	2.68
Permute-MNIST	3.41	2.97
Pixel-CIFAR-10	3.57	3.25
Add-Task	0.31	0.18

D.3 Impact of α hyper-parameter.

We explore the sensitivity to the α hyper-parameter in the Algorithm 3. We use the PTB-300 language modelling dataset. In this experiment we train FPTT on the following α values: $\{1.0, 0.8, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. Table D.2 lists the validation accuracy for these α values. It shows that FPTT is insensitive to $\alpha \leq 0.5$. Note that very small value of α, i.e. $\alpha \to 0$ would lead FPTT to ignore the regularizer and would only optimize the instantaneous loss at every step resulting in diverging iterates. While very high value of α would lead FPTT to only optimize the regularizer and hence very poor generalization performance.

Table D.2: PTB-300 language modelling (validation perplexity) : various α values.

α	Validation Perplexity
1.0	265
0.8	115
0.5	110
0.1	112
0.05	116
0.01	112
0.005	113
0.001	114

D.4 Comparison with Online Gradient Descent.

Our parameter update equations perform gradient descent with dynamic regularizer in order to constraint the iterates to be nearby. In this experiment we show that such an additional regularizer is required to achieve better generalization error. We train LSTMs on the PTB-300 dataset with following training algorithms : (a) BPTT : the

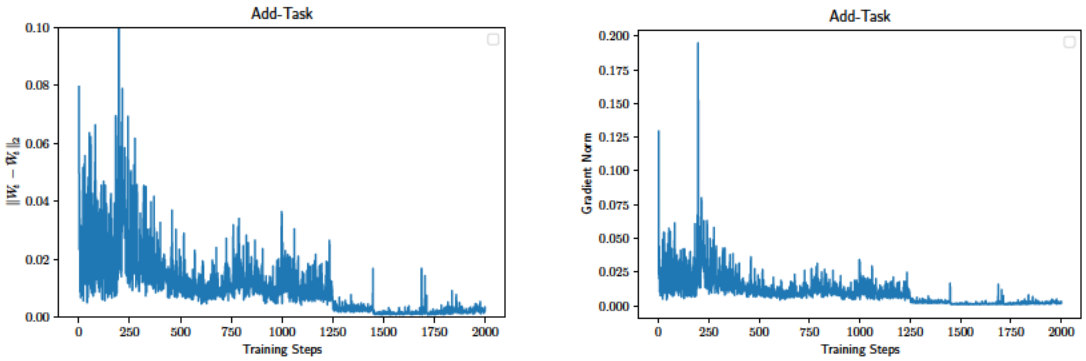
standard back-propagation through time method, (b) FPTT (OGD) : the proposed algorithm without incorporating the dynamic regularizer, (c) FPTT : the proposed algorithm with dynamic regularizer. Table D.3 shows results for these three settings along with the known baselines for this dataset. It can be inferred that incorporating the dynamic regularizer improves the performance of the proposed algorithm.

Table D.3: Ablative results for PTB word level language modelling : Sequence length (300), 1-Layer LSTM. Comparing the FPTT scheme with and without the dynamic regularizer.

Dataset	**PTB-w**	
	Perplexity	**#Params**
FastGRNN (Kusupati et al., 2018)	116.11	53K
IncrementalRNN (Kag et al., 2020)	**115.71**	30K
SpectralRNN (Zhang et al., 2018a)	130.20	31K
LSTM (Zhang et al., 2018a)	130.21	64K
LSTM (Kusupati et al., 2018)	**117.41**	210K
LSTM	117.09	210K
FPTT (OGD) LSTM	113.74	210K
FPTT LSTM	**106.27**	210K

D.5 Convergence $W_t - \bar{W}_t$.

In order to show that the proposed algorithm converges to a single average iterate towards the end of the training phase, we show the convergence plot for $\|W_t - \bar{W}_t\|_2$ and the gradient norm for the training steps. Figure D·1 shows these plots for the Add-Task with sequence length $T = 200$. As evident from these plots, the gap between W_t and $\bar{W}_t$ closes as the training progresses. During the initial training steps, one can observe that there's a significant difference between W_t and $\bar{W}_t$ and as the training progresses this difference goes to 0. Similarly, the gradient norm reaches 0 by the end of the training phase.



(a) **(b)**

Figure D·1: Add Task ($T = 200$): Convergence plot for $\|W_t - \bar{W}_t\|_2$ and the gradient norm plot. As expected the parameter iterates W_t start to converge to the average iterate $\bar{W}_t$. Similarly the gradient norms start decays to near 0 as the training progresses.

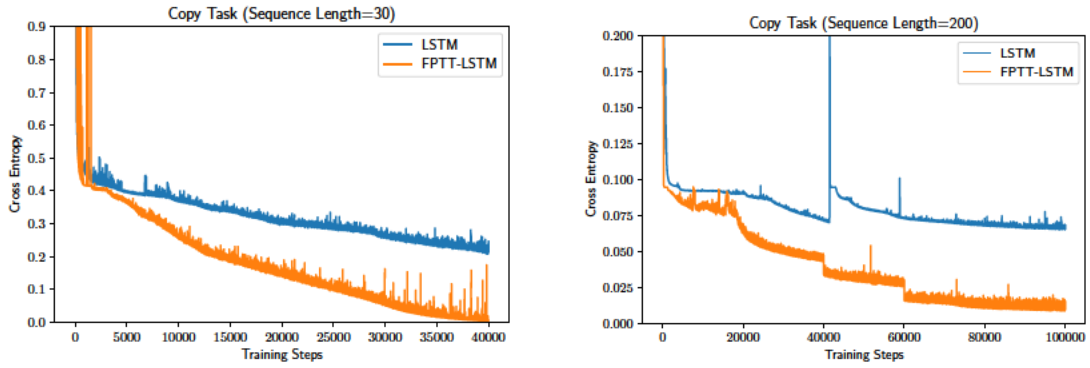
D.6 Copy Task Experiments.

In this section, we perform experiments on the Copy-Task dataset (Arjovsky et al., 2016; Hochreiter and Schmidhuber, 1997b). For a fixed length T, an example input sequence consists of values (x, y). Both x and y are sequences of length $T + 20$. The entries of the sequence are drawn from the alphabet $\{a_i\}_{i=0}^{9}$. The first 10 entries of the sequence x are drawn uniformly, independently and with replacement from $\{a_i\}_{i=0}^{7}$. These first 10 entries are the sequence which need to be remembered and referred to as the magic sequence. The next $T - 1$ entries are set to a_8 representing the blank sequence and needs to be ignored or treated as noise in the sequence. The next entry is a single character a_9 which implies a delimiter in the input sequence and remaining entries are set as a_8. The target label y consists of the character a_8 till the length $T + 10$ and followed by the magic 10 character sequence in the remaining 10 locations. The main motivation behind this task is to remember the magic sequence throughout the length T and as T increases this task becomes harder for the recurrent neural networks as they have to remember the magic sequence through a very large sequence

length. We measure the error in this task using categorical cross-entropy between the label y and the predicted label $\hat{y}$ output by the RNN after going through the input sequence.

Note that there is a memoryless baseline (Arjovsky et al., 2016) for the Copy-Task. It would be to predict a_8 for the sequence length $T+10$ and then predict each of the final 10 characters from the set $\{a+i\}_{i=0}^{7}$ independently and uniformly at random. This results in the cross-entropy value of $\frac{10 \log 8}{T+20}$.

In our experiment we use $1-$layer LSTM with hidden state size 128 and batch size 100. We use Adam as the optimizer and $2e-4$ as the learning rate. We provide an independent training set at each step and evaluate the performance on an independent test set of the same batch size. Figures D·2 shows the convergence plot for the training LSTMs with BPTT and FPTT on sequence length $T=30$ and $T=200$. It can be seen that LSTMs trained with BPTT stay close to the baseline solution for quite a while and then start to decay. While FPTT-LSTMs reach the memoryless baseline quickly and reach the convergence much faster than BPTT.



(a) **(b)**

Figure D·2: Convergence plot Copy Task for Sequence Lengths (a) $T=30$: naive strategy results in 0.39 as the solution, and (b) $T=200$: naive strategy results in 0.09 as the solution.

D.7 Proofs

Proposition 8. *In the Algorithm 3, suppose, the sequence W_t is bounded and converges to a limit point W_∞. Further assume the loss function ℓ_t is $\beta-$Smooth and $\gamma-$Lipschitz. Let the cumulative loss be $F = \frac{1}{T}\sum_{t=1}^{T} \nabla \ell_t(W_\infty)$ after T iterations* [1]*. It follows that W_∞ is a stationary point of Eq. 2.3, i.e., $\lim_{L\to\infty} \frac{\partial F}{\partial W}(W_\infty) = 0$.*

Rewriting the first equation in Eq. 5.4 as:

$$W_{t+1} = \bar{W}_t + \frac{1}{\alpha}(\nabla \ell_{t-1}(W_t) - \nabla \ell_t(W_{t+1})) \tag{D.1}$$

We assumed that the sequence W_t is bounded and converges to a limit point W_∞. By Cesaro mean[2] argument

$$W_{t+1} \to W_\infty \implies \frac{1}{T}\sum_{t=1}^{T} W_{t+1} \overset{T\to\infty}{\longrightarrow} W_\infty \tag{D.2}$$

Summing Eq. D.1 over t

$$\begin{aligned} \frac{1}{T}\sum_{t=1}^{T} W_{t+1} &= \frac{1}{T}\sum_{t=1}^{T} \bar{W}_t + \frac{1}{T}\sum_{t=1}^{T} \frac{1}{\alpha}(\nabla \ell_{t-1}(W_t) - \nabla \ell_t(W_{t+1})) \\ \implies \frac{1}{T}\sum_{t=1}^{T} W_{t+1} &= \frac{1}{T}\sum_{t=1}^{T} \bar{W}_t - \frac{1}{\alpha T}\nabla \ell_T(W_{T+1}) \qquad \text{Telescoping Sum} \end{aligned}$$

Since loss functions are $\gamma-$Lipschitz, we get that $\nabla \ell_T(W_{T+1})$ term is bounded, i.e. $\|\nabla \ell_T(W_{T+1})\| \leq \gamma$. Hence, $\frac{1}{T}\nabla \ell_T(W_{T+1}) \overset{T\to\infty}{\longrightarrow} 0$.

$$\implies \frac{1}{T}\sum_{t=1}^{T} \bar{W}_t \overset{T\to\infty}{\longrightarrow} W_\infty \tag{D.3}$$

[1] For simplicity in exposition, we concatenate all the losses ℓ_t into a single online stream and get rid of the index N, that gets repeated to provide T iterations of the gradient updates

[2] https://www.ee.columbia.edu/~vittorio/CesaroMeans.pdf

Summing second equation in Eq. 5.4 over t, we get

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\bar{W}_{t+1} &= \frac{1}{2T}\sum_{t=1}^{T}(\bar{W}_t + W_{t+1}) - \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) \\
\implies \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) &= \frac{1}{2T}\sum_{t=1}^{T}(\bar{W}_t + W_{t+1}) - \frac{1}{T}\sum_{t=1}^{T}\bar{W}_{t+1} \\
&= \frac{1}{2T}\sum_{t=1}^{T}W_{t+1} + \frac{1}{2T}\sum_{t=1}^{T}\bar{W}_t - \frac{1}{T}\sum_{t=1}^{T}\bar{W}_{t+1} \\
&= \frac{1}{2T}\sum_{t=1}^{T}W_{t+1} - \frac{1}{2T}\sum_{t=1}^{T}\bar{W}_t + \frac{1}{T}(\bar{W}_1 - \bar{W}_{T+1}) \\
\implies \lim_{T\to\infty}\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) &= \lim_{T\to\infty}\Big(\frac{1}{2T}\sum_{t=1}^{T}W_{t+1} - \frac{1}{2T}\sum_{t=1}^{T}\bar{W}_t + \frac{1}{T}(\bar{W}_1 - \bar{W}_{T+1})\Big)
\end{aligned}$$

Note that the quantity $(\bar{W}_1 - \bar{W}_{T+1})$ is finite. Hence, $\lim_{T\to\infty}\frac{1}{T}((\bar{W}_1 - \bar{W}_{T+1})) = 0$. From the Eq. D.2 and Eq. D.3, the other two terms have limits that exists. Plugging in these values in the above expression we get,

$$\implies \lim_{T\to\infty}\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) = \frac{1}{2}W_\infty - \frac{1}{2}W_\infty + 0 = 0 \tag{D.4}$$

From $\beta-$smoothness assumption on the loss function, we have that

$$\begin{aligned}
&\implies \|\nabla\ell_t(W_{t+1}) - \nabla\ell_t(W_\infty)\| \le \beta\|W_{t+1} - W_\infty\| \\
\implies \|\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) - \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_\infty)\| &\le \frac{\beta}{2\alpha T}\sum_{t=1}^{T}\|W_{t+1} - W_\infty\|
\end{aligned}$$

Note $W_{t+1} \to W_\infty \implies \forall \delta > 0, \exists t_\delta$ s.t. $\forall t > t_\delta, \|W_{t+1} - W_\infty\| < \delta$

$$\implies \|\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) - \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_\infty)\| \leq \frac{\beta}{2\alpha T}(T - t_\delta)\delta + \frac{\beta}{2\alpha T}\sum_{t=1}^{t_\delta}\|W_{t+1} - W_\infty\|$$

As $T \to \infty$, both terms on the right hand becomes arbitrarily close to 0, i.e. we have sequence of vectors $\{\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) - \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_\infty)\}_{T=1}^{\infty}$ that converge to the 0 vector.

$$\implies \lim_{T\to\infty}\|\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) - \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_\infty)\| = 0$$

From Eq. D.4, we know that $\frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_{t+1}) \overset{T\to\infty}{\longrightarrow} 0$

$$\implies \frac{1}{2\alpha T}\sum_{t=1}^{T}\nabla\ell_t(W_\infty) \overset{T\to\infty}{\longrightarrow} 0$$

This is the proposed stationarity condition, and our claim follows.

Appendix E

Appendix to Global Layered CNNs

E.1 Experiments with other PDEs.

In this section, we explore few additional PDEs, namely with first order and Laplace differential operators. Our objective is to demonstrate that the Global layer can be naturally extended to any other PDE that is amenable to an iterative solver, thus yielding an efficient update equation.

$$\text{First Order} \quad \frac{\partial}{\partial x}H(x,y) + \frac{\partial}{\partial y}H(x,y) = f(I(x,y)) \tag{E.1}$$

$$\text{Second Order} \quad \frac{\partial^2}{\partial x^2}H(x,y) + \frac{\partial^2}{\partial y^2}H(x,y) = f(I(x,y)) \tag{E.2}$$

Similar to advection-diffusion PDE, these PDEs can be discretized by replacing the differential operators with finite differences. Yielding the following update equations. For simplicity, we take $\delta_x = \delta_y = \delta$,

$$\frac{H(x,y) - H(x-\delta,y)}{\delta} + \frac{H(x,y) - H(x,y-\delta)}{\delta} = f(I(x,y))$$

$$\implies H(x,y) = \frac{1}{2}\Big[H(x-\delta,y) + H(x,y-\delta) + \delta * f(I(x,y))\Big] \tag{E.3}$$

Similarly, the Laplace operator yields the following update

$$\frac{H(x+\delta,y) + H(x-\delta,y) - 2H(x,y)}{\delta^2} + \frac{H(x,y+\delta) + H(x,y-\delta) - 2H(x,y)}{\delta^2} = f(I(x,y))$$

$$\implies H(x,y) = \frac{1}{4}\Big[H(x+\delta,y)+H(x-\delta,y)+H(x,y+\delta)+H(x,y-\delta)-\delta^2 * f(I(x,y))\Big] \tag{E.4}$$

Table E.1 shows the Global layer performance when we use the first order or second order differential operators. It also shows the performance of the update equation when the training and inference use different number of updates (i.e., the hyper-parameter K). For example, while training with large K yields good performance, during inference we can trade-off some accuracy for less number of updates, yielding faster inference.

Table E.1: Ablative Experiments on CIFAR-10 : Training with different iterative steps in the solver and inference with varying steps.

Training			**Accuracy with Inference@K**			
PDE	**Initial Guess**		**K=1**	**K=5**	**K=7**	**K=10**
(First Order)	1-conv	**K=1**	90.71	75.44	61.87	43.02
		K=5	19.2	92.2	90.97	86.75
		K=7	10.01	88.41	92.23	90.19
		K=10	13.52	72.84	89.7	92.47
(First Order)	2-conv	**K=1**	93.17	72.15	62.24	28.56
		K=5	15.26	93.86	93.03	89.4
		K=7	23.21	92.16	93.87	92.79
		K=10	19.87	84.86	92.45	93.89
(Second Order)	1-conv	**K=1**	91.05	72.62	66.57	59.5
		K=5	12.13	91.84	90.65	87.45
		K=7	16.95	89.38	92.34	91.51
		K=10	10.51	65.98	89.13	92.18
(Second Order)	2-conv	**K=1**	93.08	73.34	64.71	55.49
		K=5	21.9	93.55	92.76	90.66
		K=7	17.66	90.14	93.54	93.25
		K=10	18.15	69.72	90.27	94.01

E.2 Discretizing the Diffusion PDE.

In this section, we discretize the following Advection-Diffusion PDE.

$$\frac{\partial}{\partial t}H(x,y,t) + \frac{\partial}{\partial x}\big(u(x,y,t)H(x,y,t)\big) + \frac{\partial}{\partial y}\big(v(x,y,t)H(x,y,t)\big)$$

$$= \frac{\partial}{\partial x}\Big(D_x \frac{\partial}{\partial x} H(x,y,t)\Big) + \frac{\partial}{\partial y}\Big(D_y \frac{\partial}{\partial y} H(x,y,t)\Big) + f(I(x,y)) \tag{E.5}$$

Assume the discrete steps for x, y and t by δ_x, δ_y and δ_t respectively. Following (Hutomo et al., 2019), we replace the partial differential operators with their finite difference, resulting in

$$\frac{H^t_{x,y} - H^{t-1}_{x,y}}{2\delta_t} + \Big(\frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x} H^t_{x,y} + u\frac{H^t_{x+1,y} - H^t_{x-1,y}}{2\delta_x}\Big) + \Big(\frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y} H^t_{x,y} + u\frac{H^t_{x,y+1} - H^t_{x,y-1}}{2\delta_y}\Big)$$

$$= D_x \frac{H^t_{x+1,y} - H^{t+1}_{x,y} - H^{t-1}_{x,y} + H^t_{x-1,y}}{\delta_x^2} + D_y \frac{H^t_{x,y+1} - H^{t+1}_{x,y} - H^{t-1}_{x,y} + H^t_{x,y-1}}{\delta_y^2} + f(I(x,y))$$

Thus, re-arranging the above equation, we obtain

$$\begin{aligned} LH^{k+1}_{x,y} &= (1 - 2B_x - 2B_y)H^{k-1}_{x,y} - 2EH^k_{x,y} + 2\delta_t f(I(x,y)) \\ &+ (-A_x + 2B_x)H^k_{x+1,y} + (A_x + 2B_x)H^k_{x-1,y} \\ &+ (-A_y + 2B_y)H^k_{x,y+1} + (A_y + 2B_y)H^k_{x,y-1} \end{aligned} \tag{E.6}$$

where $L = (1 + 2B_x + 2B_y)$, and

$$u_x = \frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x}; v_y = \frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y}; E = (u_x + v_y)\delta_t$$

$$A_x = \frac{u\delta_t}{\delta_x}; A_y = \frac{v\delta_t}{\delta_y}; B_x = \frac{D_x\delta_t}{\delta_x^2}; B_y = \frac{D_y\delta_t}{\delta_y^2};$$

E.3 MNIST Experiments.

Architecture details.

- *Resnet baseline from Neural ODEs (Chen et al., 2018) and NeuPDE(Sun et al., 2020).* This network begins with a full convolutional layer with (64 output

channels, 3×3 kernel, stride 1). It is followed by the batch norm and ReLU non-linearity. This is followed by another full convolutional layer (64 output channels, 3×3 kernel, stride 2), batch norm and ReLU non-linearity and full convolutional layer (64 output channels, 3×3 kernel, stride 2. Finally, there are 6 basic residual blocks (each consisting of 2 conv layers), followed by average pooling and the classifier layer.

- *Neural ODE architecture.* This network follows the exact same early layers as the above Resnet architecture except the residual layers are now replaced with one ODE layer. Thus, yielding gains in the parameter storage, but the computational footprint goes up, as the number of ODE updates are typically much larger than 6.

- *NeuPDE architecture.* This architecture replaces the residual block with 6 finite-difference layers introduced in (Sun et al., 2020).

- *Resnet-Global (136K params).* We use the same Resnet architecture described above and replace the 6 residual blocks with one Global layer discussed in Sec. 7.2.2. We use constant diffusion coefficients ($D_x = D_y = 1$) and use identity as the function f. We set the velocity (u, v) as the depthwise convolutional layers followed by batch norm and ReLU non-linearity.

- *Resnet (33K params).* This network begins with a full convolutional layer with (16 output channels, 3×3 kernel, stride 1). It is followed by the batch norm and ReLU non-linearity. This is followed by 5 basic residual blocks (each consisting of 2 conv layers), followed by average pooling and the classifier layer.

- *Resnet-Global (10K params).* This networks uses the earlier Resnet architecture ($33K$ params) and replaces the residual layers with one Global layer.

Hyper-parameter details. For all the MNIST experiments, we minimize the cross-entropy loss on the training data using the SGD optimizer with learning rate 0.1 and momentum 0.9. We use weight decay of $1e-4$ and batch size of 128. Following (Sun et al., 2020; Chen et al., 2018), we train the model for 160 epochs with learning rate decay by $\frac{1}{10}$ after $60, 100$ and 140 epochs.

E.4 CIFAR Experiments.

Architecture details.

- *Resnet32, Resnet56.* These architectures come from the initial Residual Networks paper (He et al., 2016). First, it has a full convolution layer (16 output channels, 3×3 kernel, stride 1), followed by basic residual blocks (16 output channels) repeated m times. This is followed by a downsampling convolutional layer (16 output channels, 3×3 kernel, stride 2). This process is repeated again, i.e. m basic residual blocks followed by downsampling layer but with 32 channels, and finally one last time m basic residual blocks with 64 channels. Remaining network consists of average pooling followed by fully connected layer. We obtain Resnet32 with $m = 5$ and Resnet56 with $m = 6$.

- *Resnet-Global ($m = 1, 2$).* We get the global variants of the above described Resnet architectures by replacing the repeated basic residual blocks with one Global layer. We get the Resnet-Global($m = 1$) by keeping only one Residual block as the initialization of the PDE and Resnet-Global ($m = 2$) by keeping two Residual blocks.

- *WideResnet.* This architecture is similar to the Resnet32 and Resnet56, except two changes, i.e. $m = 6$ and the width of the feature maps is multiplied with

4. Thus, feature sizes grow as $(64, 128, 256)$ in the network. This is commonly referred to as the WRN-40-4 (Zagoruyko and Komodakis, 2016).

- *WideResnet-Global.* Similar to the Global variants of the Resnet32 and Resnet56, we create the WideResnet-Global by replacing the repitions in the WideResnet with a Global layer.

- *Densenet.* We used the Densenet-BC variant from the Densenet paper (Huang et al., 2017) for its cost efficiency. We refer the reader to the original paper for detailed architecture setup. Here we provide minimal details to replicate this architecture Densenet-BC ($k = 12, L = 100$). We use the growth-rate 12 and 3 dense blocks with 16 dense layers. This network begins with a full convolution (16 output channels, 3×3 kernel, stride 1). This is followed by a dense block consisting of 16 dense layers, and a transition layer (batch-norm, ReLU, full conv (same output channels, 1×1 kernel, stride 1), then average pooling to down sample the feature map). This process is repeated two more times, yielding three dense blocks. Note that a Densenet concatenates all the feature maps within a dense block. Finally, a batch norm is applied on the resulting before forwarding to a classifier.

- *Densenet-Global.* Note that Densenet is not exactly repeating the same layer in each block (since the feature maps are concatenated). In each dense block, there are 16 dense layers and all the feature maps are forwarded as the output of this block. We cut down the size of this network by using only 8 dense layers in each dense block and apply a Global layer followed by the original transition layer. Thus, yielding Densenet-Global architecture.

- *DARTS.* (Liu et al., 2019a) search for architecture cells using their differential architecture search. They found two optimized cells for the CIFAR dataset,

namely, a normal and a reduction cell. A reduction cell transforms the feature map into a downsampled version of the feature map (similar to the operation performed by the transition layer in Densenet or the stride 2 convolution in Resnet). While a normal cell transforms the feature map into another feature map of the same dimension. We refer the user to their implementation (https://github.com/quark0/darts) for the details of these two cells. The final DARTS architecture used in our works is the same in their implementation, that consists of 20 such cells. First is a full convolution with 36 output channels, after which they apply their normal cells and have reduction cells after nearly every 6 normal cells. Followed by the classifier layer.

- *DARTS-Global.* We create the DARTS-Global variant by replacing all but repetitions of the normal cell with two and applying a Global layer at each resolution. We use identity as the function f. We set the velocity (u, v) and diffusion coefficients (D_x, D_y) as the depthwise convolutional layers followed by batch norm and ReLU non-linearity.

- *Budget-Resnet.* We use the same architecture as the Resnet32 but with reduced feature maps, i.e. instead of $(16, 32, 64)$ we have $(6, 8, 8)$ as the feature maps and same repetitions, i.e. $m = 5$. This yields a Resnet model with 13K parameters and nearly 3.4M MAcs.

- *Budget-Resnet-Global.* We use similar architectural setup as the Resnet-Global but we constrain the feature map sizes to be within 3.5M MACs. Instead of using $(16, 32, 64)$, we use $(10, 9, 16)$ as the feature map sizes. Note that this configuration is not a result of any architecture search, but instead a simply random allocation of the feature map to yield a model within similar compute characteristics of the Budget-Resnet model. Since our aim in the budget exper-

iments is to show that with similar compute as the original model, the proposed model achieves much better performance, this configuration suffices. One can resort to architectural search schemes to find the optimized model with best performance, but that is beyond the scope of this work.

- *Budget-WideResnet.* This model is similar as the WideResnet model ($m = 6$ repetitions) except the number of feature maps go from $(64, 128, 256)$ to $(8, 8, 8)$.
- *Budget-WideResnet-Global.* Similar to WideResnet-Global ($m = 1$ repetitions) except the number of channels reduce from $(64, 128, 256)$ to $(12, 16, 12)$.
- *Budget DARTS.* Similar to the DARTS model with number of initial channels reduced from 36 to 3.
- *Budget DARTS-Global.* Similar to the DARTS-Global model with number of initial channels reduced from 36 to 5.

Hyper-parameter details. We follow the recommendations of the related works for various training strategies. We do only use the standard data augmentation discussed in the main text. For all our experiments, we use the SGD optimizer with momentum 0.9. All our models are trained for 300 epochs with pre-defined learning rate decay of $\frac{1}{10}$ at 150 and 225 epochs.

We search of other hyper-parameters (batch size and weight decay) whenever no recommendation is available from the baseline. We use the batch size of 64 for Densenet experiments while the rest of the experiments we use the batch size of 32. We use a weight decay of $1e-4$ for Resnet and Wide-Resnet experiments. While for the Global variants of these architecture we found $5e-5$ to yield best validation performance.

Note that for the DARTS and DARTS-Global experiments, we follow the setup described in (Liu et al., 2019a), it includes cutout data augmentation, auxiliary losses

from each reduced cell blocks with weight 0.4, a path dropout of 0.2 and a cosine learning rate scheduler. For computational purposes, we run both the baseline and Global variant only up to 300 epochs as opposed to the recommended 600 epochs. As per their setup, we use a batch size of 96 and weight decay of $3e-4$. For the DARTS-Global experiments, we use a weight decay of $8e-4$.

E.5 ImageNet Experiments.

Architecture details.

- *MobileNetV2.* We used the MobileNetV2 (Sandler et al., 2018) architecture with width multiplier 1.0 (see Table 2 in MobileNetV2 paper). This network has nearly 3.4M parameters and 300M MACs.

- *MobileNetV2-Global.* We create the Global variant of the MobileNetV2 architecture by replacing the repetitions in all resolutions $(112, 56, 28, 14)$ except the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.

- *MobileNetV2-Global-s (2× less MACs).* We modify the earlier MobileNetV2-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

- *MobileNetV3.* We used the MobileNetV3-Large (Howard et al., 2019) architecture with width multiplier 1.0 (see Table 1 in MobileNetV3 paper). This network has nearly 5.4M parameters and 219M MACs.

- *MobileNetV3-Global.* We create the Global variant of the MobileNetV3-Large architecture by replacing the repetitions in all resolutions $(56, 28, 14)$ except

the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.

- *MobileNetV3-Global-s (2× less MACs).* We modify the earlier MobileNetV3-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

- *EfficientNet-B0.* We used the EfficientNet-B0 (Tan and Le, 2019) architecture with width multiplier 1.0 (see Table 1 in EfficientNet paper). This network has nearly 5.3M parameters and 390M MACs.

- *EfficientNet-B0-Global.* We create the Global variant of the EfficientNet-B0 architecture by replacing the repetitions in all resolutions $(112, 56, 28, 14)$ except the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.

- *EfficientNet-B0-Global-s (2× less MACs).* We modify the earlier EfficientNet-B0-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

Hyper-parameter details. We used RMSProp optimizer with momentum 0.9 for all the ImageNet experiments as per the experimental setup recommended by the baselines (Howard et al., 2019; Tan and Le, 2019). We used weight decay $1e-5$. For both MobileNetv3 and EfficientNet-B0, we used dropout 0.2. We also used an exponential moving average (ema) with the decay of 0.9999. As recommended in the EfficientNet-B0, we use AutoAugment policy along with stochastic depth of 0.8. We used a batch size of 512 and learning rate 0.05, with the learning rate decay of about 0.97 every 2.4 epochs. We train these models for 300 epochs.

Inference and Training time comparison. Table E.2 compares the inference and training time for various ImageNet models used in the main text. Global

architecture shows up to 2× reduction in inference and train time.

Table E.2: ImageNet: Train & Inference times (cost of one pass through train and test dataset on a V100 GPU).

Architecture	Accuracy	Train Time(s)	Inference Time(s)
MobileNetV2	72.0	2181s	91s
MobileNetV2-Global	69.03	1414s	58s
MobileNetV3	75.2	1714s	71s
MobileNetV3-Global	71.89	1090s	45s
EfficientNet-B0	77.1	2667s	111s
EfficientNet-B0-Global	74.53	1311s	54s

E.6 Discussion.

Comments about MDEQ Empirical Evaluations. MDEQ did an unfair comparison with Resnet. They added auxiliary losses at various input resolutions for the deep equilibrium features but fail to add such auxiliary losses to the Resnet training routine. It has been shown in the literature that adding auxiliar losses improves performance in neural networks (Liu et al., 2019a; Kag and Saligrama, 2021b; Trinh et al., 2018) . They should have enabled such auxiliary losses in the Resnet experiments as well for fair comparison.

E.7 Advantages of the Global layer over Neural ODEs and NeuPDE.

- As authors of NeuPDE point out (see Line 2-3 on pg 367 http://proceedings.mlr.press/v107/sun20a/sun20a.pdf), they are unable to scale-up to large-scale datasets such as ImageNet. That Neural ODE is not scalable to ImageNet datasets is well-known. For instance, see MDEQ https://arxiv.org/pdf/2006.08656.pdf Sec. 2, para 3, where this point is discussed. For reference, we repeat their point; even to handle MNIST, with input resolution $28 \times 28 \times 1$, Neural ODE must downsample to 7×7, and that its lack of scalability to realistic

vision tasks arises from numerical instability. In fact, MDEQ while reporting ImageNet results, leaves out Neural ODEs.

- MDEQ, which we compare extensively, can be thought of as the "scalable" version of Neural ODEs. Our reported results are significantly better than MDEQ (see Table 7.3 & 7.5).

- We emphasize that NeuPDE suffers the same scalability since they are really Neural ODEs but with the input replaced by a larger feature input such as monomials of the input, or a collection of partial derivatives of the input (see Eq. 1 or Eq. 9 in http://proceedings.mlr.press/v107/sun20a/sun20a.pdf). This is like substituting linear features with polynomial features in SVMs. Our approach has no connection to NeuPDEs. We learn a PDE and solve it to equilibrium (more in the spirit of DEQ/MDEQ).

E.8 Illustrative Example Visualizations.

Similar to the illustration in the method section, we add another representation for the letter 3 for a different input in the Figure E·1. Here also, it is evident that Global layer representation highlights the corners very brightly in contrast to the representation from other feature backbones.

We show the velocity vectors associated with this new example in the Figure E·2. One can clearly notice that there is some non-zero activity along the corners of the digit 3.

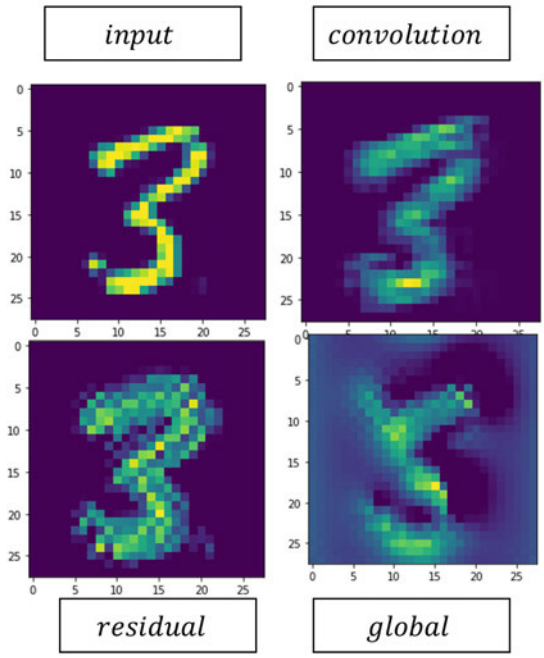


Figure E·1: Another example to demonstrate the visual differences between different representation for the MNIST input letter 3

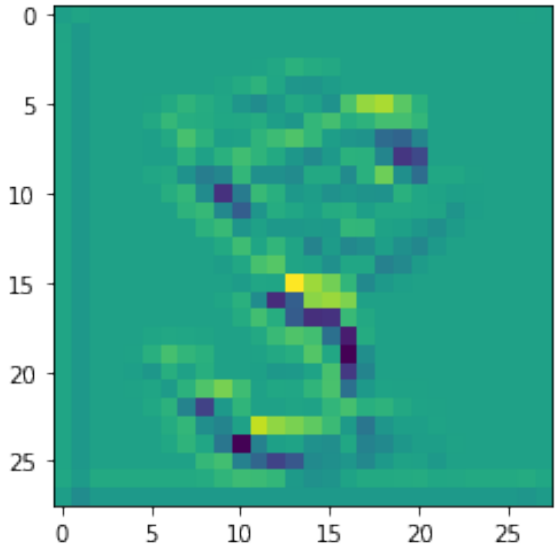


Figure E·2: This represents the velocity vectors associated with the example in Figure E·1. Note that there is some non-zero activity along the corners (represented by the very bright or very dark spots on the edges of the letter 3).

Appendix F

Appendix to SI-CNNs

F.1 Toy Example: CIFAR-10 Dataset

In this section, we provide training setup and architectural details for the toy example discussed in the Sec. 8.1.

Dataset. We use the CIFAR-10 dataset (Krizhevsky and Hinton, 2009). It consists of 32×32 RGB images of 10 classes. It has 50K train and 10K test splits. We follow the standard data augmentation techniques including: (a) random horizontal flip, (b) random crop, (c) auto-augment (Cubuk et al., 2019), (d) cutout (DeVries and Taylor, 2017), and (e) standard mean-variance normalization.

Table F.1: Toy Example Architecture Details.

Stage	Input	Operator	Stride	# In Channels	# Out Channels
0	$32 \times 32 \times 3$	Conv2d, 3×3	1	3	32
1	$30 \times 30 \times 32$	(Spatially Interpolated) Inverted Residual	1	32	16
2	$28 \times 28 \times 16$	Conv2d, 1×1	1	16	32
3	$28 \times 28 \times 32$	Global Pool & Classifier	1	32	10

Models. We create a simple convolutional architecture with the structure defined in Table F.1. We use either inverted residual or its spatially interpolated variant yielding the following two architectures.

1. *ConvNet-IR.* Stage 1 in Table F.1 uses Inverted Residual block with expansion ratio 6.

2. *SI-ConvNet-IR.* Stage 1 in Table F.1 uses Spatially Interpolated Inverted Resid-

ual block with expansion ratio 2 in the cheaper branch and 4 in the anchors branch.

Training Details. We learn both these architectures for 200 epochs using the training set and report the accuracy on the test set. We use a batch size of 256 and minimize the standard cross-entropy loss using SGD optimizer with momentum 0.9 and learning rate of 0.1. We also include the weight decay term with value $1e-5$. We use the cosine schedule for the learning rate decay.

F.2 Classification and Segmentation Architecture Details

F.2.1 Classification Architectures and ImageNet backbones for Segmentation.

We describe our ImageNet classification architectures in this section. First, we describe the mobilenet architectures including MobileNetV3-large, MobileNetV3-small and Multi-HardwareMobileNet model. Table F.2 and Table F.3 describe the mobilenetv3 large and its spatially interpolated variant, SI-MobileNetV3-large.

Note that similar to the baseline architecture we replicate the residual block and other layer details in these tables such as input size, number of input-output channels, stride and the number of expansion channels. Other details such as type of non-linearity and squeeze-excite operator are same as the baselines and we refer the reader to the original paper for explicit details. We will release our implementation in the final version for the model definition as well as the pre-trained model weights for reproduction. Since Spatially-Interpolated variants include additional details such as the anchors and cheaper feature computation, we provide these details in the extra columns namely: anchor resolution (i.e. the input to the anchor feature branch), number of cheaper features (this is the number of expansion channels for the cheaper

feature branch), number of anchor channels (this is the number of expansion channels for the anchor feature branch), and number of cheaper groups (refers to the number of groups in the cheaper feature used for further reduction in the computation).

Table F.4 and Table F.5 provide the architecture definition for the MobileNetV3-small and SI-MobileNetV3-small models. Finally, Table F.6 and Table F.7 lists the architecture definition for the Multi-HardwareMobileNet and SI-Multi-HardwareMobileNet. Note that we borrow the MobileNetV3 (Howard et al., 2019) and Multi-HardwareMobileNet (Chu et al., 2021) architecture definitions from their papers for completeness. Similar to the MobileNetV3 convention, we refer the MobileNetV3-large architecture with width (number of features) multiplied by 0.75 as the MobileNetV3-large architecture.

We borrow the EfficientNet-B0 definition from (Tan and Le, 2019) and show the same in the Table F.8. We create its spatially interpolated variant, SI-EfficientNet-B0 in the Table F.9. Similar to the EfficientNet scaling, we create the B1, B2, B3 variants and the corresponding spatially interpolated variants, SI-B1, SI-B2, SI-B3.

Table F.2: MobileNetV3-Large model.

Input	Operator	Stride	# In Channels	# Out Channels	# Exp Channels
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	16	-
$112 \times 112 \times 16$	Inverted Residual 3×3	1	16	16	16
$112 \times 112 \times 16$	Inverted Residual 3×3	2	16	24	64
$56 \times 56 \times 24$	Inverted Residual 3×3	1	24	24	72
$56 \times 56 \times 24$	Inverted Residual 5×5	2	24	40	72
$28 \times 28 \times 40$	Inverted Residual 5×5	1	40	40	120
$28 \times 28 \times 40$	Inverted Residual 5×5	1	40	40	120
$28 \times 28 \times 40$	Inverted Residual 3×3	2	40	80	240
$14 \times 14 \times 80$	Inverted Residual 3×3	1	80	80	200
$14 \times 14 \times 80$	Inverted Residual 3×3	1	80	80	184
$14 \times 14 \times 80$	Inverted Residual 3×3	1	80	80	184
$14 \times 14 \times 80$	Inverted Residual 3×3	1	80	112	480
$14 \times 14 \times 112$	Inverted Residual 3×3	1	112	112	672
$14 \times 14 \times 112$	Inverted Residual 5×5	2	112	160	672
$7 \times 7 \times 160$	Inverted Residual 5×5	1	160	160	960
$7 \times 7 \times 160$	Inverted Residual 5×5	1	160	160	960
$7 \times 7 \times 160$	Conv2d, 1×1	1	160	960	-
$7 \times 7 \times 960$	Global Pool 7×7	-	960	960	-
$1 \times 1 \times 960$	Conv2d 1×1	1	960	1280	-
$1 \times 1 \times 1280$	Conv2d 1×1	1	1280	1000	-

Table F.3: Spatially Interpolated MobileNetV3-Large model. Legends used in the table: (a) Anchor Channels (Anc. Chan.), (b) Cheaper Channels (Cheap Chan.), (c) Cheaper Groups (Cheap Grp).

Input	Operator	Stride	# In Ch.	# Out Ch.	Anc. Chan.	Anchors Resolution	Cheap Chan.	Cheap Grp.
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	16	-	-	-	-
$112 \times 112 \times 16$	IR 3×3	1	16	16	16	-	-	-
$112 \times 112 \times 16$	SI-IR 3×3	2	16	24	64	$56 \times 56 \times 16$	32	1
$56 \times 56 \times 24$	SI-IR 3×3	1	24	24	72	$28 \times 28 \times 24$	48	2
$56 \times 56 \times 24$	SI-IR 5×5	2	24	40	72	$28 \times 28 \times 24$	36	1
$28 \times 28 \times 40$	SI-IR 5×5	1	40	40	120	$14 \times 14 \times 40$	60	2
$28 \times 28 \times 40$	SI-IR 5×5	1	40	40	120	$14 \times 14 \times 40$	60	2
$28 \times 28 \times 40$	SI-IR 3×3	2	40	80	240	$14 \times 14 \times 40$	240	2
$14 \times 14 \times 80$	SI-IR 3×3	1	80	80	200	$7 \times 7 \times 80$	160	2
$14 \times 14 \times 80$	SI-IR 3×3	1	80	80	184	$7 \times 7 \times 80$	160	2
$14 \times 14 \times 80$	SI-IR 3×3	1	80	80	184	$7 \times 7 \times 80$	128	2
$14 \times 14 \times 80$	SI-IR 3×3	1	80	112	480	$7 \times 7 \times 80$	400	1
$14 \times 14 \times 112$	SI-IR 3×3	1	112	112	672	$7 \times 7 \times 112$	560	2
$14 \times 14 \times 112$	IR 5×5	2	112	160	672	-	-	-
$7 \times 7 \times 160$	IR 5×5	1	160	160	960	-	-	-
$7 \times 7 \times 160$	Conv2d, 1×1	1	160	960	-	-	-	-
$7 \times 7 \times 960$	Pool 7×7	-	960	960	-	-	-	-
$1 \times 1 \times 960$	Conv2d 1×1	1	960	1280	-	-	-	-
$1 \times 1 \times 1280$	Conv2d 1×1	1	1280	1000	-	-	-	-

F.2.2 Segmentation Architectures

For segmentation experiments, we follow the MOSAIC (Wang and Howard, 2021) setup to create the baselines. We replicate their architecture definition with our ImageNet pre-trained backbones as discussed in the previous section. For MobileNetV3 backbones, we create the segmentation ready backbones by halving the number of channels in the last few residual blocks as described in the segmentation evaluations

Table F.4: MobileNetV3-Small model.

Input	Operator	Stride	# In Channels	# Out Channels	# Exp Channels
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	16	-
$112 \times 112 \times 16$	Inverted Residual 3×3	2	16	16	16
$56 \times 56 \times 16$	Inverted Residual 3×3	2	24	24	72
$28 \times 28 \times 24$	Inverted Residual 3×3	1	24	24	88
$28 \times 28 \times 24$	Inverted Residual 5×5	2	24	40	96
$14 \times 14 \times 40$	Inverted Residual 5×5	1	40	40	240
$14 \times 14 \times 40$	Inverted Residual 5×5	1	40	40	240
$14 \times 14 \times 40$	Inverted Residual 5×5	1	40	48	120
$14 \times 14 \times 48$	Inverted Residual 5×5	1	48	48	144
$14 \times 14 \times 48$	Inverted Residual 5×5	2	48	96	288
$7 \times 7 \times 96$	Inverted Residual 5×5	1	96	96	576
$7 \times 7 \times 96$	Inverted Residual 5×5	1	96	96	576
$7 \times 7 \times 96$	Conv2d, 1×1	1	96	576	-
$7 \times 7 \times 576$	Global Pool 7×7	-	576	576	-
$1 \times 1 \times 576$	Conv2d 1×1	1	576	1024	-
$1 \times 1 \times 1024$	Conv2d 1×1	1	1024	1000	-

Table F.5: Spatially Interpolated MobileNetV3-Small model. Legends used in the table: (a) Anchor Channels (Anc. Chan.), (b) Cheaper Channels (Cheap Chan.), (c) Cheaper Groups (Cheap Grp).

Input	Operator	Stride	# In Ch.	# Out Ch.	Anc. Chan.	Anchors Resolution	Cheap Chan.	Cheap Grp.
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	16	-	-	-	-
$112 \times 112 \times 16$	IR 3×3	2	16	16	16	-	-	-
$56 \times 56 \times 16$	IR 3×3	2	24	24	72	$28 \times 28 \times 16$	24	1
$28 \times 28 \times 24$	SI-IR 3×3	1	24	24	88	$14 \times 14 \times 24$	36	1
$28 \times 28 \times 24$	SI-IR 5×5	2	24	40	96	$14 \times 14 \times 24$	48	1
$14 \times 14 \times 40$	SI-IR 5×5	1	40	40	240	$7 \times 7 \times 40$	80	1
$14 \times 14 \times 40$	SI-IR 5×5	1	40	40	240	$7 \times 7 \times 40$	120	1
$14 \times 14 \times 40$	SI-IR 5×5	1	40	48	120	$7 \times 7 \times 40$	60	2
$14 \times 14 \times 48$	SI-IR 5×5	1	48	48	144	$7 \times 7 \times 48$	144	1
$14 \times 14 \times 48$	SI-IR 5×5	2	48	96	288	$7 \times 7 \times 48$	144	2
$7 \times 7 \times 96$	SI-IR 5×5	1	96	96	576	$3 \times 3 \times 96$	576	2
$7 \times 7 \times 96$	SI-IR 5×5	1	96	96	576	$3 \times 3 \times 96$	192	2
$7 \times 7 \times 96$	Conv2d, 1×1	1	96	576	-	-	-	-
$7 \times 7 \times 576$	Pool 7×7	-	576	576	-	-	-	-
$1 \times 1 \times 576$	Conv2d 1×1	1	576	1024	-	-	-	-
$1 \times 1 \times 1024$	Conv2d 1×1	1	1024	1000	-	-	-	-

in (Howard et al., 2019). For the Multi-HardwareMobileNet experiment, we use the pre-trained backbones from the ImageNet experiment. For both these architecture families, after pre-training with ImageNet dataset, we discard the classification head and add the segmentation head as described in (Wang and Howard, 2021). Note that the MOSAIC segmentation decoder utilizes the expanded feature maps in the inverted residual blocks. For MobileNetV3 and Multi-HardwareMobileNet architectures, this corresponds to the expanded feature maps after the depthwise stage. For Spatially Interpolated (SI) variants, we concatenate the Anchors and Cheaper feature branches and forward the same to the decoder. This does not incorporate heavy storage or computational overhead, and is correctly reflected in our storage and compute characteristics in the Table 8.3.

F.3 ImageNet Classification (Training Procedure & Hyper-parameters)

In this section, we describe our training procedure and the hyper-parameters including data augmentations used in learning different architectures. We point out that

Table F.6: Multi-Hardware MobileNet model.

Input	Operator	Stride	# In Channels	# Out Channels	# Exp Channels
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	32	-
$112 \times 112 \times 32$	Inverted Residual 3×3	2	32	32	96
$56 \times 56 \times 32$	Inverted Residual 3×3	1	32	32	64
$56 \times 56 \times 32$	Inverted Residual 5×5	2	32	64	160
$28 \times 28 \times 64$	Inverted Residual 3×3	1	64	64	192
$28 \times 28 \times 64$	Inverted Residual 3×3	1	64	64	128
$28 \times 28 \times 64$	Inverted Residual 3×3	1	64	64	192
$28 \times 28 \times 64$	Inverted Residual 5×5	2	64	128	384
$14 \times 14 \times 128$	Inverted Residual 3×3	1	128	128	384
$14 \times 14 \times 128$	Inverted Residual 3×3	1	128	128	384
$14 \times 14 \times 128$	Inverted Residual 3×3	1	128	128	384
$14 \times 14 \times 128$	Inverted Residual 3×3	1	128	160	768
$14 \times 14 \times 160$	Inverted Residual 3×3	1	160	160	640
$14 \times 14 \times 160$	Inverted Residual 3×3	1	160	192	960
$14 \times 14 \times 192$	Inverted Residual 5×5	1	192	96	384
$14 \times 14 \times 96$	Inverted Residual 5×5	1	96	96	384
$14 \times 14 \times 96$	Inverted Residual 5×5	1	96	96	384
$14 \times 14 \times 96$	Conv2d, 1×1	1	96	480	-
$14 \times 14 \times 480$	Global Pool 14×14	-	480	480	-
$1 \times 1 \times 480$	Conv2d 1×1	1	480	1280	-
$1 \times 1 \times 1280$	Conv2d 1×1	1	1280	1000	-

our ImageNet architecture codebase is built on top of the timm library (Wightman, 2019). We follow their recommended hyper-parameter and data augmentation strategies that yield results similar to the publicly available results (Top-1 accuracy, number of parameters, MACs, etc.) from various baselines. For a fair comparison, we use the same setup for baseline and our spatially interpolated architectures. Below, we list training procedure for various architecture families for the ImageNet dataset. In terms of the data augmentation, as recommended by the previous works (Wightman, 2019; Tan and Le, 2019; Tan and Le, 2021), we use the rand augment strategy along with the random horizontal flips, random crop to 224 size and standard mean-variance normalization. For the segmentation experiments, we pre-train the backbones using the exact same procedure described below. Post pre-training, the segmentation experiments follow the training procedure described in the main text (see Sec. 8.3.3). We use half-precision to train all the architectures (baseline as well as the proposed spatially interpolated architectures).

MobileNet (MobileNetV3, Multi-HardwareMobileNet) and EfficientNet Architectures (EfficientNet-B0-B3, EfficientNetV2-small). For all the

Table F.7: Spatially Interpolated Multi-Hardware MobileNet model. IR stands for Inverted Residual block. Legends used in the table: (a) Anchor Channels (Anc. Chan.), (b) Cheaper Channels (Cheap Chan.), (c) Cheaper Groups (Cheap Grp).

Input	Operator	Stride	# In Ch.	# Out Ch.	Anc. Chan.	Anchors Resolution	Cheap Chan.	Cheap Grp.
$224 \times 224 \times 3$	Conv2d, 3×3	2	3	32	-	-	-	-
$112 \times 112 \times 32$	SI-IR 3×3	2	32	32	96	$56 \times 56 \times 32$	64	1
$56 \times 56 \times 32$	SI-IR 3×3	1	32	32	64	$28 \times 28 \times 32$	64	2
$56 \times 56 \times 32$	SI-IR 5×5	2	32	64	160	$28 \times 28 \times 32$	80	1
$28 \times 28 \times 64$	SI-IR 3×3	1	64	64	192	$14 \times 14 \times 64$	96	2
$28 \times 28 \times 64$	SI-IR 3×3	1	64	64	128	$14 \times 14 \times 64$	96	2
$28 \times 28 \times 64$	SI-IR 3×3	1	64	64	192	$14 \times 14 \times 64$	96	2
$28 \times 28 \times 64$	SI-IR 5×5	2	64	128	384	$14 \times 14 \times 64$	384	1
$14 \times 14 \times 128$	SI-IR 3×3	1	128	128	384	$7 \times 7 \times 128$	96	2
$14 \times 14 \times 128$	SI-IR 3×3	1	128	128	384	$7 \times 7 \times 128$	96	2
$14 \times 14 \times 128$	SI-IR 3×3	1	128	128	384	$7 \times 7 \times 128$	96	2
$14 \times 14 \times 128$	SI-IR 3×3	1	128	160	768	$7 \times 7 \times 128$	640	1
$14 \times 14 \times 160$	SI-IR 3×3	1	160	160	640	$7 \times 7 \times 160$	320	2
$14 \times 14 \times 160$	SI-IR 3×3	1	160	192	960	$7 \times 7 \times 160$	320	2
$14 \times 14 \times 192$	IR 5×5	1	192	96	384	-	-	-
$14 \times 14 \times 96$	IR 5×5	1	96	96	384	-	-	-
$14 \times 14 \times 96$	Conv2d, 1×1	1	96	480	-	-	-	-
$14 \times 14 \times 480$	Pool 14×14	-	480	480	-	-	-	-
$1 \times 1 \times 480$	Conv2d 1×1	1	480	1280	-	-	-	-
$1 \times 1 \times 1280$	Conv2d 1×1	1	1280	1000	-	-	-	-

MobileNet and EfficientNet architectures, we use the following training procedure. We use the RMSProp optimizer with momentum 0.9 and learning rate 0.064 and weight decay of $1e-5$. We use a batch size of 1024. We use 5 linear warm up epochs from $1e-6$ to the learning rate 0.064. Thereafter, we decay the learning rate by 0.97 for every 2.4 epochs. We use a drop-out of 0.2 for MobileNet and EfficientNet-B0-B1 architectures and increase this to 0.3 for EfficientNet-B2-B3 architectures as per the paper recommendations (Tan and Le, 2019). We train the EfficientNet architectures for 450 epochs and train the MobileNet architectures for 600 epochs. These hyper-parameters have been successfully employed by the timm library (Wightman, 2019) to reproduce the original paper results.

Resnet50 Architecture. We minimize the standard cross-entropy loss with SGD optimizer with momentum of 0.9 and learning rate 0.4 for a batch size of 768. We include a weight decay term of $1e-4$ and decay the learning rate with the cosine scheduler for 200 epochs. Note that we include warm-up epochs (10) for the learning rate, starting from $1e-4$.

Table F.8: EfficientNet-B0 model.

Input	Operator	Stride	# In Channels	# Out Channels	# Exp Channels
$224 \times 224 \times 3$	Conv2d, 3×3	1	3	32	-
$112 \times 112 \times 32$	Inverted Residual(Exp=1) 3×3	1	32	16	-
$112 \times 112 \times 16$	Inverted Residual(Exp=6) 3×3	2	16	24	96
$56 \times 56 \times 24$	Inverted Residual(Exp=6) 3×3	1	24	24	144
$56 \times 56 \times 24$	Inverted Residual(Exp=6) 5×5	2	24	40	144
$28 \times 28 \times 40$	Inverted Residual(Exp=6) 5×5	1	40	40	240
$28 \times 28 \times 40$	Inverted Residual(Exp=6) 3×3	2	40	80	240
$14 \times 14 \times 80$	Inverted Residual(Exp=6) 3×3	1	80	80	480
$14 \times 14 \times 80$	Inverted Residual(Exp=6) 3×3	1	80	80	480
$14 \times 14 \times 80$	Inverted Residual(Exp=6) 5×5	1	80	112	480
$14 \times 14 \times 112$	Inverted Residual(Exp=6) 5×5	1	112	112	672
$14 \times 14 \times 112$	Inverted Residual(Exp=6) 5×5	1	112	112	672
$14 \times 14 \times 112$	Inverted Residual(Exp=6) 5×5	2	112	192	672
$7 \times 7 \times 192$	Inverted Residual(Exp=6) 5×5	1	192	192	1152
$7 \times 7 \times 192$	Inverted Residual(Exp=6) 5×5	1	192	192	1152
$7 \times 7 \times 192$	Inverted Residual(Exp=6) 5×5	1	192	192	1152
$7 \times 7 \times 192$	Inverted Residual(Exp=6) 3×3	1	192	320	1152
$7 \times 7 \times 320$	Conv 1×1, Global-Pooling, Classifier	1	320	1280	-

F.4 Other Residual Blocks (Implementation)

Table F.10 shows the EfficientNetV2-small architecture used in the ablations. We create the spatially interpolated variant, SI-EfficientNetV2-small as per the Table F.11. Similarly, we create the interpolated variant of Resnet50 (see Table F.12) in the Table F.13. Note that we borrow these baseline implementations from the publicly available timm library (Wightman, 2019).

F.5 Discussion on Optimal Configuration between anchor and cheaper feature branch

While there are various ways to balance the computation between anchors and cheaper block features, a more computational approach to finding such a configuration requires an expensive architecture search over the number of anchors and cheaper block projection dimensions. In this work, we follow a simple and widely applicable guideline that enables us to achieve substantial computational benefits without any significant loss in performance metrics such as accuracy. As a general rule in spatial interpo-

Table F.9: Spatially Interpolated EfficientNet-B0 model. IR stands for Inverted Residual block. Clf refers to classifier layer. Pool refers to adaptive global pooling. Legends used in the table: (a) Anchor Channels (Anc. Chan.), (b) Cheaper Channels (Cheap Chan.), (c) Cheaper Groups (Cheap Grp).

Input	Operator	Stride	# In Ch.	# Out Ch.	Anc. Chan.	Anchors Resolution	Cheap Chan.	Cheap Grp.
$224 \times 224 \times 3$	Conv2d, 3×3	1	3	32	-	-	-	-
$112 \times 112 \times 32$	IR(Exp=1) 3×3	1	32	16	-	-	-	-
$112 \times 112 \times 16$	IR(Exp=6) 3×3	2	16	24	96	$56 \times 56 \times 16$	48	1
$56 \times 56 \times 24$	SI-IR(Exp=6) 3×3	1	24	24	144	$28 \times 28 \times 24$	48	1
$56 \times 56 \times 24$	SI-IR(Exp=6) 5×5	2	24	40	144	$28 \times 28 \times 24$	72	1
$28 \times 28 \times 40$	SI-IR(Exp=6) 5×5	1	40	40	240	$14 \times 14 \times 40$	80	1
$28 \times 28 \times 40$	SI-IR(Exp=6) 3×3	2	40	80	240	$14 \times 14 \times 40$	160	1
$14 \times 14 \times 80$	SI-IR(Exp=6) 3×3	1	80	80	480	$7 \times 7 \times 80$	320	2
$14 \times 14 \times 80$	SI-IR(Exp=6) 3×3	1	80	80	480	$7 \times 7 \times 80$	320	2
$14 \times 14 \times 80$	SI-IR(Exp=6) 5×5	1	80	112	480	$7 \times 7 \times 80$	320	1
$14 \times 14 \times 112$	SI-IR(Exp=6) 5×5	1	112	112	672	$7 \times 7 \times 112$	448	2
$14 \times 14 \times 112$	SI-IR(Exp=6) 5×5	1	112	112	672	$7 \times 7 \times 112$	448	2
$14 \times 14 \times 112$	SI-IR(Exp=6) 5×5	2	112	192	672	$7 \times 7 \times 112$	448	1
$7 \times 7 \times 192$	SI-IR(Exp=6) 5×5	1	192	192	1152	$7 \times 7 \times 192$	768	2
$7 \times 7 \times 192$	SI-IR(Exp=6) 5×5	1	192	192	1152	$3 \times 3 \times 192$	768	2
$7 \times 7 \times 192$	SI-IR(Exp=6) 5×5	1	192	192	1152	$3 \times 3 \times 192$	768	2
$7 \times 7 \times 192$	IR(Exp=6) 3×3	1	192	320	1152	-	-	-
$7 \times 7 \times 320$	1×1, Pool, Clf	1	320	1280	-	-	-	-

lation, we one-third the number of channels in the cheaper branch that uses full input resolution, and down-sample the input to half the resolution for processing the anchor features with the same number of channels as in the original architecture. This strategy seems to provide similar benefits across different architecture families such as EfficientNet and MobileNetV3. We leave the problem of neural architecture search over various design trade-offs (number of anchors, cheaper block dimensions, interpolation strategies, feature merging, etc.) to future work.

Table F.10: EfficientNetV2-Small model.

Stage	Operator	Stride	# In Channels	# Out Channels	# Layers
0	Conv2d, 3×3	2	3	24	1
1	Fused-MBConv (Exp=1) 3×3	1	24	24	2
2	Fused-MBConv (Exp=4) 3×3	2	24	48	4
3	Fused-MBConv (Exp=4) 3×3	2	48	64	4
4	Inverted Residual (Exp=4) 3×3	2	64	128	6
5	Inverted Residual (Exp=6) 3×3	1	128	160	9
6	Inverted Residual (Exp=6) 3×3	2	160	256	15
7	Conv 1×1, Global-Pooling, Classifier	1	256	1280	1

Table F.11: Spatially Interpolated EfficientNetV2-Small model. IR stands for Inverted Residual block.

Stage	Operator	Stride	In Ch.	Out Ch.	Layers
0	Conv2d, 3×3	2	3	24	1
1	Fused-MBConv (Exp=1) 3×3	1	24	24	2
2	SI-Fused-MBConv (Anchor Exp=4, Cheaper Exp=2) 3×3	2	24	48	4
3	SI-Fused-MBConv (Anchor Exp=4, Cheaper Exp=2) 3×3	2	48	64	4
4	SI-IR (Anchor Exp=4, Cheaper Exp=3, Cheaper Groups=2) 3×3	2	64	128	6
5	SI-IR (Anchor Exp=6, Cheaper Exp=3, Cheaper Groups=2) 3×3	1	128	160	9
6	SI-IR (Anchor Exp=6, Cheaper Exp=3, Cheaper Groups=2) 3×3	2	160	256	15
7	Conv 1×1, Global-Pooling, Classifier	1	256	1280	1

Table F.12: Resnet50 model.

Stage	Operator	Stride	In Ch.	Out Ch.	Layers
0	Conv2d, 7×7	2	3	64	1
1	MaxPool, 3×3	2	-	-	1
2	Bottleneck $(1 \times 1, 64) \rightarrow (3 \times 3, 64) \rightarrow (1 \times 1, 256)$	2	64	256	3
3	Bottleneck $(1 \times 1, 128) \rightarrow (3 \times 3, 128) \rightarrow (1 \times 1, 512)$	2	256	512	4
4	Bottleneck $(1 \times 1, 256) \rightarrow (3 \times 3, 256) \rightarrow (1 \times 1, 1024)$	2	512	1024	6
5	Bottleneck $(1 \times 1, 512) \rightarrow (3 \times 3, 512) \rightarrow (1 \times 1, 2048)$	2	1024	2048	3
7	AveragePooling, Classifier	1	2048	1000	1

F.6 Mean and Standard Deviations

We report the mean and standard deviations for the ImageNet classification experiments discussed in the Sec. 8.3.2. Table F.15 reports the mean and standard deviations of three different runs and shows that our empirical evaluation is stable between different runs.

Table F.13: Spatially Interpolated Resnet50 model.

Stage	Operator	Stride	In Ch.	Out Ch.	Layers
0	Conv2d, 7×7	2	3	64	1
1	MaxPool, 3×3	2	-	-	1
2	SI-Bottleneck Anchors $(1 \times 1, 64) \rightarrow (3 \times 3, 64) \rightarrow (1 \times 1, 256)$ Cheaper $(1 \times 1, 16) \rightarrow (3 \times 3, 16) \rightarrow (1 \times 1, 256)$	2	64	256	3
3	SI-Bottleneck Anchors $(1 \times 1, 128) \rightarrow (3 \times 3, 128) \rightarrow (1 \times 1, 512)$ Cheaper $(1 \times 1, 32) \rightarrow (3 \times 3, 32) \rightarrow (1 \times 1, 512)$	2	256	512	4
4	SI-Bottleneck Anchors $(1 \times 1, 256) \rightarrow (3 \times 3, 256) \rightarrow (1 \times 1, 1024)$ Cheaper $(1 \times 1, 64) \rightarrow (3 \times 3, 64) \rightarrow (1 \times 1, 1024)$	2	512	1024	6
5	Bottleneck $(1 \times 1, 512) \rightarrow (3 \times 3, 512) \rightarrow (1 \times 1, 2048)$	2	1024	2048	3
7	AveragePooling, Classifier	1	2048	1000	1

Table F.14: Resnet50 and EfficientNetv2-small. Spatially Interpolated Bottleneck and Fused-Inverted Residual Blocks.

Architecture	**Resolution**	**Accuracy (%)**	**Params (M)**	**MACs (B)**
Resnet50	224	79.51	25.56	4.09
Resnet50 (ours)	224	79.19	27.51	2.67
EfficientNetv2-s	384	83.88	21.46	7.96
EfficientNetv2-s (ours)	384	83.45	21.84	4.93

Table F.15: Mean & Standard Deviation over 3 runs: ImageNet Classification. We compare MobileNetV3 and EfficientNet architectures with the proposed Spatially Interpolated (SI) variants. It clearly shows that SI-MobileNetV3 and SI-EfficientNet achieve up to 40% compute reduction without any significant loss in accuracy. In addition, this improvement does not come with additional storage overhead.

Architecture	**Image Size**	**Accuracy**	**Params**	**MACs (Savings)**	**CPU (4 threads) Latency (ms)**	**CPU (1 thread) Latency (ms)**
MobileNetV3-Large	224×224	75.2 ±0.16 %	5.4M	219M (1.0×)	150 ±8 (1.0×)	305 ±12 (1.0×)
SI-MobileNetV3-Large	224×224	75.1 ±0.12 %	5.2M	171M (0.8×)	110 ±10 (0.7×)	240 ±8 (0.8×)
EfficientNet-B0	224×224	76.84 ±0.08 %	5.3M	390M (1.0×)	290 ±12 (1.0×)	626 ±14 (1.0×)
SI-EfficientNet-B0	224×224	76.75 ±0.06 %	5.4M	264M (0.7×)	220 ±11 (0.8×)	493 ±16 (0.8×)
EfficientNet-B1	240×240	78.83 ±0.09 %	7.8M	700M (1.0×)	467 ±15 (1.0×)	950 ±20 (1.0×)
SI-EfficientNet-B1	240×240	78.84 ±0.11 %	7.8M	477M (0.7×)	320 ±18 (0.7×)	780 ±15 (0.8×)
EfficientNet-B2	260×260	80.09 ±0.15 %	9.2M	1B (1.0×)	730 ±20 (1.0×)	1457 ±24 (1.0×)
SI-EfficientNet-B2	260×260	79.81 ±0.11 %	9.2M	0.7B (0.7×)	480 ±15 (0.6×)	1109 ±24 (0.8×)
EfficientNet-B3	300×300	81.5 ±0.06 %	12.3M	1.9B (1.0×)	1377 ±25 (1.0×)	2819 ±25 (1.0×)
SI-EfficientNet-B3	300×300	81.2 ±0.12 %	12.4M	1.3B (0.7×)	998 ±30 (0.7×)	2465 ±26 (0.9×)

Appendix G

Appendix to DCL

G.1 Toy Examples

Example 1. The loss for one data point is

$$\ell(\theta, n_1, n_2) = \left(750 + \frac{1}{2}\left(\theta + 15 + n_1\right)^2\right)(1 - \sigma(\theta)) + \frac{10}{2}\left(\theta - 15 + n_2\right)^2 \sigma(\theta)$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is sigmoid function. Each data point has (n_1, n_2) arguments which are independent Gaussian noises $n_1, n_2 \sim \mathcal{N}(0, 10)$. The average loss plot in Figure 9·1 is calculated by averaging 10000 datapoint losses which is large enough to approximate the expectation.

Example 2. The objective function plotted in Figure 9·2a is

$$\begin{aligned} f(\theta) = &- \exp\left[-\frac{1}{2}\left((\theta_1 + 3)^2 + (\theta_2 + 2)^2\right)\right] - \exp\left[-\frac{1}{2}\left((\theta_1 - 3)^2 + (\theta_2 - 4)^2\right)\right] \\ &- \exp\left[-\frac{1}{4}\left(\theta_1^2 + \theta_2^2\right)\right] + 2.25 \end{aligned}$$

The constraint function plotted in Figure 9·2b is

$$g(\theta) = 3.75 - 4\exp\left[-\frac{1}{8}\left((\theta_1 + 3)^2 + (\theta_2 + 2)^2\right)\right] - 4\exp\left[-\frac{1}{8}\left((\theta_1 - 3)^2 + (\theta_2 - 4)^2\right)\right]$$

where the feasible set is defined as $g(\theta) \leq 0$.

Objective minimization. We minimize only $f(\theta)$ without constraint function.

Fixed Lagrangian. We minimize $f(\theta) + \lambda g(\theta)$.

Cosine Scheduled Lagrangian. We minimize $f(\theta) + \lambda_{\text{cosine}} g(\theta)$ where λ_{cosine} goes from 0 to a maximum value based on a cosine scheduling.

We use learning rate of 0.1×0.9^z where z is the largest non-positive number that decreases the current Lagrangian for Lagrange based methods or objective for CE training. We use full gradients in each iterate and run the methods for 200 iterates. We pick best performing λ parameters out of $\{.5, 1, 1.5, 2\}$ for each initialization and method pair.

G.2 Proof of Lemma 1

The proof follows from the total-variation inequality (Csiszár and Shields, 2004), which states that for two probability vectors P_1, P_2 on the K-dimensional simplex, we have the following variational characterization for functions $g : [K] \to \mathbb{R}$:

$$\|P_1 - P_2\|_1 = \frac{1}{\alpha} \sup_{\|g\|_\infty \leq \alpha} E_{P_1} g(Z) - E_{P_2} g(Z)$$

Pinsker's inequality (Csiszár and Shields, 2004) says that,

$$\|P_1 - P_2\|_1 \leq \sqrt{\frac{1}{2} D_{KL}(P_1 \mid P_2)}$$

For a given example $(\mathbf{x}, y)$ we choose,

$$g(i) = \frac{\log \frac{P_2(i)}{P_1(i)}}{P_1(i) - P_2(i)} \mathbf{1}_{y=i}.$$

Next, we substitute EMA for $P_1 = f(\hat{\theta}, \mathbf{x})$ and the current model for $P_2 = f(\theta, \mathbf{x})$. We note that $\log(x)/(x-1),\ x > 0$ is a positive monotonically decreasing sequence and

bounded from above by $1/x$. It follows that, $\|g\|_\infty \leq \frac{P_2(y)}{P_1(y)}$. Putting these together, combining with the total-variation inequality and Pinsker's inequality, and summing over all the data, we obtain:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j\in[K]}\mathbf{1}_{y_i=j}\log(f_j(\theta,\mathbf{x}_i)) - \frac{1}{N}\sum_{i=1}^{N}\sum_{j\in[K]}\mathbf{1}_{y_i=j}\log(f_j(\hat{\theta},\mathbf{x}_i)) \leq \frac{1}{N}\sum_{i=1}^{N}\frac{P_2(y_i)}{P_1(y_i)}\sqrt{\frac{1}{2}D_{KL}(P_1 \mid P_2)}$$

We note that LHS is the difference of cross entropy losses, and the conditions of our Lemma ensure positivity of LHS. Therefore, squaring both sides and using the fact that $P_1(y_i) \geq \epsilon$, the result follows using Jensen's inequality, i.e.,

$$\frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{2}D_{KL}(P_1 \mid P_2)} \leq \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}D_{KL}(P_1 \mid P_2)}$$

G.3 Architecture & Baseline Details

In this section, we enumerate the definitions of the architectures used in the main text. Table 13.2 and Table 9.3 shows the resource usage of the ResNet18, ResNet50, and ShuffleNetV2 architectures. Below, we describe individual model for completeness.

- **CIFAR-100 & Tiny-ImageNet.** We borrow the ShuffleNetV2 (Ma et al., 2018) definitions from `https://github.com/kuangliu/pytorch-cifar`. ResNet models share a similar structure except their building blocks. Their architecture described in sequence consists of a convolutional block, followed by four residual block stages, followed by the adaptive average pooling layer and the classifier layer. ResNet18 uses 'BasicBlock' (He et al., 2016) as the building block while ResNet50 uses 'Bottleneck' (He et al., 2016). Different capacity models in this family differ only in the number of repetitions of the residual block and the number of filters in each

stage. Below, we write the different of repetitions and the number of filters for the four different residual stages.

- *Resnet18* has $[64, 128, 256, 512]$ filters and repeats the 'BasicBlock' $[2, 2, 2, 2]$ times.
- *Resnet50* has $[64, 128, 256, 512]$ filters and repeats the 'Bottleneck' $[3, 4, 6, 3]$ times.

• **ImageNet.** Architectural definitions for ImageNet datasets differ only in the first convolutional block that is adapted to the input resolution $224 \times 224 \times 3$ which is larger than the CIFAR-100's $32 \times 32 \times 3$. We use the ResNet18 and ResNet50 definitions from the popular timm repository (Wightman, 2019) while we borrow the ShuffleNetV2 definition from the official PyTorch repository https://pytorch.org/hub/pytorch_vision_shufflenet_v2/. For the CIFAR-100 pre-training experiment, we borrow the pre-trained weights available from the respective repositories.

We point out that all our models match their publicly available implementations and achieve similar accuracy as these public implementations. For the pre-training baseline reported in the Table 13.3, we borrow the respective model's ImageNet definition along with their pre-trained weights. Then, we scale the $32 \times 32 \times 3$ image to the ImageNet resolution $224 \times 224 \times 3$ and fine-tune this updated model on the CIFAR-100 dataset with similar setup as the other CIFAR-100 experiments except with reduced learning rate 0.01 and 30 epochs since the model is already pre-trained and has learnt good representations.

G.4 Hyper-parameters

For all our experiments (both CE, FL, and DCL), we use the popular cosine learning rate scheduler for the SGD optimizer with 0.1 learning rate, 0.9 momentum and $5e-4$ weight decay. We use 128 as the batch size and 0.996 as the EMA hyper-parameter. We report the EMA accuracy for all our methods including the baselines. We point

out that for fair comparison, similar to DCL, we use EMA for the baselines as well and include the similar multi-stage training for the baselines (CE and FL). Each stage consists of 200 epochs. In the absence of a validation set, we tuned various hyper-parameters described below using the augmented train accuracy as a proxy.

For DCL, we scan the different hyper-parameters in the following ranges: (a) $\tau \in \{1, 3.5, 4\}$, (b) $\lambda_{\text{min}} \in \{0.01, 0.1\}$, (c) $\lambda_{\text{max}} \in \{1, 2, 5, 10\}$, (d) Budget $\delta \in \{0.01, 0.02, 0.05, 0.1\}$, and (e) $\lambda_T \in \{50, 200\}$. We replace the $\arg\min$ in the Algorithm 5, with stochastic gradient descent over the entire dataset.

For FL, we scan the different hyper-parameters in the following ranges: (a) $\tau \in \{1, 3.5, 4\}$, and (b) $\lambda \in \{0.01, 0.1, 1, 2, 5, 10\}$. We replace the $\arg\min$ in the Algorithm 5, with stochastic gradient descent over the entire dataset.

Note that the default hyper-parameters: $\tau = 4$, $\lambda_{\text{min}} = 0.1$, $\lambda_{\text{max}} = 5$, $\delta = 0.05$, and $\lambda_T = 200$, work well in most of our experiments.

We run the ImageNet experiments for 90 epochs with SGD optimizer with 0.9 as momentum and 0.1 as the learning rate with cosine decay. We use the batch size of 256 for training with a weight decay term of $1e - 5$. For ViT-Tiny model, we follow setup (hyper-parameters, data augmentations, etc.) described in the (Wightman, 2019) repository.

Appendix H

Appendix to OSP

H.1 Appendix to §11.2

H.1.1 Proof of Proposition 1

Proof. We recall the notation. $\alpha \in [0,1]^K$ is such that $\sum \alpha_k \leq 1$. The $\mathcal{T}_k^\alpha$ are the optimising solutions to the OSP problems at error $\alpha_k \varepsilon$, i.e.

$$\mathcal{T}_k^\alpha \in \arg\max \mathbb{P}(\mathcal{T}) \text{ s.t. } \mathbb{P}(X \in \mathcal{T}, Y \neq k) \leq \alpha_k \varepsilon,$$

while the $\mathcal{S}_k^\alpha$ are produced by removing the smaller overlap with smaller labels in $\mathcal{T}_k^\alpha$, i.e.

$$\mathcal{S}_k^\alpha = \mathcal{T}_k^\alpha \setminus \bigcup_{k'<k} \mathcal{T}_{k'}^\alpha.$$

We first argue that the total overlap of the $\mathcal{T}$s is small.

Lemma 3. *Let $\mathcal{T}_k^\alpha$ be generated as above. Then*

$$\sum_k \mathbb{P}(\bigcup_{k' \neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha) \leq 2\varepsilon.$$

Since the total overlap is $\bigcup_k \left(\mathcal{T}_k^\alpha \cap \bigcup_{k' \neq k} \mathcal{T}_{k'}^\alpha\right)$, this also controls the probability of the

total overlap, that is,

$$\mathbb{P}(\bigcup_{k,k'\neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha) \leq 2\varepsilon.$$

This lemma is sufficient to show the claim, since

$$\begin{aligned}
\sum_k \mathbb{P}(\mathcal{S}_k) &= \sum_k \mathbb{P}(\mathcal{T}_k^\alpha \setminus \bigcup_{k'<k} \mathcal{T}_{k'}^\alpha) \\
&\geq \sum_k \mathbb{P}(\mathcal{T}_k^\alpha) - \sum_k \mathbb{P}(\mathcal{T}_k^\alpha \cap \bigcup_{k'<k} \mathcal{T}_{k'}^\alpha) \\
&\geq \sum_k \mathbb{P}(\mathcal{T}_k^\alpha) - \sum_k \mathbb{P}(\mathcal{T}_k^\alpha \cap \bigcup_{k'\neq k} \mathcal{T}_{k'}^\alpha) \\
&\geq C(\varepsilon; \mathscr{S}) - \sum_k \mathbb{P}(\mathcal{T}_k^\alpha \cap \bigcup_{k'\neq k} \mathcal{T}_{k'}^\alpha) \\
&\geq C(\varepsilon; \mathscr{S}) - 2\varepsilon,
\end{aligned}$$

where we have used that $\sum_k \mathbb{P}(\mathcal{T}_k^\alpha) \geq C(\varepsilon; \mathscr{S})$, which holds because the $\mathcal{T}_k^\alpha$ optimise a relaxation of (SC), and the final inequality is due to the above lemma. □

We conclude by proving the above lemma

Proof of Lemma 3. Since the labels of Y are mutually, exclusive,

$$\sum_k \mathbb{P}(\bigcup_{k'\neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha) = \sum_k \sum_j \mathbb{P}(\bigcup_{k'\neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha, Y = j).$$

Applying Fubini's theorem, and recalling that the probability of an intersection of events is smaller than the probability of either of the events, we see that

$$\begin{aligned}
\sum_k \mathbb{P}(\bigcup_{k'\neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha) &= \sum_k \sum_j \mathbb{P}(\mathcal{T}_k^\alpha \cap (\bigcup_{k'\neq k} \mathcal{T}_{k'}^\alpha), Y = j) \\
&\leq \sum_k \left(\sum_{j\neq k} \mathbb{P}(\mathcal{T}_k^\alpha, Y = j) \right) + \mathbb{P}(\bigcup_{k'\neq k} \mathcal{T}_{k'}^\alpha, Y = k),
\end{aligned}$$

Now, notice that the sum in the brackets is simply $\mathbb{P}(\mathcal{T}_k^\alpha, Y \neq k)$. Taking the union bound over the second probability, we find the upper bound

$$\begin{aligned}
\sum_k \mathbb{P}(\bigcup_{k' \neq k} \mathcal{T}_k^\alpha \cap \mathcal{T}_{k'}^\alpha) &\leq \sum_k \mathbb{P}(\mathcal{T}_k^\alpha, Y \neq k) + \sum_k \sum_{k' \neq k} \mathbb{P}(\mathcal{T}_{k'}^\alpha, Y = k) \\
&= \sum_k \mathbb{P}(\mathcal{T}_k^\alpha, Y \neq k) + \sum_{k'} \sum_{k \neq k'} \mathbb{P}(\mathcal{T}_{k'}^\alpha, Y = k) \\
&= \sum_k \mathbb{P}(\mathcal{T}_k^\alpha, Y \neq k) + \sum_{k'} \mathbb{P}(\mathcal{T}_{k'}^\alpha, Y \neq k') \\
&= 2 \sum_k \mathbb{P}(\mathcal{T}_k^\alpha, Y \neq k) \\
&\leq 2 \sum \alpha_k \varepsilon = 2\varepsilon,
\end{aligned}$$

where the first equality is by Fubini's theorem again, the second equality is by the disjointness of the values of Y, and the final inequality is due to the constraints of the OSP problems.

□

H.1.2 Asyptotically Feasible Finite Sample Analysis for SC

In parallel to the OSP problems, one can directly give finite sample analyses for the SC problem. We begin by defining the solution concept here.

Definition 2. *We say that a class $\mathscr{S}$ is learnable with abstention if for every $(\delta, \zeta, \sigma, \nu) \in (0,1)^4$, there exists a finite $m(\delta, \sigma, \nu, \zeta)$ and an algorithm $\mathfrak{A} : (\mathcal{X} \times \{+,-\})^m \to \mathscr{S}^K$ such that for any law $\mathbb{P}$, and $\varepsilon > 0$, given n i.i.d. samples drawn from $\mathbb{P}$, the algorithm produces sets $\{\mathcal{S}_k\}$ from $\mathscr{S}$ such that with probability at least*

$1-\delta$ *over the data,*

$$\sum_k \mathbb{P}(\mathcal{S}_k) \geq C(\varepsilon; \mathscr{S}) - \sigma$$
$$\mathbb{P}(\mathcal{E}_{\{\mathcal{S}_k\}}) \leq \varepsilon + \nu$$
$$\mathbb{P}(\bigcup_{k,k' \neq k} \mathcal{S}_k \cap \mathcal{S}_{k'}) \leq \zeta.$$

Notice that the recovered sets need not be disjoint, which may be amended by by eliminating the overlap from one of the sets as in §11.2.2. The resulting (improper) sets attain coverage of at least $C - \zeta - \nu$ with high probability.

The main point characterisation here is similar,

Proposition 4. *A class $\mathscr{S}$ is learnable with abstention if and only if it has finite VC dimension, and further,*

$$m(\delta, \sigma, \nu, \zeta) = \widetilde{O}(\mathrm{poly}(K) \max(\sigma^{-2}, \nu^{-2}, \zeta^{-1}) \mathrm{VC}(\mathscr{S})).$$

The proof of the necessity of finite VC dimension follows from observing that if the data is realisable, i.e., corresponds to $Y = 2\mathbb{1}\{X \in \mathcal{S}\} - 1$ for some $\mathcal{S} \in \mathscr{S}$ then at least one of the recovered sets is a good classifier, at which point standard lower bounds for realisable PAC learning apply. The sufficiency follows from utilising the finite VC property to uniformly bound errors incurred by empirical means. The proof is presented in §H.1.3.

H.1.3 Proofs of Propositions 2 and 4

Proofs of Necessity of Finite VC dimension

In both cases, we reduce the problems to realisable PAC learning, and invoke standard bounds for the same, for instance the one of (Mohri et al., 2018, Ch.3). To this

end, suppose $\delta \leq 1/100$, and consider the restricted class of joint laws $\mathbb{P}$ such that $\mathbb{P}(Y = k|X = x) = \mathbb{1}\{X \in \mathcal{S}_{k,*}\}$ for some disjoint $\{\mathcal{S}_{k,*}\} \in \mathscr{S}$ that together cover $\mathcal{X}$.[1]

Proof for One-Sided Prediction. Notice that $\mathcal{S}_1^*$ is feasible for OSP-1 for any value of ε. If we can solve OSP-1, then we would have found a set $\mathcal{S}$ such that

$$\mathbb{P}(\mathcal{S}) \geq \mathbb{P}(\mathcal{S}_{1,*}) - \sigma$$
$$\mathbb{P}(X \in \mathcal{S} \cap \mathcal{S}_{1,*}^c) = \mathbb{P}(X \in \mathcal{S}, Y = 2) \leq \varepsilon + \nu.$$

Further,

$$\begin{aligned}\mathbb{P}(\mathcal{S}^c) &= \mathbb{P}(\mathcal{S}^c \cap \mathcal{S}_{1,*}) + \mathbb{P}(\mathcal{S}^c \cap \mathcal{S}_{1,*}^c) \\ &= \mathbb{P}(\mathcal{S}^c \cap \mathcal{S}_{1,*}) + \mathbb{P}(\mathcal{S}_{1,*}^c) - \mathbb{P}(\mathcal{S} \cap \mathcal{S}_{1,*}^c).\end{aligned}$$

But $\mathbb{P}(\mathcal{S}^c) = 1 - \mathbb{P}(\mathcal{S}) \leq 1 - \mathbb{P}(\mathcal{S}_{1,*}) + \sigma \leq \mathbb{P}(\mathcal{S}_{1,*}^c) + \sigma$.

Thus, we have

$$\begin{aligned}&\mathbb{P}(\mathcal{S}^c \cap \mathcal{S}_{1,*}) + \mathbb{P}(\mathcal{S}_{1,*}^c) - \mathbb{P}(\mathcal{S} \cap \mathcal{S}_{1,*}^c) \leq \mathbb{P}(\mathcal{S}_{1,*}^c) + \sigma \\ &\qquad \implies \mathbb{P}(\mathcal{S}^c \cap \mathcal{S}_{1,*}) \leq \sigma + \mathbb{P}(\mathcal{S} \cap \mathcal{S}_{1,*}^c) \leq \varepsilon + \sigma + \nu.\end{aligned}$$

But then, viewed as a standard classifier for the problem separating the class $\{1\}$ from $[2 : K]$, $\mathcal{S}$ has risk at most $2\varepsilon + \sigma + \nu$. Consequently, an algorithm for solving OSP yields an algorithm for realisable PAC learning for this problem. Thus, invoking

[1] Strictly speaking, this requires that $\mathscr{S}$ is rich enough to express such a class. This is a very mild assumption. For the purposes of the lower bound, in fact, this can be weakened still - all we really need is a binary law, and that if $\mathcal{S} \in \mathscr{S}$, then $\mathcal{S}^c \in \mathscr{S}$. Then we can take $\mathbb{P}(Y = 1|X = x) = \mathbb{1}\{X \in \mathcal{S}\}, \mathbb{P}(Y = 2|X = x) = \mathbb{1}\{X \in \mathcal{S}^c\}$, and the entirety of the following argument goes through without change.

the appropriate standard lower bound, we conclude that

$$m_{\text{OSP}} \geq \frac{\text{VC}(\mathscr{S}) - 1}{32(2\varepsilon + \sigma + \nu)}. \qquad \square$$

Proof for Learning With Abstention. Notice that $\{\mathcal{S}_{k,*}\}$ serve as a feasible solution for any ε, and have total coverage 1. Thus, if SC is possible, we may recover sets $\{\mathcal{S}_k\}$ such that

$$\begin{aligned} \sum \mathbb{P}(\mathcal{S}_k) &\geq 1 - \sigma \\ \mathbb{P}(\mathcal{E}_{\{\mathcal{S}_k\}}) &\leq \varepsilon + \nu \\ \mathbb{P}\left(\bigcup_k (\mathcal{S}_k \cap \bigcup_{k' \neq k} \mathcal{S}_{k'})\right) &\leq \nu. \end{aligned}$$

Now notice that $\mathcal{S}_{1,*}$ and $\mathcal{S}_{1,*}^c$ correspond to the realisable classifiers for the binary classification problem separating $\{1\}$ from $[2:K]$.[2] But, in the same way, we may view $\mathcal{S}_1$ and $\mathcal{S}_1^c$ as binary classifiers for this problem. Now notice that for this binary classification problem, $\mathcal{S}_1$ incurs small error. Indeed, denoting $\mathcal{S}_{\neq 1} = \bigcup_{k' \neq 1} \mathcal{S}_{k'}$, we find that

$$\begin{aligned} \mathbb{P}(X \in \mathcal{S}_1, Y \neq 1) + \mathbb{P}(X \in \mathcal{S}_1^c, Y = 1) &= \mathbb{P}(X \in \mathcal{S}_1, Y \neq 1) + \mathbb{P}(X \in \mathcal{S}_{\neq 1} \cap \mathcal{S}_1^c, Y = 1) \\ &\quad + \mathbb{P}(X \in \mathcal{S}_{\neq 1}^c \cap \mathcal{S}_1^c, Y = 1) \\ &\leq \mathbb{P}(X \in \mathcal{S}_1, Y \neq 1) + \mathbb{P}(X \in \mathcal{S}_{\neq 1}, Y = 1) \\ &\quad + \mathbb{P}(X \in \mathcal{S}_1^c \cap \mathcal{S}_{\neq 1}^c) \\ &\leq \mathbb{P}(\mathcal{E}_{\{\mathcal{S}_k\}}) + (1 - \mathbb{P}(\mathcal{S}_1 \cup \mathcal{S}_{\neq 1}^c))) \\ &\leq \varepsilon + K\nu + \sigma + \zeta, . \end{aligned}$$

where the second line's inequality is just non-negativity of probabilities, and the

[2] Again, this needs that $\mathscr{S}$ is rich enough to include $\mathcal{S}_{1,*}^c$.

third line's inequality is due the fact that $\mathbb{P}(\mathcal{E})$ is controlled, and the following inclusion-exclusion argument, first note that

$$\mathbb{P}(\mathcal{S}_1 \cup \mathcal{S}_{\neq 1}) = \mathbb{P}(\bigcup \mathcal{S}_k) = \sum_k \mathbb{P}(\mathcal{S}_k) - \sum_k \mathbb{P}(\mathcal{S}_k \cap \bigcup_{k'>k} \mathcal{S}_{k'}).$$

Next, observe that if $j > k$, $\mathcal{S}_j \subset \bigcup_{k'>k} \mathcal{S}_{k'}$, and similarly $\bigcup_{k'>j} \mathcal{S}_{k'} \subset \bigcup_{k'>k} \mathcal{S}_k$. Thus,

$$\mathbb{P}(\mathcal{S}_1 \cup \mathcal{S}_{\neq 1}) = \mathbb{P}(\bigcup \mathcal{S}_k) \geq \sum_k \mathbb{P}(\mathcal{S}_k) - K\mathbb{P}(\mathcal{S}_1 \cap \bigcup_{k'>1} \mathcal{S}_{k'}).$$

Now invoking the SC solution conditions, the first sum is at least $1 - \sigma$, while the second probability is bounded by the probability of overlap, giving

$$\mathbb{P}(\mathcal{S}_1 \cup \mathcal{S}_{\neq 1}) \geq 1 - \sigma - K\nu.$$

Thus, a SC yields a realisable PAC learner for the binary classifier problem separating $\{1\}$ from $[2 : K]$, giving the bound

$$m_{\text{SC}} \geq \frac{\text{VC}(\mathscr{S}) - 1}{32(\varepsilon + \sigma + K\nu + \zeta)}. \qquad \square$$

Note that these bounds are likely loose. The problems have plenty of structure that is not exploited in either of the above statements, and tighter inequalities would be of interest. However the point we intend to pursue - that assuming finiteness of VC dimensions in the upper bound analyses is not lossy, is sufficiently made above.

Proofs of the Upper Bounds

We mainly make use of the following uniform generalisation bound on the suprema of empirical processes due to the finiteness of VC dimension. This is again standard, and may be seen in (Mohri et al., 2018).

Lemma 5. *Let $\mathscr{S}$ have finite VC dimension. Then for any distribution $\mathbb{P}$, if $\widehat{P}_m$ denotes the empirical law induced by m i.i.d. samples from $\mathbb{P}$, then with probability at least $1-\delta$ over these samples,*

$$\sup_{\mathcal{S}\in\mathscr{S},k\in[1:K]} |\widehat{\mathbb{P}}_m(X\in\mathcal{S},Y=k)-\mathbb{P}(X\in\mathcal{S},Y=k)| \le C_K\sqrt{\frac{\textsc{vc}(\mathscr{S})\log m+\log(C/\delta)}{m}},$$

where C_K is a constant independent of $\mathscr{S},\delta,\mathbb{P},m$.

Notice that by summing over the values of Y, this also controls the error in the objects $\mathbb{P}(X\in\mathcal{S})$ and $\mathbb{P}(X\in\mathcal{S},Y\neq k)$, possibly with an error blowup of K, which can be absorbed into C_K.

For the purposes of the following, let $\Delta_{m,\mathscr{S}}(\delta)$ be the value of the upper bound above.

Proof of Upper Bound for OSP. For $\alpha\in[0,1]$, define $\mathscr{S}_\alpha\subset\mathscr{S}$ to be the subset of $\mathcal{S}$s that have $\mathbb{P}(\mathcal{E}^1_{\mathcal{S}})\le\alpha$, and let σ,ν be quantities that we will choose.

We give a two phase scheme - first we collect all sets $\mathcal{S}$ such that $\widehat{P}_m(\mathcal{E}^1_{\mathcal{S}})\le\varepsilon+\nu/2$ into the set $\widehat{\mathscr{S}}_{\varepsilon+\nu/2}$. Notice that as long as $\nu/2>\Delta_{m,\mathscr{S}}(\delta/2)$, we have w.p. at least $1-\delta/2$ that

$$\mathscr{S}_\varepsilon\subset\widehat{\mathscr{S}}_{\varepsilon+\nu/2}\subset\mathscr{S}_{\varepsilon+\nu}.$$

Due to the upper inclusion, with probability at least $1-\delta/2$, every set in $\widehat{\mathscr{S}}_{\varepsilon+\nu/2}$ has error level at most $\varepsilon+\nu$.

Next, we choose the $\mathcal{S}\in\widehat{\mathscr{S}}_{\varepsilon+\nu/2}$ that has the biggest coverage. If $\mathscr{S}_\varepsilon\subset\widehat{\mathscr{S}}_{\varepsilon+\nu/2}$, and $\sigma>\Delta_{m,\mathscr{S}}(\delta/2)$, we are again assured that the selected answer will be at least $\sup_{\mathcal{S}\in\mathscr{S}_\varepsilon}\mathbb{P}(\mathcal{S})-\Delta_{m,\mathscr{S}}(\delta/2)>L_k-\sigma$ with probability at least $1-\delta/2$. By the union bound, these will hold simultaneously with probability at least $1-\delta$. Since we want the smallest σ,ν, but for the arguments to follow we need that these are bigger than

$2\Delta_{m,\mathscr{S}}(\delta/2)$, we can set

$$\nu = \sigma = 4\Delta_{m,\mathscr{S}}(\delta/2) = 4C\sqrt{\frac{\textsc{vc}(\mathscr{S})\log m + \log(2C/\delta)}{m}},$$

concluding the proof. □

Proof of Upper Bound for SC. This proceeds similarly to the above. To neatly present this, we let $\mathscr{R} = \{\mathcal{S} : \mathcal{S} = \bigcup_{k,k'\neq k} \mathcal{S}_k \cap \mathcal{S}_{k'}, \{\mathcal{S}_k\}) \in \mathscr{S}\}$ be the class of sets obtained by taking pairwise intersection of k-tuples in $\mathscr{S}$. Note that VC dimesnsion of the sets obtained by taking pairwise intersection of sets in $\mathscr{S}$ at most doubles the VC dimesnsion, while taking the $\binom{K}{2}$ unions in turn blows it up by a factor of $O(K^2 \log K$ by Lemma 3.2.3 of (Blumer et al., 1989). Thus $\textsc{vc}(\mathscr{R}) = O(K^2\textsc{vc}(\mathscr{S})\log K)$. Now we may proceed as above, first by filtering the pairs of sets that satisfy the intersection constraint with value $\zeta/2$ on the empirical distribution, and then similarly checking the sum-error constraints and finally optimising the sum of their masses. The bounds are the same as the above, except with $\textsc{vc}(\mathscr{S})$ replaced by $O(K^2\textsc{vc}(\mathscr{S})\log K)$. □

Analyses not pursued here

We first point out that there is nothing special about the VC theoretic analysis here - alternate methods like Rademacher complexity or a covering number analysis may replace Lemma 5. Similarly, the same analysis could be extended, via Rademacher complexities, to the setting of indicators relaxed to Lipschitz surrogates by exploiting Talagrand's lemma.

We note a few further analyses that we do not pursue here - firstly, using the technique of (Rigollet and Tong, 2011), it should be possible to give analyses for SC under convex surrogates of the indicator losses and a slight extension of the class $\mathscr{S}$ while directly attaining the constraints (instead of asymptotically) with high prob.

Additionally, a number of papers concentrate on deriving fast rates for the excess risks under the assumption of realizability (i.e., under the assumption that level sets of η can be expressed via $\mathscr{S}$), and that Tsybakov's noise condition holds at the level relevant to the optimal solution.

H.2 Algorithmic rewriting of Section 11.3

We specify the conclusions of §11.3 without any of the justifying development.

Model class and Architecture We use a DNN with the following structure:

- A 'backbone', parametrised by θ, which may have any convenient architecture.
- A 'last layer' with K outputs, denoted f_k, and associated weights w_k for each. We denote $\mathbf{w} = (w_1, \ldots, w_K)$.
- Let $\xi_\theta(x)$ denote the backbone's output on a point x. The DNN's outputs are

$$f(x; \theta, \mathbf{w}) = (f_1, \ldots, f_k)(x; \theta, \mathbf{w}) = \mathrm{softmax}(\langle w_1, \xi_\theta(x)\rangle, \ldots, \langle w_K, \xi_\theta(x)\rangle).$$

Objective function and Training We use the following objective function, where the $\{(x_i, y_i)\}_{i=1}^n$ comprise the training dataset, $\theta, \mathbf{w}$ are model parameters, $\{\varphi_k\}$ are autotuned hyperparameters, $\{\lambda_k\}$ are autotuned multipliers, and μ is the single externally tuned parameter. Similarly to $\mathbf{w}$, we define $\boldsymbol{\varphi} := (\varphi_1, \ldots, \varphi_K)$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_K)$.

$$\begin{aligned}\widetilde{M}^{\text{res.}}(\theta, \mathbf{w}, \boldsymbol{\varphi}, \boldsymbol{\lambda}, \mu) = \sum_{k=1}^{K} & \frac{\sum_{i:y_i=k} -\log f_k(x_i; \theta, \mathbf{w})}{n_k} \\ & + \lambda_k \left(\frac{\sum_{i:y_i \neq k} -\log(1 - f_k(x_i; \theta, \mathbf{w})}{n_{\neq k}} - \varphi_k \right) + \mu\varphi_k,\end{aligned}$$

where $n_k := |\{i : y_i = k\}|, n_{\neq k} := |\{i : y_i \neq k\}|$.

The minimax problem we propose is

$$\min_{\theta,\mathbf{w},\boldsymbol{\varphi}} \max_{\boldsymbol{\lambda}:\forall k,\lambda_k\geq 0} \widetilde{M}^{\text{res.}}(\theta,\mathbf{w},\boldsymbol{\varphi},\boldsymbol{\lambda},\mu), \tag{H.1}$$

which is optimised via SGDA in §11.4.

Overall Scheme and Model Selection is presented in Algorithm 9. The subroutine involving the minimax solution requires training data, but this is not mentioned in the same since the focus is on model selection. The training procedure is described in §11.4. $\widehat{\mathbb{P}}_V$ refers to the empirical law on the validation dataset.

Algorithm 9 OSP-Based Selective Classifier: Model Selection

1: **Inputs**: Validation data $\{V\}$, List of μ values $\mathbf{M}$, List of t values $\mathbf{T}$, Target Error ε.
2: **for** each $\mu \in \mathbf{M}$, **do**
3: $\quad(\theta(\mu),\mathbf{w}(\mu)) \leftarrow$ minimax solution of the program (H.1) with this value of μ.
4: **for** each $(\mu,t) \in \mathbf{M}\times\mathbf{T}$, **do**
5: $\quad\mathcal{S}_k(\mu,t) \leftarrow \{x : k = \arg\max_j f_j(x;\theta(\mu),\mathbf{w}(\mu))\} \cap \{x : f_k(x;\theta(\mu),\mathbf{w}(\mu)) > t\}$.
6: $\quad\widehat{E}_V(\mu,t) \leftarrow \widehat{\mathbb{P}}_V(\mathcal{E}_{\{\mathcal{S}_k(\mu,t)\}})$.
7: $\quad\widehat{C}_V(\mu,t) \leftarrow \sum_k \widehat{\mathbb{P}}_V(X \in \mathcal{S}_k(\mu,t))$.
8: $(\mu_*,t_*) = \arg\max_{\mathbf{M}\times\mathbf{T}} \widehat{C}_V(\mu,t)$ s.t. $\widehat{E}_V(\mu,t) \leq \varepsilon$.
9: **Return**: $\{\mathcal{S}_k(\mu_*,t_*)\}$.

H.3 Experimental Details

The table below presents the values of the various hyperparameters used for the entries of Table 11.2.

The following two tables update the numbers for Deep Gamblers to the case where we scan for 40 values of $\mathcal{o}$ in the set $[1,10)$ (as intended in the specifications) instead of $[1,2)$.

Dataset	Algorithm	Hyper-parameters
CIFAR-10	Softmax Response	$t = 0.0445$
	Selective Net	$\lambda = 32, c = 0.51, t = 0.24$
	Deep Gamblers	$o = 1.179, t = 0.03$
	OSP-Based	$\mu = 0.49, t = 0.8884$
SVHN-10	Softmax Response	$t = 0.0224$
	Selective Net	$\lambda = 32, c = 0.79, t = 0.86$
	Deep Gamblers	$o = 1.13, t = 0.23$
	OSP-Based	$\mu = 1.67, t = 0.9762$
Cats v/s Dogs	Softmax Response	$t = 0.029$
	Selective Net	$\lambda = 32, c = 0.7, t = 0.73$
	Deep Gamblers	$o = 1.34, t = 0.06$
	OSP-Based	$\mu = 1.67, t = 0.9532$

Table H.1: Final hyper-parameters used for all the algorithms (at the desired 0.5% error level) in Table 11.2.

Dataset	Target	OSP-based		SR		SN		DG	
	Error	Cov.	Error	Cov.	Error	Cov.	Error	Cov.	Error
CIFAR-10	2%	**80.6**	1.91	75.1	2.09	73.0	2.31	72.9	1.99
	1%	**74.0**	1.02	67.2	1.09	64.5	1.02	63.5	1.01
	0.5%	**64.1**	0.51	59.3	0.53	57.6	0.48	56.1	0.51
SVHN-10	2%	**95.8**	1.99	95.7	2.06	93.5	2.03	94.7	2.01
	1%	**90.1**	1.03	88.4	0.99	86.5	1.04	89.7	0.99
	0.5%	**82.4**	0.51	77.3	0.51	79.2	0.51	81.4	0.51
Cats & Dogs	2%	**90.5**	1.98	88.2	2.03	84.3	1.94	87.4	1.94
	1%	**85.4**	0.98	80.2	0.97	78.0	0.98	81.7	0.98
	0.5%	**78.7**	0.49	73.2	0.49	70.5	0.46	74.5	0.48

Table H.2: Performance at Low Target Error. This repeats Table 11.2, except that the hyperparameter scan for the DG method is corrected, and the entries in the DG columns are updated to show the resulting values. Notice that the performance in the last column is worse than in Table 11.2.

Dataset	Target	OSP-based		SR		SN		DG	
	Coverage	Cov.	Error	Cov.	Error	Cov.	Error	Cov.	Error
CIFAR-10	100%	100	9.74	99.99	**9.58**	100	11.07	100	10.95
	95%	95.12	**6.98**	95.24	8.74	94.71	8.34	95.01	8.29
	90%	90.02	**4.67**	90.51	6.52	89.56	6.45	90.01	6.28
SVHN-10	100%	100	4.27	99.97	**3.86**	100	4.27	100	4.01
	95%	95.05	**1.83**	95.06	1.86	95.14	2.53	95.01	2.07
	90%	90.09	**1.01**	89.99	1.04	90.14	1.31	90.01	1.06
Cats & Dogs	100%	100	5.93	100	**5.72**	100	7.36	100	6.16
	95%	95.13	**2.97**	95.02	3.46	95.21	5.1	95.1	4.28
	90%	90.01	**1.74**	90.02	2.28	90.18	3.3	90.02	2.5

Table H.3: Performance at High Target Coverage. Similarly to the previous table, this repeats Table 11.3 but with the scan for the DG method corrected. Again note the reduced performance in the final column relative to Table 11.3.

Appendix I

Appendix to Hybrid Models

I.1 Illustrative Example Details

Edge and Cloud devices. We use a V100 GPU as the cloud device. It has a 16GB VRAM and the server associated with this GPU has 1TB storage. We use STM32F746 [1], an ARM Cortex-M7 MCU as the edge device. It has 320KB SRAM and 1MB Flash storage.

On-Cloud Baseline. We use the MASS-600 (Cai et al., 2020) as the global model in this experiment. It achieves 79.9% Top1 accuracy on the ImageNet dataset. This model has a computational footprint of 595M MACs. As per our benchmarking, the inference latency of this model on a V100 GPU is 25ms.

On-Device Baselines. (Lin et al., 2020a) have explored deploying tiny MCUNet models on the STM32F746 MCU. We borrowed their pre-trained MCUNet models from their github repository [2] [3]. There are three different models with varying inference latencies and accuracies, namely: (a) model-A (12M MACs, 0.6M parameters, 200ms latency, 51.1% accuracy), (b) model-B (81M MACs, 0.74M parameters, 1075ms latency, 60.9% accuracy), and (c) model-C (170M MACs, 1.4M parameters, 1400ms latency, 63.5% accuracy). We use the smalled model, i.e., model-A as the base model in training the hybrid model. This model has the least latency as well as the least

[1] https://www.st.com/en/microcontrollers-microprocessors/stm32f746ng.html
[2] https://github.com/mit-han-lab/tinyml/tree/master/mcunet
[3] https://github.com/rouyunpan/mcunet

accuracy among the three models. Note that this model has the input resolution of $96 \times 96 \times 3$ and as a result the input size becomes nearly 28KB.

While we are aware of the updated version of MCUNet, i.e., MCUNetV2 (Lin et al., 2021), we point out that MCUNetV2 models are not publicly available for us to include in the empirical evaluation. Besides, our hybrid framework is completely agnostic to what the base and global model choices are. One can use any base and global model in the above setup and observe the relative gap between various baselines.

Energy Profile Base Model Execution and Communication Cost. We assume that the MCU device operates at 200MHz clock speed and is connected to a 3.6V power supply. It has an active mode current characteristics of 72μA per MHz. Thus, it consumes $= 3.6 \times 72 \times 200 = 51.84$ mW (i.e. mili Joules per second) energy in active mode. As a result, the energy consumed by the base model (200ms latency) on this edge device is $= 51.84 \times 0.2 = 10.368$ mili Joules. Similar calculation yield the energy consumption the on-device models: model-B (60mJ) and model-C (80mJ).

We use the NB-IoT (Chen et al., 2017) communication protocol with 110kbps transmission rate and a transmission current characteristics of 30mA. Thus, it consumes $= 30 \times 3.6 = 108$ mili Joules per second in transmission mode. Let us calculate the time taken to transfer the input image from the base model on the edge device to the global model on cloud. The base model input has size 28KB and the transmission rate is 110kbps. Thus, it takes $= \frac{28*8}{110} \approx 2$ seconds to transfer this image from the edge device to the cloud. As a result, the energy consumed by the edge device in transmitting this image is $= 108 * 2 = 216$mJ.

Split-Computation Baseline. We use this term to refer methods (Kang et al., 2017; Teerapittayanon et al., 2017) wherein the initial part of the global model executes on-device while the remaining executes on-cloud. The initial network allows for

an early exit classifier for easy examples. Split into early layers results in high communication cost but split in later layers results in higher base computation latency, making it ineffective in the edge-cloud setup.

We trained the BranchyNet (Teerapittayanon et al., 2017) method to represent this baseline. We create an exit classifier after the first MBConv layer in the global model and train such an exit classifier. This split results in spending 48M MACs in the base computation and then features are transferred to the global model in case the exit classifier sends the example to further layers. Note that in this case the feature cost is more than twice the input size and thus, this baseline suffers much worse from the communication delay than others. This method can always ditch the processing on the edge and send all examples to the cloud, thus achieving global accuracy with same latency and energy consumption as the on-cloud solution.

NAS+Partition Baseline. We use this baseline to represent methods such as LENS(Odema et al., 2021) that extend split-computation by performing a neural architecture search to find the best split under the communication delay constraint. To be consistent with our evolutionary search experiments, we use the MASS to find the best split with the communication latency mentioned earlier. This search space contains architectures ranging from small to large across diverse target accuracies.

Dynamic Neural Networks. This baseline refers to various recent methods (Nan and Saligrama, 2017a; Bolukbasi et al., 2017; Li et al., 2021) that propose dynamic computation and includes base-model based entropy or margin threshoding.

Hybrid Model training. We use the MCUNet model-A as the base model and the MASS-600 as the global model. We train the hybrid models using these base and global architecture pairs. Let c be the coverage of the base model, i.e. c fraction of examples are inferred on the base and $1 - c$ fraction of examples are sent to the global model. At $c = 100\%$, we obtain the on-device performance. Although hybrid

models trivially achieve on-cloud performance by sending all examples to the global model, i.e. at $c = 0\%$, our hybrid training procedure obtains the on-cloud solution at with less communication. Let ℓ_b denote the inference latency on the edge (in this case it is 200ms), while ℓ_g denote the inference latency on the cloud (in this case it is the sum of communication cost and inference latency of the global model, i.e. $2000 + 25 = 2025$ms). Thus, we can write the inference latency of the hybrid model at a coverage c as follows:

$$\text{Hybrid Latency@c} = \ell_b + (1 - c)\ell_g$$

Similarly, using we can compute the energy consumption for the hybrid inference as follows:

$$\text{Hybrid Energy@c} = \mu_b + (1 - c)\mu_g$$

where μ_b and μ_g are the edge device energy consumption for on-device and on-cloud inference. In this case, $\mu_b = 10.37$mJ and $\mu_g = 216$mJ.

Upper Bound. Let us develop an upper-bound on achievable accuracy as a function of latency. Recall from Table 12.1 that communication latency dominates processing time. Also recall the notion of coverage, c, which denotes the fraction of examples locally processed. Note that there is a one-to-one correspondence between coverage and latency of the hybrid system. Suppose α_b, α_g denotes base accuracy and cloud accuracy. The base predictor predicts $(1 - \alpha_b)$ fraction incorrectly, and among these suppose we make the reasonable assumption that the router is agnostic to which of those are correctly classified at the cloud. Then an upper bound on the target hybrid accuracy is given by the expression:

$$\text{Hybrid Acc@c} \leq \min\left\{\frac{\alpha_g - \alpha_b}{1 - \alpha_b} * (1 - c) + \alpha_b, \alpha_g\right\}$$

Notice that $c = 0$, $c = 1$, we recover global and base accuracies, and $c = \alpha_b$, is the cut-off point, namely cloud accuracy is achieved at the latency associated with the coverage c.

Intuition on the Upper Bound. The expression $c\alpha_b + (1-c)\alpha_g$ is an underestimate of the best possible accuracy - operationally, one can see this by the fact that this is achieved simply by abstaining randomly at the rate c. Quantitatively, note that $c\alpha_b + (1-c)\alpha_g \leq \alpha_g$ under the natural assumption that $\alpha_g \geq \alpha_b$, and further that $c\alpha_b + (1-c)\alpha_g = \alpha_b + (1-c)(\alpha_g - \alpha_b) \leq \alpha_b + \frac{1-c}{1-\alpha_b}(\alpha_g - \alpha_b)$. Let us explain our upper bound in detail:

- Let $c \geq \alpha_b$. If the router were perfect, then it would assign every point that the base model gets correct to the base - this gives us accuracy of 1 on an α_b fraction of inputs.
- Now we need to consider how the remaining $1 - \alpha_b$ points are assigned. Here, we make the assumption that since the router is small and cannot model the accuracy of the global model perfectly, it randomly spreads these points across the base an global models. In this case, to get overall coverage $1-c$, it must assign a fraction of $(c-\alpha_b)/(1-\alpha_b)$ of these remaining points to the base (in which case it we get none of these points wrong), and $(1-c)/(1-\alpha_b)$ of the remaining points to the global. We will now upper bound the accuracy of the global model on these remaining points.
- Here we use a second assumption - that every point that the base gets right is also gotten right by the global model. This is true to a good approximation, mainly due to the strong capacity difference between the models.
- This means that the conditional accuracy of the global model on the points that the base gets wrong is severly depressed - in particular, this is at most $\frac{\alpha_g - \alpha_b}{1-\alpha_b}$ (imagine that there were $N \gg 1$ test points. The global gets $\alpha_g N$ correct. We remove the $\alpha_b N$ that the base gets right to get that there are $(\alpha_g - \alpha_b)N$ points that the global gets right and the base doesn't. Now normalise by the $(1-\alpha_b)N$ points that the base got wrong).

- This gives us the overall accuracy

$$\alpha_b \cdot 1 + (1-\alpha_b)\left(\frac{(c-\alpha_b)}{1-\alpha_b}\cdot 0 + \frac{1-c}{1-\alpha_b}\cdot\frac{\alpha_g-\alpha_b}{1-\alpha_b}\right),$$

  which is exactly our upper bound (well, this, or α_g, whichever is smaller, since the global dominates the base and so we can never get more than α_g accuracy).

- To sum up, the upper bound arises under the assumptions that the global model strictly dominates the base model, and that conditionally on the base model getting a query point wrong, the router is unable to meaningfully discriminate between whether the global model gets queries right or wrong.

I.2 Joint Neural Architecture Search (NAS) for Hybrid Models.

Algorithm 10 Evolutionary Joint Architecture Search

1: **Input:** Data $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$, Architecture Search Space $\mathscr{A}$ (e.g. (Cai et al., 2020)).
2: **Hyper-parameters:** Number of generations G, Resource Constraint φ.
3: **Hyper-parameters:** Number of popular architecture pairs N_{pop}, .
4: **Hyper-parameters:** Number of popular parent architecture pairs N_{par}, .
5: **Initialize:** Set of popular architecture pairs $\Omega_{\text{pop}} = \{(\alpha_b^i, \alpha_g^i)\}_{i=1}^{N_{\text{pop}}}$ within constraints
6: **for** $g = 1$ **to** G **do**
7: $\quad\Omega_{\text{par}} \leftarrow N_{\text{par}}$ highest score (Eq. I.1) pairs from Ω_{pop} {Obtain the parent set Ω_{par}}
8: $\quad\Omega_{\text{child}} \leftarrow \emptyset$ {Initialize children set Ω_{child}}
9: $\quad$**for** $n = 1$ **to** N_{pop} **do**
10: $\quad\quad$Randomly pick (α_b^i, α_g^i) from Ω_{par} {Pick base & global pair}
11: $\quad\quad(\alpha_b^m, \alpha_g^m) \leftarrow \text{Mutate}(\alpha_b^i, \alpha_g^i)$ {Add mutations on width, depth, etc. }
12: $\quad\quad$Compute the oracle o_ϱ for $\theta_{\alpha_b^m}, \theta_{\alpha_g^m}$. {Compute proxy oracle supervision}
13: $\quad\quad$**if** $\mathcal{R}(o_\varrho, \theta_{\alpha_b^m}, \theta_{\alpha_g^m}) > \varphi$ **then** GOTO 9. {Repeat if constraint not satisfied}
14: $\quad\quad$Add (α_b^m, α_g^m) to Ω_{child} {Add as promising child pair}
15: $\quad\Omega_{\text{pop}} = \Omega_{\text{par}} \bigcup \Omega_{\text{child}}$ {Add children to popular set }
16: **Return :** Ω_{pop} {Return promising architecture pairs }

This section proposes a joint neural architecture search (NAS) scheme to resolve the outer max problem of (12.3). NAS methods are strongly dependent on two aspects

- *A favourable architecture space* which contains a range of architectures of varying capacities, and
- *Fast scoring proxies* that produce estimates of the overall accuracy of the best model realized by a given architecture without training the same (which would be very slow).

Given these two, NAS problems can be reduced to combinatorial optimisation and approached by any standard heuristic. In our case, we will define a favourable property of the architecture space and then proxy scores for accuracy *at a given resource consumption*. These are plugged into a simple evolutionary search to yield a concrete method.

Architecture Search Space. Our method is designed for implementation on a *Marked Architecture Search Space* (MASS). It is a set of architectures $\mathscr{A}$ such that each architecture $\alpha \in \mathscr{A}$ is associated with a known set of canonical parameters θ_α, which are known to be representative of the models under this architecture. That is, for most tasks, fine-tuning θ_α returns a good model with this architecture. Such search spaces have been proposed by, for instance, (Cai et al., 2020) and (Yu et al., 2019).

Proxy Score Function. We denote the value of program (12.4) for a given pair of architectures α_b, α_g as $\mathcal{A}(\alpha_b, \alpha_g; \varrho)$. It requires a (slow) maximisation over $b \in \alpha_b, g \in \alpha_g$, as well as training a router to attain the resource constraint. Below, we describe how to construct fast viable estimates for a MASS.

Substitute for optimisation over (b, g). MASS gives canonical models θ_α for each architecture α. Since these represent the actual performance, we take the simple approach of using the base model $\hat{b} = \theta_{\alpha_b}$ and $\hat{g} = \theta_{\alpha_g}$ to compute scores. This is the

main reason we invoke the MASS concept itself.

Substitute for optimisation over r. To quickly approximate the result of training over r, we use the routing oracle o to design a constraint-aware score.

Suppose we are given a pair of models b, g. Let $\mathcal{A}_o(b, g)$ and $\mathcal{C}_o(b, g)$ be the empirical accuracy and coverage of the oracle model $(o_{b,g}, b, g)$. If $\mathcal{C}_o \geq \mathcal{C}_\varrho$ then o can serve as the appropriate oracle. On the other hand, if $\mathcal{C}_o < \mathcal{C}_\varrho$, we observe that we can increase the coverage of this hybrid model by simply taking a subset of $o^{-1}(\{1\})$ of relative size $\mathcal{C}_\varrho - \mathcal{C}_o$, and flipping their assignment to 0.

Importantly, which points are flipped this way is irrelevant when it comes to determining the resulting accuracy - indeed, from the perspective of o, switching the label of *any* point from 1 to 0 incurs the same error. Further, the accuracy that results is an upper bound on the optimal accuracy of a hybrid system satisfying the resource constraints. This gives the following proxy score for given b, g:

$$\mathcal{S}_o(b, g; \varrho) := \mathcal{A}_o(b, g) - (\mathcal{C}_\varrho - \mathcal{C}_o(b, g))_+ .$$

Overall Score. Plugging in the canonical models into the above we get the score

$$\mathcal{S}(\alpha_b, \alpha_g; \varrho) = \mathcal{S}_o(\theta_{\alpha_b}, \theta_{\alpha_g}; C_\varrho), \tag{I.1}$$

which is effective under the assumptions that the architectures admit canonical models as above, and that the oracle accuracy $\mathcal{A}_o$ induces the same ordering on base-global pairs as the routing optimisation.

Search Algorithm. Finally, we have a scoring function and a space in hand, and so can instantiate a search algorithm. Due to its simplicity and prevalance, we propose using an evolutionary algorithm for this (Elsken et al., 2019; Liu et al., 2021). Algorithm 10 summarizes the search scheme.

I.3 Empirical Validation of Joint NAS over Hybrid Systems

End-to-end optimization across Edge/Cloud neural architectures, predictors, and rout-

ing models achieves 80% latency reduction with an Edge model 1/4*th the size of Cloud model.*

So far, we have trained hybrid models using off-the-shelf architectures not tuned to maximize hybrid performance. Here, we search for optimised base and global pairs using the proposed evolutionary search Algorithm 10. We use the OFA space (Cai et al., 2020) as our MASS. We constrain the search to operate at fixed base MACs and coverage level 70% (on average, only 30% may be queried). Recall that coverage is a surrogate for latency in high communication latency regime. After finding base and global pairs from the evolutionary search, we create hybrid models with the newly found architectures. We perform this experiment for three edge device constraints as in the previous section: 75M, 150M, and 225M. Note that the smallest model in the MASS is close to 75M MACs. Hence, we cannot search for a model below these constraints.

Figure I·1 plots the operating curves for the hybrid models found using the NAS. as well as using existing base architectures (MBV3-215M, MASS-240M). Table 12.5 shows the hybrid performance at the various coverage levels. Evolutionary search based hybrid models provide the following benefits:

- *Evolutionary search yields higher accuracy hybrid models than off-the-shelf classifiers.* As illustrated in Figure I·1, evolutionary architecture search based hybrid models pareto dominate the hybrid models using the best known base models in MBV3 and MASS family of architectures.
- *Up to* 80% *latency reduction to achieve SoTA accuracy.* Using a base with 225M MACs and 76.5% accuracy, the hybrid model achieves 79.6% accuracy at 80% coverage.
- *Hybrid models outperform entropy thresholding.*

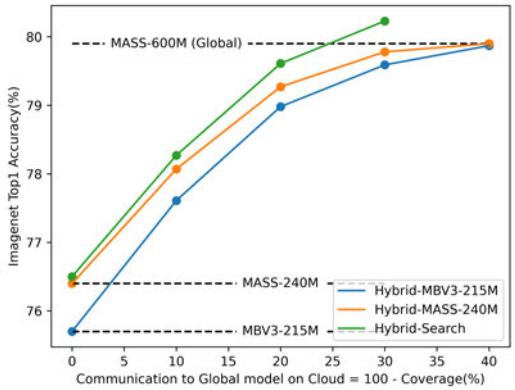


Figure I·1: Comparing hybrid models trained with Off-the-Shelf architectures vs architectures found using Joint NAS (Algorithm 10).

I.4 Algorithms

We summarise the methodological proposals as algorithms. The overall method is to begin with training a super-net in the sense of (Cai et al., 2020), for which the methods of their paper can be utilised. This produces a set of architectures $\mathscr{A}$, with associated canonical models for each $\alpha \in \mathcal{A}$. The overall procedure then is summarised as Algorithm 11. This uses the two main procedures of architecture search (Algorithm 10) and hybrid training (Algorithm 7) as subroutines, which in turn may be executed in a modular way as discussed at length in the main text.

In addition, we frequently tune a given router r and base and global models to locally trade-off resource usage levels and accuracy (which saves on retraining on each different value of ϱ that one may be interested in. This is realised by finding a value t adjusted to the constraint, and using the routing function $r(x;t) = \mathbb{1}\{r_o(x) \geq r_1(x) + t\}$. Such a t may be found as in Algorithm 12.

Algorithm 11 End-to-end Hybrid Procedure

1: **Input:** Training data $B = \{(x^i, y^i)\}_{i=1}^N$, Validation data $V = \{(x^j, y^j)\}_{j=1}^M$, resource constraint ϱ.
2: Train supernet using the method of (Cai et al., 2020). *(Architecture Search)*
3: $\mathscr{A} \leftarrow$ resulting set of algorithms.
4: $(\alpha_b, \alpha_g) \leftarrow$ output of Algorithm 10 with $V, \varrho, \mathscr{A}$.
5: Train initial models $b^0 \in \alpha_b, g^0 \in \alpha_g$ using B *(Hybrid Training)*
6: $(r, b, g) \leftarrow$ output of Algorithm 7 instantiated with B, b^0, g^0, and with appropriate hyperparameters.
7: **Return:** (r, b, g)

Algorithm 12 Tuning Routing Model

1: **Input:** Validation data $V = \{(x^j, y^j)\}_{j=1}^M$, target resource level ϱ, Hybrid model (r, b, g).
2: $\mathcal{T} \leftarrow \{r_0(x) - r_1(x) : x \in V\}$.
3: $c^* \leftarrow \min c : \mathcal{R}_r + \mathcal{R}(\alpha_b) + (1 - c)\mathcal{R}_g \leq \varrho$.
4: $t^* \leftarrow c^*$th quantile of $\mathcal{T}$.
5: **Return:** t^*.

I.5 Implementation Details

I.5.1 Hyper-parameter Settings.

We use SGD with momentum as the default optimizer in all our experiments. We initialize our hybrid models from the corresponding pre-trained models and use a learning rate of $1e-4$ for learning base and global models. We use a learning rate of $1e-2$ for learning the routing network. We decay the learning rate using a cosine learning rate scheduler. As recommended in the earlier works, we use a weight decay of $1e-5$. We set the number of epochs to be 50. We use a batch size of 256 in our experiments.

I.5.2 Model Details

Entropy Thresholding Baseline. As per recommendation in the literature (Teerapittayanon et al., 2017; Gangrade et al., 2021) we compute the entropy H of the base prediction probability distribution $b_y(x)$. This baseline allows access to a tunable threshold t. Predictions with entropy below this threshold are kept with the base model while the predictions with entropy above this threshold are sent to the cloud model. We use similar tuning as Algorithm 12 to trade-off resource usage.

Routing Model. Our routing model uses predictions from the base model and creates a 2-layer neural network from these predictions. We create meta features from these predictions to reduce the complexity of the network, by (a) adding entropy as a feature, (b) and adding correlations between top 10 predictions, resulting in a 101 dimensional input feature vector. The feed-forward network has 256 neurons in the first and 64 neurons in the second layer. The final layer outputs a two dimensional score leading to binary prediction for the routing r. Note that the routing network described in this manner contributes to less than 2% compute budget of the base model and hence its compute cost is negligible in comparison to the base and global models.

MBV3. We have used the MobileNetV3 (Howard et al., 2019) models as base in the hybrid models designed for mobile devices (see Sec. 12.3.1). We borrowed pre-trained models from publicly available implementation [4]. Table I.1 lists the performance and compute characteristics of these borrowed models.

MASS. We borrowed the pre-trained (Cai et al., 2020) models from the official public repository [5]. Table I.2 lists the accuracy, number of parameters and MACs for these models. We note that these models have been specialized by the authors with fine-tuning to achieve the reported performance.

[4] https://github.com/rwightman/pytorch-image-models
[5] https://github.com/mit-han-lab/once-for-all

Table I.1: MBV3 models in our setup.

	Top1 Accuracy	#Params	#MACs
MBV3-48	67.613	2.54M	48.3M
MBV3-143	73.3	3.99M	143.4M
MBV3-112	71.7	-	112M
MBV3-91	70.4	-	91M
MBV3-215	75.721	5.48M	215.3M

Table I.2: Once-for-All Pre-trained models in our setup.

	Accuracy	Params	MACs
MASS-600 ('flops@595M_top1@80.0_finetune@75')	79.9	9.1M	595M
MASS-482 ('flops@482M_top1@79.6_finetune@75')	79.6	9.1M	482M
MASS-389 ('flops@389M_top1@79.1_finetune@75')	79.1	8.4M	389M
MASS-240 ('LG-G8_lat@24ms_top1@76.4_finetune@25')	76.4	5.8M	230M
MASS-151 ('LG-G8_lat@16ms_top1@74.7_finetune@25')	74.6	5.8M	151M
MASS-101 ('note8_lat@31ms_top1@72.8_finetune@25')	72.8	4.6M	101M
MASS-67 ('note8_lat@22ms_top1@70.4_finetune@25')	70.4	4.3M	67M

I.6 Difference between AppealNet and our Hybrid design.

Below we highlight main difference between AppealNet ((Li et al., 2021)) and our proposal.

- AppealNet formulation does not explicitly model any coverage constraint that enables the base model to operate at a tunable coverage level. In contrast, we explicitly model a coverage penalty.

- Jointly learning the routing without any supervision is a hard problem. Instead, we relax this formulation by introducing the routing oracle that specializes in a routing network for a given base and global pair. With this oracle, the task of learning routing reduces to a binary classification problem with the routing labels obtained from the oracle. This also decouples the routing task from the base and global entanglement.

- AppealNet does not use supervision like us, and as such such strategies ultimately resemble thresholding on examples whose anticipated loss exceeds some threshold.

To see this consider Algo. 1 line 7 (Li et al., 2021) for m examples. Taking the derivative wrt q yields $\beta \frac{1}{m} \sum_x \frac{1}{1-q(0|x)} = \frac{1}{m} \sum_x \ell(f_1(x), y) - \ell(f_0(x), y)$. The RHS is the *excess loss*. For small values of $q(0|x)$, we can approximate the LHS to yield: $\beta \frac{1}{m} \sum_x (1 + q(0|x)) \approx$ Excess-Loss. Simplifying we get: $\frac{1}{m} \sum_x q(0|x) \approx \frac{1}{\beta}$Excess-Loss $- 1$. This expression suggests a relaxed objective that we should enforce the fact that examples sent to the cloud is broadly proportional to excess loss, and as such represents very weak supervision.

- In addition, we propose a neural architecture search that finds a pair of base and global architectures that optimise the hybrid accuracy at any given combined resource usage.

- Empirically, AppealNet does not have any evaluations for the ImageNet scale dataset. The closest comparison we can find is with the Tiny-ImageNet dataset (one-tenth of the size of the ImageNet). While we cannot compare the two directly, since we solve a much harder problem than Tiny-ImageNet, we can make the following observations. At 70% coverage level, for AppealNet, the minimum performance difference between the hybrid model and the global model is $\approx 1.2\%$ (see AppealNet, Fig. 5(d)), while our closest to the global in case of the MobileNet baseline is 0.3% (see our paper Table 1, row 3). Note that AppealNet performance will go down on ImageNet in comparison to Tiny-ImageNet due to the hardness of the problem.

- We compare Entropy Thresholding, AppealNet, and the proposed Hybrid model on the CIFAR-100 dataset. Table I.3 shows that entropy thresholding outperforms AppealNet. In addition, the proposed Hybrid model outperforms these baselines substantially.

Table I.3: Hybrid models for CIFAR-100 at various coverages.

Scheme	Base Acc.(%)	Accuracy(%) at Cov.		
		80%	70%	40%
Entropy	52.56	57.77	59.95	65.1
AppealNet	52.56	57.65	59.89	64.9
Hybrid-r	52.56	58.43	61.18	66.9
Hybrid-rb	52.56	59.32	62.48	67.4

I.7 Difference between LENS and our Hybrid design.

Although below we highlight main difference between LENS ((Odema et al., 2021)) and our proposal, we emphasize that LENS studies the edge-cloud interactions purely from systems perspective and as such does not dwelve into the learning aspects and the trade-off required in routing the inputs in severely resource constrained edge devices as well as their limited communication capabilities.

- (A) **Our Objective:** On large-scale tasks (such as ImageNet), for the given WiFi rate, our goal is to realize the accuracy of an arbitrarily large DNN model (deployable on the cloud) by means of a hybrid method deployed on edge. We selectively route difficult inputs on the edge to the cloud, maintaining top-accuracy, while consuming minimal energy/latency.

  Our Edge. Our edge device only has CPU or MCU compute capabilities (see Sec. 12.3.1). In addition, these edge device are severely resouce constrained, namely they only allow low powered transmission as well as low transmission rates. These constraints limits the model that can be deployed on the edge to be very low footprint. For instance, our illustrative example only has 110Kbps transmission rate (see Sec. I.1).

  This together with our desired accuracy places stringent constraints on edge to crisply learn hard-to-predict examples, and characterizes **fundamental limits of hybrid ML**.

- (B) **Circuit/Systems Prior Works (ex: LENS or Neurosurgeon).** Their goal for a given dataset is to split/partition computation of a (suitably optimized) DNN to minimize latency/energy in response to changing wireless conditions.

  LENS Edge. (Odema et al., 2021) has GPU compute capabilities on the edge device. As a result, even large DNNs can be comfortably executed on this device without a significant delay as compared to the on-cloud solution. In addition, their edge device leverage high transmission rate (up to 25Mbps). As a result, LENS explores a different setting **agnostic** to data. They leverage low transmission latency to compensate the difference in edge and cloud GPU times, motivating partitioning.

- (C) **Objective (B) is suboptimal for objective (A).** (B) requires the same network model on the edge and the cloud, which artificially constrains DNNs to fit into edge's specifications, while hoping to realize cloud-server gains on the partitioned parts. For large-scale tasks (ImageNet), high-accuracy can only be achieved by large DNNs (even with NAS optimization (Cai et al., 2020)), which are not edge-deployable, and using different architectures on edge/cloud is fundamental in (A).

- (D) **LENS baselines are too weak.** Direct comparisons are difficult due to different system choices(see (B)). Still we can note that LENS:

  – reports results on small CIFAR-10 dataset, which is not representative

  – uses VGG-16 based architecture(large DNN)—typically not edge-deployable

  – with all-cloud processing, achieves 82%, significantly lower than VGG16 published results (93.56% see `https://github.com/geifmany/cifar-vgg`);

  – with optimal NAS+partitioning gets 77% under 25% energy reduction (see Fig. 6)

In contrast, for CIFAR-10 we trained standard (tiny) models (see MCUNet model described as On-Device baseline in Sec. I.1) deployable on MCUs.

  - MCUNet (Lin et al., 2020a) is deployable on the resource constrained MCUs.
  - VGG-16 on CPU has 280X worse latency w.r.t. our model
  - With all-edge processing we get 91.7% accuracy consuming 11mJ, and 85% under 25% energy reduction.

  This shows using same model on cloud/edge is suboptimal (see (C)).

- (E) **Large-scale Task: LENS code is unavailable; Direct Evaluation is difficult.** LENS employs GP-based bayesian optimization to NAS, which is known to produce poor results (see (White et al., 2021)), which is also evident from (D). Due to lack of publicly avaiable codebase, we created a similar baseline to see the performance gap between our proposal and LENS on the illustrative example in the introduction (see Figure 12·2). We optimized NAS+partitioning method by optimizing over OFA models/architectures (Cai et al., 2020). These architectures range from small to large models across diverse target accuracies. Hybrid methods overwhelmingly dominate NAS+partitioning methods (pink in Figure I·2), again reinforcing our point (C).

I.8 Once-for-All Search Experiments

For our evolutionary search experiments, we used the (Cai et al., 2020) as the MASS. In this space, there are two MobileNetV3 backbones available: (a) one with width multiplier 1 and (b) another with width multipler 1.2. The range of models in these two space together is around 75−600M MACs. MASS allows searching over expansion factor options [3,4,6], width multiplier [1, 1.2], convolutionarl kernel sizes [3,5,7], block depths [2,3,4], and resolutions [144, 160, 176, 192, 208, 224].

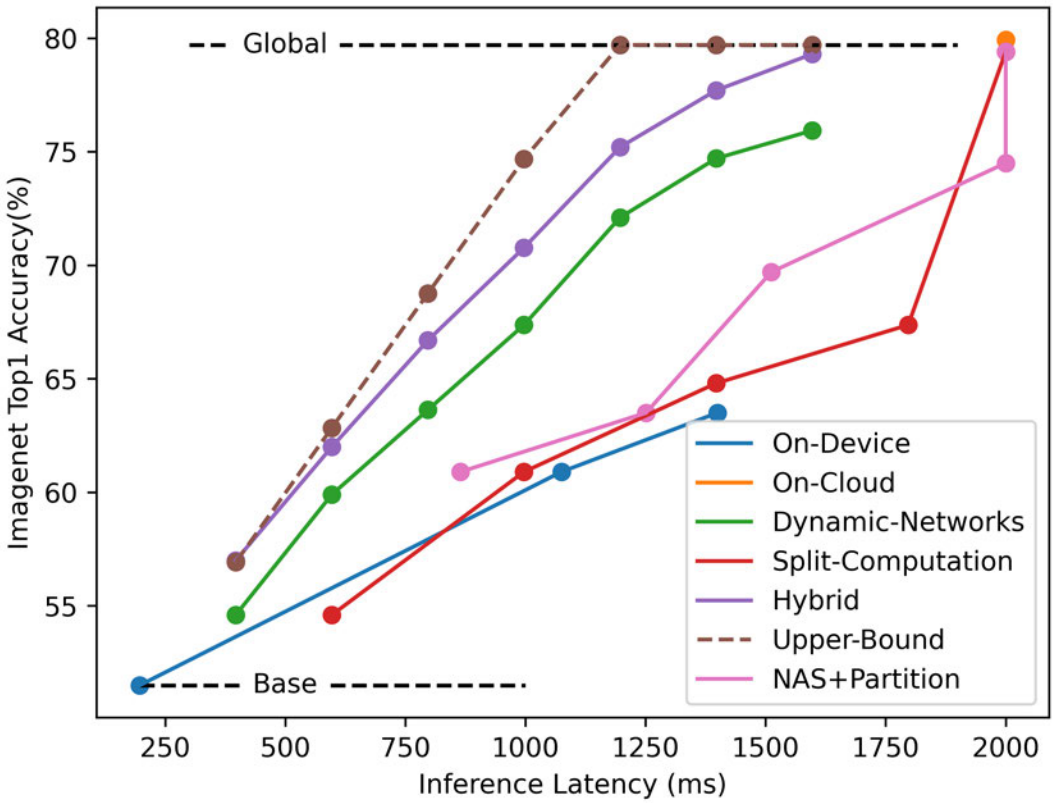


Figure I·2: Image recognition on the ImagetNet dataset: Accuracy vs Energy and Latency plot. This clearly shows that the hybrid design pareto dominates on-device as well as other baselines while getting significantly closer to the upper-bound in hybrid design.

To perform a mutation, each optimization variable is modified with probability 0.1, where modification entails re-sampling the variable from a uniform distribution over all of the options. The population size is set to 100, and the parent set size is set to 25.

Table I.4 shows the characteristics of the base and global models found using this search. Similar to (Cai et al., 2020) we fine tune these models further for 50 epochs with their setup to achieve the final accuracy.

Table I.4: Joint Evolutionary Architecture Search: Models found at three different base MAC constraints (75M, 150M, 225M).

	Top1 Accuracy	#Params	#MACs
Search-MASS-225	76.5	5.8M	225M
Search-MASS-149	74.5	5.3M	149M
Search-MASS-75	70.8	4.5M	75M

I.9 MCUNet Router Deployment Overhead

We deploy both MCUNet and our base with routing model on the MCU using the TensorFlow Lite for Microcontrollers (TFLM) runtime. Due to lack of operator support for reductions and sorting in TFLM, we replace the relevant operators with supported operations whose compute and memory complexity upperbounds the un-supported operations. Table I.5 compares the performance energy profile of the hybrid model and the baseline when deployed on the micro-controller (STM32F746) with 320KB SRAM & 1MB Flash. It clearly shows that there is a negligible cost of deploying the proposed routing scheme and only results in $< 2\%$ slowdown.

Table I.5: Profiling the on device latency and energy overhead associated with deploying the Hybrid model (MCUNet + router) as compared to deploying the plain MCUNet model on the MCU.

Model	Latency	SRAM	Energy
MCUNet	0.25368s	156708 bytes	0.1112 joules
Hybrid-MCUNet	0.25951s	158036 bytes	0.1134 joules

I.10 Ablative Experiments

I.10.1 Base and Global on same device

So far we have focused on a setup where base and global models are deployed on separate hardware. In this experiment, we deploy the base and global models on the same device. As a result, there is no communication delay in the setup. In such a setup, we can use a simpler evaluation metric for inference latency, i.e., hybrid MACs, i.e., the amount of multiply-add operations required to execute the hybrid model. We pick up an architecture family and create a hybrid model using the smallest and largest architecture. For convenience, we perform this experiment for a known family, namely MobileNetV3 (MBV3) (Howard et al., 2019). From MBV3, we pick the smallest model (48M MACs, 67.6% accuracy) as the base and largest model (215M MACs, 75.7% accuracy) as global to create the Hybrid-MBV3 model. Figure I·3 shows the hybrid model performance against the intermediate points in the MBV3 space.

Figure I·3 shows the hybrid model performance against the intermediate points in the MBV3 space as well as entropy thresholding baseline. These experiments provide evidence for the following properties of hybrid models:

- *Hybrid achieves SOTA w.r.t a global model with up to* 40% *lower latency.* Global model in MBV3 achieves 75.7% accuracy by off-loading every example to the cloud while hybrid model achieves same accuracy by sending only 60% examples to the

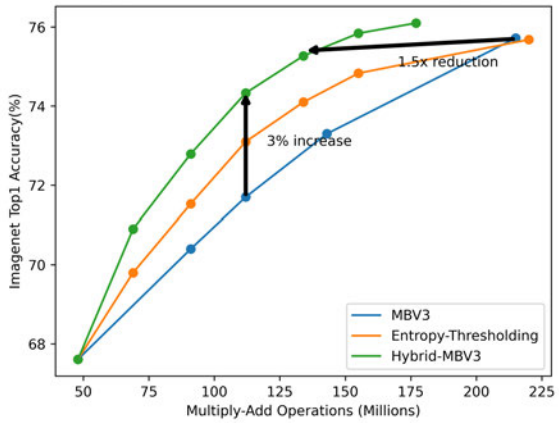


Figure I·3: MBV3: Plot for hybrid MACs vs accuracy.

cloud. Thus, saving 40% communication cost.

- *Training a Hybrid model for intermediate latency is inexpensive.* To achieve a single model at any latency, we find an architecture with this constraint and train it to non-trivial performance. Hybrid model with extreme points trades off latency for accuracy and save compute for training models for any intermediate constraint.
- *Hybrid models dominate entropy thresholding baseline used in dynamic neural networks.* Hybrid models outperform entropy thresholding at every coverage level with up to 1.5% accuracy gains.

I.10.2 Router validation

We evaluate the performance of the router against the oracle supervision and show that the router learnt using the procedure described in Sec. 12.2.1 generalizes well. For instance, while training a hybrid model with pre-trained MBV3-small and MBV3-large models, on the oracle labels, the router achieves a training accuracy of $\approx 87\%$ and this translates into a validation accuracy of $\approx 84\%$. In contrast, entropy thresholding on the validation dataset achieves $\approx 77\%$ accuracy on the oracle labels.

I.10.3 IMDb Experiments

We also learn hybrid model in the NLP domain. We train a sentiment classifier on the IMDb dataset (Maas et al., 2011). We pick-up off-the-shelf pre-trained classifiers, namely (a) albert-base-v1 (Lan et al., 2019) (11M params, 91% accuracy) as the base and (b) bert-large (Devlin et al., 2019) (340M params, 96% accuracy) as the global. We use the hidden state from the last timestep in the sequence along with the classifier logits and entropy as the feature for the routing model. In order to save computation, we learn the hybrid model by training only the router. Table I.6 shows the performance of the hybrid models and the entropy thresholding baseline at various coverage levels. It shows that hybrid models provide similar benefits on IMDb dataset. We note that, for further model footprint reduction, similar hybrid models can be constructed using efficient recurrent neural networks (Kag et al., 2020; Kag and Saligrama, 2021a) as the base model and large transformer models as the global model.

Table I.6: Hybrid models for IMDb at various coverages.

Base MACs	Base Acc.(%)	Cov.=97% Acc. (%)	Cov.=95% Acc. (%)	Cov.=93% Acc. (%)
Entropy	91	93.78	94.38	95.25
Hybrid	91	94.31	95.45	96.01

I.10.4 MCUNet experiment with EfficientNet-B7

In the main text, due to limited compute resources, we restricted our global model to be the MASS-600 (Cai et al., 2020) model. In this ablation, we explore the effect of deploying a significantly expensive model on the cloud. We choose the best performing model in the EfficientNet (Tan and Le, 2019) family, i.e., EfficientNet-B7. This model has 37B MACs and stores 66M network parameters. We borrow the im-

plementation from the timm repository [6] that achieves an accuracy of 86.5% on the ImageNet classification task. In contrast, the MASS-600 model has ≈ 600M MACs, 9.1M parameters and achieves ≈ 80% accuracy. For simplicity, we assume the cloud resources render the inference on EfficientNet-B7 to be similar to MASS-600. In this experiment, we train a hybrid-r model with MCUNet base used in the Sec. 12.3.1. Thus, we can compare the performance of the hybrid models across different global models. Table I.7 compares the accuracies obtained by the hybrid models with two different global models (MASS-600 and EfficientNet-B7). It clearly shows the following benefits:

- Deploying a better global model improves the hybrid performance and with cloud resources such large models do not affect the energy consumption on the edge device.
- Assuming that the cloud has access to large compute pool, the inference latency on the edge device does not suffer as well.
- It shows that our algorithm procedure to train hybrid models works across global models in different architecture families.

Table I.7: EfficientNet-B7 as Global model: Hybrid models on STM32F746 MCU: Accuracy achieved by different methods at various latency constraints. Base model is the MCUNet model with 12M MACs and 200ms latency.

Method	Accuracy (%) at Latency (ms)					
	200	600	1000	1400	1600	2000
On-Cloud (MASS-600)	-	-	-	-	-	79.9
On-Cloud (EfficientNet-B7)	-	-	-	-	-	86.5
On-Device	51.1	-	60.9	63.5	-	-
Hybrid (Global=MASS-600)	-	62.3	71.2	77.9	79.5	-
Hybrid (Global=EfficientNet-B7)	-	64.9	76.2	82.8	85.7	-

[6] EfficientNet-B7-ns model from https://github.com/rwightman/pytorch-image-models

I.11 Dynamic Communication Latency

Although our setup in Figure 12·2 assumes a constant communication latency, we can easily modify the setup to incorporate dynamic latency delay. Assuming the base processing latency B, global processing latency G, communication delay D, and router coverage C, we can use the following constraint on the coverage C to achieve a target average latency L,

$$B + (1 - C) * (D + G) \leq L$$

As per the main text, $B = 200$ms, $G = 25$ms and the communication delay $D = 2000$ms. In case the communication delay D has a high variation and ranges between $[D_{\min}, D_{\max}]$, we can store a lookup table on the edge device to use a different coverage threshold for different observed latency, by solving the above constraint at pre-defined communication delay intervals.

For the illustrative example, we simulate this setting by drawing communication latencies uniformly at random between $[D_{\min}, D_{\max}]$. Figure I·4 compares the dynamic communication with the constant communication cost setting. It has three dynamic ranges $[200, 2000]$, $[1000, 3000]$, $[500, 3500]$, where the last two range keep the mean communication cost same as the constant setting. It can be clearly seen that proposed hybrid models outperform all the baselines in each setting.

I.12 Algorithm Convergence Analysis

In this ablation, we show that the Algorithm 7 for training hybrid models convergences empirically. We pick up the base and global model as described in the setup in the Figure 12·3. We plot the training losses for the three components (router, base, global models) in the Figure I·5. It shows that base and global models convergence

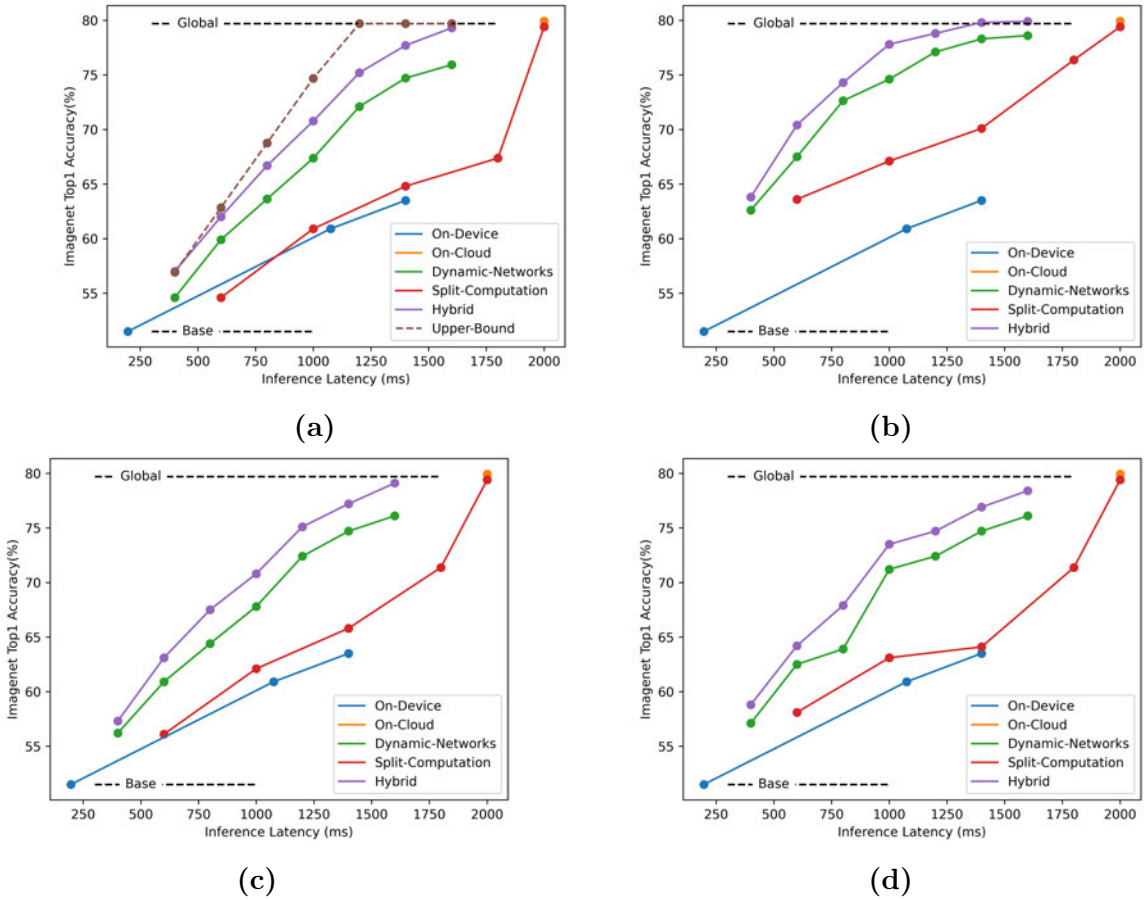
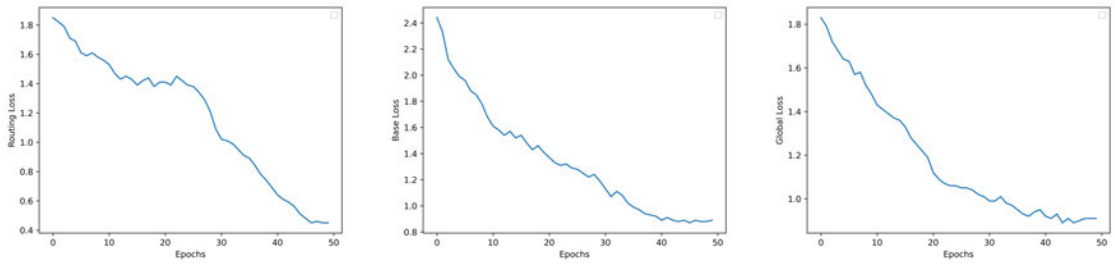


Figure I·4: Setup is same as Figure 12·2: (a) Constant Communication Latency (b) Dynamic Communication latency $[400, 2400]$ (c) Dynamic Communication latency $[1000, 3000]$ (d) Dynamic Communication latency $[500, 3500]$

to a stable loss towards the end of the training cycle.



(a) **(b)** **(c)**

Figure I·5: Algorithm Convergence. We show the training losses for the three components in the Algorithm 7: (a) router, (b) base, and (c) global model.

Appendix J

Appendix to DiSK

J.1 Details for Illustrative Example (1D Intervals)

Dataset Overview. We generate a synthetic toy dataset with one dimensional features $x \in [0, 9]$ and binary class labels $y \in \{\text{Red}, \text{Blue}\}$. We use $\sigma(x)$ to denote the sigmoid function with a scaled by parameter $\kappa > 0$, i.e., $\sigma(x) = \frac{1}{1+\exp(-\kappa x)}$.

Function Classes. Let $\mathcal{H}$ be the 1-interval function class parametrized by two variables $\{a, b\}$, i.e., for $h \in \mathcal{H}$

$$h(x; a, b) = \sigma(x - a) - \sigma(x - b); \quad 0 < a < b < 9$$

Similarly, let $\mathcal{F}$ be the 2-interval family parametrized by four variables $\{a, b, c, d\}$, i.e., for $f \in \mathcal{F}$

$$f(x; a, b, c, d) = h(x; a, b) + h(x; c, d); \quad 0 < a < b < c < d < 9$$

Note that any function in $\mathcal{H}$ behaves as an indicator for the interval (a, b). Similarly, any function in $\mathcal{F}$ behaves as an indicator for two exclusive intervals $\{(a, b), (c, d)\}$.

Data Generation. We assume that the data is generated using the function $f^* \in \mathcal{F}$ with parameters (a^*, b^*, c^*, d^*). Dataset is sampled with balanced data from both classes. We label x as red if $f^*(x) < 0.5$, otherwise we label the point as blue.

We sample 1000 i.i.d. data points as the training set and 100 data points as the test set. Figure 13·1 shows the train data. We draw an independent validation set of 100 data points for hyper-parameter tuning.

Large Capacity Teacher T belongs to the 2-interval function class $\mathcal{F}$ and is learnt with all the training data points. We learn the teacher with cross-entropy loss. We use the SGD optimizer with momentum 0.9, learning rate 0.1, weight decay 0.01, and minimize the loss for 200 epochs. Note that the teacher recovers the underlying function f^* as shown by the two intervals in Figure 13·1.

Capacity Constrained Student S belongs to the 1-interval function class $\mathcal{H}$ and it has access to all the training dataset. Note that best possible hypothesis in $\mathcal{H}$ cannot recover the performance of the function f^* and hence the student will have to settle on one of the many local minima. We show these minima as well the contour plot for the student in the Figure 13·1. We learn the student with three different loss functions (cross-entropy $\mathcal{L}_{CE}$, vanilla KD $\mathcal{L}_{KD}^{\tau,\alpha}(\mathbf{s})$, and DiSK Algorithm 7). We use similar training setup as the teacher in terms of the optimizer and training steps. For DiSK method, our guide function g has similar capacity as the student but utilizes the teacher features to learn the decision as to which points are hard-to-learn for the student. For both, KD and DiSK, we scan the α hyper-parameter over the range $\{0.0, 0.1, 0.5, 0.9, 1.0\}$. Similarly, we scan the temperature τ in the range $\{1, 2, 4\}$.

For DiSK, we scan the different hyper-parameters in the following ranges: (a) $\tau_s \in \{1, 2, 4\}$, (b) $K \in \{1, 2, 3\}$, (c) $\lambda_{\min} \in \{0.01, 0.1, 1, 5, 10\}$, (d) $\lambda_{\max} \in \{1, 5, 10, 20, 50, 100, 1000\}$, (e) Budget $\delta \in \{0.1, 0.05, 0.0\}$, and (f) $\lambda_T \in \{20, 50\}$. We replace the $\arg\min$ in Algorithm 7, with three gradient steps.

Note that although the hyper-parameter scan looks daunting, the default hyper-parameters: $\tau_s = \tau$ (teacher temperature), $K = 2$, $\lambda_{\min} = 0.1$, $\lambda_{\max} = 50$, $\delta = 0.0$ (approximate error of the global minima), $\lambda_T = 50$, work well in this setup as well as

the 2D gaussian example described below.

Vanilla KD suffers from local minima. The loss landscape of the Vanilla KD contains many local minima (see Figure 13·1(b)). Since there is a big gap between student and teacher capacity, the teacher is unable to help the student discern between these bad minima. Hence, Vanilla KD leads to one of the bad local minima with high probability (see Table 13.1).

DiSK *removes bad local minima.* In contrast, DiSK deletes harder points from the landscape and as a result settles onto the global minima for S with high probability (see Figure 13·1(c) and Table 13.1 where one cluster of blue points have been removed). Note that this also removes bad minima from the S loss landscape. Finally, DiSK learns the student that has the best performance.

J.2 Details for Illustrative Example (2D Gaussians)

Dataset Overview & Data Generation. We generate a synthetic toy dataset with two dimensional features $\mathbf{x} \in \mathbb{R}^2$ and three class labels $y \in \{\text{Red}, \text{Green}, \text{Blue}\}$. We generate six cluster centers. We assign a color to each cluster center and spread input features around these centers. Below we list the cluster centers along with their class labels.

- $(0, 0)$, Red
- $(1.5, 0)$, Blue
- $(3, 0)$, Green
- $(0, 1.5)$, Blue
- $(1.5, 1.5)$, Green
- $(3, 1.5)$, Red

Given a cluster center $\mathbf{c}$, we draw input features $\mathbf{x}$ using a Gaussian ball with radius $r = 0.05$ around the center using multi-variate Gaussian $\mathcal{N}(\mathbf{c}, r\mathbb{I})$, where $\mathbb{I}$ is the Identity matrix.

Figure 13·2.a shows the labelled data. We sample 1000 i.i.d. data points as the training set and 1000 data points as the test set with equal representation from all three classes.

Function Classes. We use two feed-forward neural networks as function classes in this example. Let $\phi(\cdot)$ denote the Batch-Norm followed by ReLU operation.

Let $\mathcal{H}$ be the two feed-forward layer neural network. Any $h \in \mathcal{H}$ can be written as

$$h(x) = W_2\phi(W_1x)$$

where $W_1 \in \mathbb{R}^{2\times 2}$ and $W_2 \in \mathbb{R}^{3\times 2}$. Note that h has only two neurons and hence a very small network.

Let $\mathcal{F}$ be the three feed-forward layer neural network. Any $f \in \mathcal{F}$ can be written as

$$f(x) = \hat{W}_3\phi(\hat{W}_2\phi(\hat{W}_1x))$$

where $\hat{W}_1 \in \mathbb{R}^{8\times 2}$, $\hat{W}_2 \in \mathbb{R}^{16\times 8}$ and $\hat{W}_3 \in \mathbb{R}^{3\times 2}$.

Note that f has 8 neurons in first and 16 neurons in the second layer. The final layer in above networks is the classifier layer that transforms the features into the class probabilities.

Large Capacity Teacher T is a 3 layer neural network with 8, 16 and 3 neurons. In between each feed-forward layer, we have batch-norm and ReLU activation non-linearity. We point out that the teacher being an over-parameterized network in this feature space, easily learns the correct decision boundary. We show this decision boundary in the Figure 13·2.a. We learn the teacher with cross-entropy loss. We use the SGD optimizer with momentum 0.9, learning rate 0.1, weight decay 0.01, and

minimize the loss for 200 epochs.

Capacity Constrained Student S is a 2 layer neural network with 2 and 3 neurons. Similar to the teacher, we have batch-norm and ReLU non-linearity in between the feed-forward layers. Since the student is severely constrained as compared to the teacher, it suffers in learning the task. Different training runs lead to some popular local minima. We show the teacher solution as well as the student local minima in Figure 13·2. For DiSK method, our guide function g has similar capacity as the student but utilizes the teacher features to learn the decision as to which points are hard-to-learn for the student. The contour plots for the student models under KD loss and DiSK loss are shown in Figure 13·2.b-13·2.c using the visualization toolkit described in (Li et al., 2018a). We following similar setup for hyper-parameter tuning as in Sec. J.1.

We see a similar result as in 1D example. KD suffers from bad local minima and converges to the global minima with only 43% of the initializations. Differently, DiSK escapes the local minima solutions and focus on the learnable part of the input space as shown in Figure 13·2.c. Our method converges to the global minima with very high probability (see Table 13.1).

J.3 Model Details

In this section, we list the model characteristics as well as their accuracy obtained using standard cross-entropy (CE) loss. Table J.1 lists all the models used in large capacity mismatch setting. While Table J.2 lists all the models in the small capacity mismatch setting. Below, we describe individual model for completeness.

Large Student-Teacher Capacity Mismatch All models in the Table J.1 belong to the same ResNet family and use the standard 'BasicBlock' as the building block. It consists of a convolutional block, followed by four residual block stages, fol-

lowed by the adaptive average pooling layer and the classifier layer. Different capacity models in this family differ only in the number of repetitions of the residual block and the number of filters in each stage. Below, we write the different of repetitions and the number of filters for the four different residual stages.

- *ResNet34* has $[64, 128, 256, 512]$ filters and repeats the 'BasicBlock' $[3, 4, 6, 3]$ times.
- *ResNet18* has $[64, 128, 256, 512]$ filters and repeats the 'BasicBlock' $[2, 2, 2, 2]$ times.
- *ResNet10* has $[64, 128, 256, 512]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.
- *ResNet10-ℓ* has $[32, 64, 128, 256]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.
- *ResNet10-m* has $[16, 32, 64, 128]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.
- *ResNet10-s* has $[8, 16, 32, 64]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.
- *ResNet10-xs* has $[8, 16, 16, 32]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.
- *ResNet10-xxs* has $[8, 8, 16, 16]$ filters and repeats the 'BasicBlock' $[1, 1, 1, 1]$ times.

Small Student-Teacher Capacity Mismatch Definitions of all models in the Table J.2 are borrowed from (Chen et al., 2022). We refer the reader to their official github repository (`https://github.com/DefangChen/SimKD.git`) for the exact definition. We trained these models on our end using the data augmentations mentioned above and found that our cross-entropy baseline as well as the vanilla KD baselines are much better than the ones reported in their work.

Guide Function Our guide function g is a three layer feed-forward network. It uses the last layer features and logits of the teacher as the input. It has 64, 128, and 1 neurons in the three layers. We include batch-norm followed by ReLU non-linearity between these layers. The final layer contains a sigmoid activation to contain the scaler output in the range $[0, 1]$.

Warm Start We note that we warm start each student model by first training them with cross entropy loss without teacher. We observe that the warm start benefits both DiSK and KD. Note that, we do not change the algorithms. We only start from

a CE pre-trained student model.

Table J.1: Models used in large capacity mismatch setting along with storage and computational requirements.

Architecture		CIFAR-100			Tiny-Imagenet		
		CE Acc.	**MACs**	**Params**	**CE Acc.**	**MACs**	**Params**
Teacher	ResNet10-ℓ	71.99	64M	1.25M	52.14	255M	1.28M
	ResNet10	75.25	253M	4.92M	56.04	1013M	5M
	ResNet18	76.56	555M	11.22M	62.48	2221M	11.27M
	ResNet34	80.46	1159M	21.32M	63.06	4637M	21.38M
Student	ResNet10-xxs	32.05	2M	13K	17.44	8M	15K
	ResNet10-xs	42.99	3M	28K	25.89	12M	31K
	ResNet10-s	52.16	4M	84K	34.65	16M	90K
	ResNet10-m	65.24	16M	320K	44.74	64M	333K

Table J.2: Models used in in small capacity mismatch setting along with storage and computational requirements.

Architecture		CIFAR-100		
		CE Acc.	**MACs**	**Params**
Teacher	ResNet32x4	81.45	1083M	7.4M
	Wide-ResNet-40-2	78.41	327M	2.25M
Student	ResNet8x4	73.89	177M	1.2M
	ShuffleNetV2	73.74	44.5M	1.4M
	Wide-ResNet-16-2	74.29	101M	700K
	Wide-ResNet-40-1	72.81	83M	570K
	MobileNetV2x2	69.24	22M	2.4M

J.4 Hyper-parameters

For both, KD and DiSK, we scan the α hyper-parameter over the range $\{0.0, 0.1, 0.5, 0.9, 1.0\}$. As per recommendations from previous works(Chen et al., 2022; Cho and Hariharan, 2019; Tung and Mori, 2019), we use $\tau = 4$ as the temperature in Eq. 13.1.

For DiSK, we scan the different hyper-parameters in the following ranges: (a) $\tau_s \in \{1, 2, 4\}$, (b) $K \in \{1, 3, 5, 10, 20, 50\}$, (c) $\lambda_{\text{min}} \in \{0.01, 0.1, 1, 5, 10\}$, (d) $\lambda_{\text{max}} \in \{1, 5, 10, 20, 50, 100, 1000\}$, (e) Budget δ within 0.2 distance from the cross-entropy trained student model's error, and (f) $\lambda_T \in \{20, 50\}$. We replace the $\arg\min$ in the Algorithm 7, with three SGD steps over the entire dataset. For all our experiments (both KD and DiSK), we use the popular cosine learning rate scheduler for the SGD

optimizer 0.1 learning rate, 0.9 momentum and $5e-4$ weight decay. We use 200 as the batch size.

Note that the default hyper-parameters: $\tau_s = \tau$ (teacher temperature), $K = 20$, $\lambda_{\min} = 0.1$, $\lambda_{\max} = 50$, $\delta =$ approximate error of the global minima (replaced by the cross-entropy error), $\lambda_T = 50$, work well in most of our experiments.

We point out that the denominator N in the budget constraint should be calibrated for the correct numerical implementation. Instead of N we use the number of wrong student predictions as the normalizer, i.e., $\sum_{i=1}^{N} y_i \neq \arg\max_y s_y(x_i)$ as this is the term that appears in the Eq. 13.2 alongside the g term in the budget constraint.

We note that we use similar parameters to train with CE in ResNet based models. Improving CE training would improve DiSK and KD as well since both are initialized with the CE trained model.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from https://www.tensorflow.org/.

Acar, D. A. E., Gangrade, A., and Saligrama, V. (2020). Budget learning via bracketing. In *International Conference on Artificial Intelligence and Statistics*, pages 4109–4119. URL: https://proceedings.mlr.press/v108/acar20a.html.

Acar, D. A. E., Zhao, Y., Matas, R., Mattina, M., Whatmough, P., and Saligrama, V. (2021). Federated learning based on dynamic regularization. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B7v4QMR6Z9w.

Altun, K., Barshan, B., and Tunçel, O. (2010). Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10):3605–3620. URL: http://dx.doi.org/10.1016/j.patcog.2010.04.019.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2012). Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Conference on Ambient Assisted Living and Home Care*, IWAAL'12, pages 216–223, Berlin, Heidelberg. Springer-Verlag. URL: http://dx.doi.org/10.1007/978-3-642-35395-6_30.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. In *ESANN : European Symposium on Artificial Neural Networks*. URL: https://www.esann.org/sites/default/files/proceedings/legacy/es2013-84.pdf.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1120–1128, New York, New York, USA. PMLR. URL: http://proceedings.mlr.press/v48/arjovsky16.html.

Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450 URL: https://arxiv.org/abs/1607.06450.

Bai, S., Kolter, J. Z., and Koltun, V. (2019a). Deep equilibrium models. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/01386bd6d8e091c2ab4c7c7de644d37b-Paper.pdf.

Bai, S., Kolter, J. Z., and Koltun, V. (2019b). Trellis networks for sequence modeling. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HyeVtoRqtQ.

Bai, S., Koltun, V., and Kolter, J. Z. (2020). Multiscale deep equilibrium models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 5238–5250. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/3812f9a59b634c2a9c574610eaba5bed-Paper.pdf.

Balduzzi, D. and Ghifary, M. (2016). Strongly-typed recurrent neural networks. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1292–1300, New York, New York, USA. PMLR. URL: https://proceedings.mlr.press/v48/balduzzi16.html.

Banbury, C., Zhou, C., Fedorov, I., Matas, R., Thakker, U., Gope, D., Janapa Reddi, V., Mattina, M., and Whatmough, P. (2021). Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3. URL: https://proceedings.mlsys.org/paper_files/paper/2021/file/c4d41d9619462c534b7b61d1f772385e-Paper.pdf.

Baradel, F., Wolf, C., and Mille, J. (2017). Pose-conditioned spatio-temporal attention for human action recognition. arXiv:1703.10106 URL: https://arxiv.org/abs/1703.10106.

Bartlett, P. L., Harvey, N., Liaw, C., and Mehrabian, A. (2019). Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *The*

Journal of Machine Learning Research, 20(63):1–17. URL: https://jmlr.org/papers/v20/17-612.html.

Bartlett, P. L. and Wegkamp, M. (2008). Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(Aug):1823–1840. URL: http://jmlr.org/papers/v9/bartlett08a.html.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013). Advances in optimizing recurrent networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. URL: https://doi.org/10.1109/ICASSP.2013.6639349.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery. URL: https://doi.org/10.1145/1553374.1553380.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. URL: http://dx.doi.org/10.1109/72.279181.

Beyer, L., Zhai, X., Royer, A., Markeeva, L., Anil, R., and Kolesnikov, A. (2021). Knowledge distillation: A good teacher is patient and consistent. URL: https://arxiv.org/abs/2106.05237.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1989). Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965. URL: https://doi.org/10.1145/76359.76371.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. (2017). Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, Proceedings of Machine Learning Research, pages 527–536. URL: http://proceedings.mlr.press/v70/bolukbasi17a.html.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends(R) in Machine Learning*, 3(1):1–122. URL: https://dl.acm.org/toc/ftml/2011/3/1.

Bradbury, J., Merity, S., Xiong, C., and Socher, R. (2016). Quasi-recurrent neural networks. *CoRR*, abs/1611.01576. URL: http://arxiv.org/abs/1611.01576.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse,

C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Journal of Mathematics and Computation*. 19(92):577–593. URL: https://doi.org/10.1090/S0025-5718-1965-0198670-6.

Bu, Y., Zou, S., and Veeravalli, V. V. (2020). Tightening mutual information-based bounds on generalization error. *IEEE Journal on Selected Areas in Information Theory*, 1(1):121–130. URL: https://buyuheng.github.io/Journal/Gen_bound.pdf.

Bucila, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA. Association for Computing Machinery. URL: https://doi.org/10.1145/1150402.1150464.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2020). Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*. URL: https://arxiv.org/pdf/1908.09791.pdf.

Cai, H., Zhu, L., and Han, S. (2019). ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*. URL: https://arxiv.org/pdf/1812.00332.pdf.

Campos, V., Jou, B., i Nieto, X. G., Torres, J., and Chang, S.-F. (2018). Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HkwVAXyCW.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*. URL: https://openaccess.thecvf.com/content/ICCV2021/html/Caron_Emerging_Properties_in_Self-Supervised_Vision_Transformers_ICCV_2021_paper.html.

Cha, J., Chun, S., Lee, K., Cho, H.-C., Park, S., Lee, Y., and Park, S. (2021). Swad: Domain generalization by seeking flat minima. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 22405–22418. Curran Associates,

Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/bcb41ccdc4363c6848a1d760f26c28a0-Paper.pdf.

Chang, B., Chen, M., Haber, E., and Chi, E. H. (2019). AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryxepo0cFX.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. (2017). Dilated recurrent neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 77–87. Curran Associates, Inc. URL: http://papers.nips.cc/paper/6613-dilated-recurrent-neural-networks.pdf.

Chen, D., Mei, J.-P., Zhang, H., Wang, C., Feng, Y., and Chen, C. (2022). Knowledge distillation with the reused teacher classifier. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11933–11942. URL: https://openaccess.thecvf.com/content/CVPR2022/html/Chen_Knowledge_Distillation_With_the_Reused_Teacher_Classifier_CVPR_2022_paper.html.

Chen, D., Mei, J.-P., Zhang, Y., Wang, C., Wang, Z., Feng, Y., and Chen, C. (2021a). Cross-layer distillation with semantic calibration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7028–7036. URL: https://doi.org/10.1609/aaai.v35i8.16865.

Chen, M., Miao, Y., Hao, Y., and Hwang, K. (2017). Narrow band internet of things. *IEEE Access*, 5:20557–20577.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021b). Evaluating large language models trained on code. URL: https://arxiv.org/pdf/2107.03374.pdf.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H.,

Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 6571–6583. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR. URL: https://proceedings.mlr.press/v119/chen20j.html.

Chen, X., Fan, H., Girshick, R., and He, K. (2020b). Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*. URL: https://arxiv.org/abs/2003.04297.

Cho, J. H. and Hariharan, B. (2019). On the efficacy of knowledge distillation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4793–4801. URL: https://openaccess.thecvf.com/content_ICCV_2019/html/Cho_On_the_Efficacy_of_Knowledge_Distillation_ICCV_2019_paper.html.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics. URL: https://aclanthology.org/W14-4012.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

Chow, C. (1957). An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, EC-6(4):247–254.

Chow, C. (1970). On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46.

Chu, G., Arikan, O., Bender, G., Wang, W., Brighton, A., Kindermans, P.-J., Liu, H., Akin, B., Gupta, S., and Howard, A. (2021). Discovering multi-hardware mobile models via architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3022–3031. URL: https://openaccess.thecvf.com/content/CVPR2021W/ECV/html/Chu_Discovering_Multi-Hardware_Mobile_Models_via_Architecture_Search_CVPRW_2021_paper.html.

Chung, J., Ahn, S., and Bengio, Y. (2016). Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704. URL: http://arxiv.org/abs/1609.01704.

Chzhen, E., Denis, C., and Hebiri, M. (2019). Minimax semi-supervised confidence sets for multi-class classification. *arXiv preprint arXiv:1904.12527*. URL: https://arxiv.org/abs/1904.12527.

Clark, K., Luong, M.-T., Manning, C. D., and Le, Q. (2018). Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics. URL: https://aclanthology.org/D18-1217.

Collins, J., Sohl-Dickstein, J., and Sussillo, D. (2017). Capacity and trainability in recurrent neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BydARw9ex.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. (2017). Recurrent batch normalization. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=r1VdcHcxx.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Cordts_The_Cityscapes_Dataset_CVPR_2016_paper.pdf.

Cortes, C., DeSalvo, G., and Mohri, M. (2016). Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer. URL: https://cs.nyu.edu/~mohri/pub/rej.pdf.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Csiszár, I. and Shields, P. (2004). Information theory and statistics: A tutorial. *Foundations and Trends® in Communications and Information Theory*, 1(4):417–528. URL: http://dx.doi.org/10.1561/0100000004.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Cubuk_AutoAugment_Learning_Augmentation_Strategies_From_Data_CVPR_2019_paper.html.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. (2020). Randaugment: Practical automated data augmentation with a reduced search space. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18613–18624. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/d85b63ef0ccb114d0a3bb7b7d808028f-Paper.pdf.

Dai, X., Kong, X., and Guo, T. (2020). *EPNet: Learning to Exit with Flexible Multi-Branch Network*, page 235–244. Association for Computing Machinery, New York, NY, USA. URL: https://doi.org/10.1145/3340531.3411973.

Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., and Gonzalez, J. E. (2021). Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16276–16285. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Dai_FBNetV3_Joint_Architecture-Recipe_Search_Using_Predictor_Pretraining_CVPR_2021_paper.html.

Dembo, R. S., Eisenstat, S. C., and Steihaug, T. (1982). Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408.

Denis, C. and Hebiri, M. (2017). Confidence sets with expected sizes for multiclass classification. *Journal of Machine Learning Research*, 18(1):3571–3598. URL: http://jmlr.org/papers/v18/16-596.html.

Denis, C. and Hebiri, M. (2019). Consistency of plug-in confidence sets for classification in semi-supervised learning. *Journal of Nonparametric Statistics*, pages 1–31. URL: https://doi.org/10.1080/10485252.2019.1689241.

Dennis, D., Acar, D. A. E., Mandikal, V., Sadasivan, V. S., Saligrama, V., Simhadri, H. V., and Jain, P. (2019). Shallow rnn: Accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems 32*, pages 12916–12926. Curran Associates, Inc. URL: http://papers.nips.cc/paper/9451-shallow-rnn-accurate-time-series-classification-on-resource-constrained-devices.pdf.

Desislavov, R., Martínez-Plumed, F., and Hernández-Orallo, J. (2023). Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857. URL: https://www.sciencedirect.com/science/article/pii/S2210537923000124.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of*

the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. URL: https://aclanthology.org/N19-1423.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*. URL: https://arxiv.org/abs/1708.04552.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157. URL: https://doi.org/10.1023/A:1007607513941.

Doimo, D., Glielmo, A., Goldt, S., and Laio, A. (2022). Redundant representations help generalization in wide neural networks. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=lC5-Ty_0FiN.

Dong, X. and Yang, Y. (2019). Network pruning via transformable architecture search. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*. Curran Associates Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/a01a0380ca3c61428c26a231f0e49a09-Paper.pdf.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=YicbFdNTTy.

Dreuning, H., Bal, H. E., and Nieuwpoort, R. V. v. (2022). Mcap: Memory-centric partitioning for large-scale pipeline-parallel dnn training. In *Euro-Par 2022: Parallel Processing: 28th International Conference on Parallel and Distributed Computing, Glasgow, UK, August 22–26, 2022, Proceedings*, page 155–170, Berlin, Heidelberg. Springer-Verlag. URL: https://doi.org/10.1007/978-3-031-12597-3_10.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

Dupont, E., Doucet, A., and Teh, Y. W. (2019). Augmented neural odes. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages

3140–3150. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*. URL: https://arxiv.org/abs/1703.11008.

El-Yaniv, R. and Wiener, Y. (2010). On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(May):1605–1641. URL: http://jmlr.org/papers/v11/el-yaniv10a.html.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017. URL: https://jmlr.org/papers/v20/18-598.html.

Erichson, N. B., Azencot, O., Queiruga, A., Hodgkinson, L., and Mahoney, M. W. (2021). Lipschitz recurrent neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=-N7PBXqOUJZ.

Filipovic, J., Madzin, M., Fousek, J., and Matyska, L. (2015). Optimizing CUDA code by kernel fusion: application on BLAS. *The Journal of Supercomputing*, 71(10):3934–3957. URL: https://doi.org/10.1007%2Fs11227-015-1483-z.

Fojo, D., Campos, V., and i Nieto, X. G. (2018). Comparing fixed and adaptive computation time for recurrent neural networks. URL: https://openreview.net/forum?id=SkZq3vyDf.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=6Tm1mposlrM.

Funahashi, K. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801 – 806. URL: http://www.sciencedirect.com/science/article/pii/S089360800580125X.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR. URL: https://proceedings.mlr.press/v48/gal16.html.

Gangrade, A., Kag, A., and Saligrama, V. (2021). Selective classification via one-sided prediction. In Banerjee, A. and Fukumizu, K., editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2179–2187. PMLR. URL: https://proceedings.mlr.press/v130/gangrade21a.html.

Geifman, Y. and El-Yaniv, R. (2017). Selective classification for deep neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 4878–4887. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/4a8423d5e91fda00bb7e46540e2b0cf1-Paper.pdf.

Geifman, Y. and El-Yaniv, R. (2019). SelectiveNet: A deep neural network with an integrated reject option. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2151–2159. PMLR. URL: https://proceedings.mlr.press/v97/geifman19a.html.

Gholami, A., Keutzer, K., and Biros, G. (2019). ANODE: unconditionally accurate memory-efficient gradients for neural odes. *CoRR*, abs/1902.10298. URL: http://arxiv.org/abs/1902.10298.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference. URL: https://arxiv.org/abs/2103.13630.

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. (2018). Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. URL: https://openaccess.thecvf.com/content_cvpr_2018_workshops/w33/html/Gholami_SqueezeNext_Hardware-Aware_Neural_CVPR_2018_paper.html.

Gong, C., He, D., Tan, X., Qin, T., Wang, L., and Liu, T.-Y. (2018). Frage: frequency-agnostic word representation. In *Advances in Neural Information Processing Systems*, pages 1334–1345. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/e555ebe0ce426f7f9b2bef0706315e0c-Paper.pdf.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819. URL: https://doi.org/10.1007%2Fs11263-021-01453-z.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983. URL: http://arxiv.org/abs/1603.08983.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1311–1320. PMLR. URL: https://proceedings.mlr.press/v70/graves17a.html.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., Piot, B., kavukcuoglu, k., Munos, R., and Valko, M. (2020). Bootstrap your own latent - a new approach to self-supervised learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

Gu, A., Goel, K., and Re, C. (2022). Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=uYLFoz1vlAC.

Gu, A., Johnson, I., Goel, K., Saab, K. K., Dao, T., Rudra, A., and Re, C. (2021). Combining recurrent, convolutional, and continuous-time models with linear state space layers. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=yWd42CWN3c.

Gu, F., Askari, A., and Ghaoui, L. E. (2020). Fenchel lifted networks: A lagrange relaxation of neural network training. In Chiappa, S. and Calandra, R., editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3362–3371. PMLR. URL: http://proceedings.mlr.press/v108/gu20a.html.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy. PMLR. URL: https://proceedings.mlr.press/v9/gutmann10a.html.

Hacohen, G. and Weinshall, D. (2019). On the power of curriculum learning in training deep networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97

of *Proceedings of Machine Learning Research*, pages 2535–2544. PMLR. URL: https://proceedings.mlr.press/v97/hacohen19a.html.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. URL: https://arxiv.org/abs/1510.00149.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. (2022). Dynamic neural networks: A survey. volume 44, pages 7436–7456, Los Alamitos, CA, USA. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3117837.

Hansen, C., Hansen, C., Alstrup, S., Simonsen, J. G., and Lioma, C. (2019). Neural speed reading with structural-jump-LSTM. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B1xf9jAqFQ.

Hardt, M., Ma, T., and Recht, B. (2018). Gradient descent learns linear dynamical systems. *Journal of Machine Learning Research*, 19(1):1025–1068. URL: https://jmlr.org/papers/v19/16-465.html.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009. URL: https://openaccess.thecvf.com/content/CVPR2022/html/He_Masked_Autoencoders_Are_Scalable_Vision_Learners_CVPR_2022_paper.html.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/He_Momentum_Contrast_for_Unsupervised_Visual_Representation_Learning_CVPR_2020_paper.html.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. URL: https://arxiv.org/abs/1606.08415.

Herbei, R. and Wegkamp, M. (2006). Classification with reject option. *The Canadian Journal of Statistics/La Revue Canadienne de Statistique*, pages 709–721.

Hermans, M. and Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc. URL: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*. URL: https://arxiv.org/abs/1503.02531.

Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen netzen. Master's thesis. Institut für Informatik, Technische Universität München. URL: http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf.

Hochreiter, S. and Schmidhuber, J. (1997a). Flat minima. *Neural computation*, 9(1):1–42.

Hochreiter, S. and Schmidhuber, J. (1997b). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hori, C., Hori, T., Lee, T.-Y., Zhang, Z., Harsham, B., Hershey, J. R., Marks, T. K., and Sumi, K. (2017). Attention-based multimodal fusion for video description. In *International Conference on Computer Vision (ICCV)*, pages 4203–4212. URL: https://openaccess.thecvf.com/content_ICCV_2017/papers/Hori_Attention-Based_Multimodal_Fusion_ICCV_2017_paper.pdf.

Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. (2019). Searching for mobilenetv3. *CoRR*, abs/1905.02244. URL: http://arxiv.org/abs/1905.02244.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861. URL: http://arxiv.org/abs/1704.04861.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html.

Huang, G. (2022). Spatially and temporally adaptive neural networks. URL: https://icml.cc/virtual/2022/workshop/13451#wse-detail-19404.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.html.

Huang, L., Zhang, C., and Zhang, H. (2020). Self-adaptive training: beyond empirical risk minimization. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19365–19376. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/e0ab531ec312161511493b002f9be2ee-Paper.pdf.

Hutomo, G. D., Kusuma, J., Ribal, A., Mahie, A. G., and Aris, N. (2019). Numerical solution of 2-d advection-diffusion equation with variable coefficient using du-fort frankel method. *Journal of Physics: Conference Series*, 1180(1):012009. URL: https://doi.org/10.1088/1742-6596/1180/1/012009.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360. URL: http://arxiv.org/abs/1602.07360.

Idelbayev, Y. and Carreira-Perpinan, M. A. (2020). Low-rank compression of neural nets: Learning the rank of each layer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8046–8056. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Idelbayev_Low-Rank_Compression_of_Neural_Nets_Learning_the_Rank_of_Each_CVPR_2020_paper.html.

Iliopoulos, F., Kontonis, V., Baykal, C., Menghani, G., Trinh, K., and Vee, E. (2022). Weighted distillation with unlabeled examples. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=M34VHvEU4NZ.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR. URL: https://proceedings.mlr.press/v37/ioffe15.html.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence*. URL: http://auai.org/uai2018/proceedings/papers/313.pdf.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html.

Jaeger, H., Lukosevicius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks : the official journal of the International Neural Network Society*, 20:335–52.

Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., and Makedon, F. (2021). A survey on contrastive self-supervised learning. *Technologies*, 9(1). URL: https://www.mdpi.com/2227-7080/9/1/2.

Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. (2017). Variable computation in recurrent neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1LVSrcge.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678.

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. (2020). Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SJgIPJBFvH.

Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., and Qu, R. (2019). A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868. URL: https://doi.org/10.1109%2Faccess.2019.2939201.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. (2017). Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1733–1741. PMLR. URL: http://proceedings.mlr.press/v70/jing17a.html.

Kag, A., Acar, D. A. E., Gangrade, A., and Saligrama, V. (2023a). Scaffolding a student to instill knowledge. In *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=N4K5ck-BTT.

Kag, A., Acar, D. A. E., and Saligrama, V. (2023b). Improving dnn generalization through data-dependent regularizers. In *Submission*. URL: https://github.com/anilkagak2/DCL_Distributionally_Constrained_Learning.

Kag, A., Fedorov, I., Gangrade, A., Whatmough, P., and Saligrama, V. (2023c). Efficient edge inference by selective query. In *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=jpR98ZdIm2q.

Kag, A. and Saligrama, V. (2021a). Time adaptive recurrent neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15149–15158. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Kag_Time_Adaptive_Recurrent_Neural_Network_CVPR_2021_paper.html.

Kag, A. and Saligrama, V. (2021b). Training recurrent neural networks via forward propagation through time. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5189–5200. PMLR. URL: https://proceedings.mlr.press/v139/kag21a.html.

Kag, A. and Saligrama, V. (2022). Condensing cnns with partial differential equations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 610–619. URL: https://openaccess.thecvf.com/content/CVPR2022/html/Kag_Condensing_CNNs_With_Partial_Differential_Equations_CVPR_2022_paper.html.

Kag, A., Wadhwa, G., Saligrama, V., and Jain, P. (2023d). Spatially interpolated inverted residual block. In *Submission*. URL: https://github.com/anilkagak2/Spatial_Interpolation.

Kag, A., Zhang, Z., and Saligrama, V. (2020). Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HylpqA4FwS.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGPLAN Notices*, 52(4):615–629. URL: https://doi.org/10.1145/3093336.3037698.

Kerg, G., Goyette, K., Puelma Touzel, M., Gidel, G., Vorontsov, E., Bengio, Y., and Lajoie, G. (2019). Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett,

R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/file/9d7099d87947faa8d07a272dd6954b80-Paper.pdf.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1oyRlYgg.

Khalil, H. (2002). *Nonlinear Systems*. Pearson Education. Prentice Hall. URL: https://books.google.com/books?id=t_d1QgAACAAJ.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/abs/1412.6980.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/5d44ee6f2c3f71b73125876103c8f6c4-Paper.pdf.

Krause, B., Kahembwe, E., Murray, I., and Renals, S. (2018). Dynamic evaluation of neural sequence models. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2766–2775. PMLR. URL: http://proceedings.mlr.press/v80/krause18a.html.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kumar, R., Rodehorst, M., Wang, J., Gu, J., and Kulis, B. (2020). Building a robust word-level wakeword verification network. In *INTERSPEECH*.

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., and Varma, M. (2018). Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi,

N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/ab013ca67cf2d50796b0c11d1b8bc95d-Paper.pdf.

Kuznetsov, V. and Mohri, M. (2015). Learning theory and algorithms for forecasting non-stationary time series. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/41f1f19176d383480afa65d325c06ed0-Paper.pdf.

Kuznetsov, V. and Mohri, M. (2016). Time series prediction and online learning. In Feldman, V., Rakhlin, A., and Shamir, O., editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1190–1213, Columbia University, New York, New York, USA. PMLR. URL: https://proceedings.mlr.press/v49/kuznetsov16.html.

Laine, S. and Aila, T. (2017). Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BJ6oOfqge.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 6402–6413. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942. URL: http://arxiv.org/abs/1909.11942.

Larsson, G., Maire, M., and Shakhnarovich, G. (2017). Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1VaB4cex.

Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. Dept. of Statistics, Stanford University. URL: https://vision.stanford.edu/teaching/cs231n/reports/2015/pdfs/yle_project.pdf.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Lei, J. (2014). Classification with confidence. *Biometrika*, 101(4):755–769. URL: https://doi.org/10.1093/biomet/asu038.

Lei, T., Zhang, Y., Wang, S. I., Dai, H., and Artzi, Y. (2018). Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4470–4481. Association for Computational Linguistics. URL: https://aclanthology.org/D18-1477.

Lezcano-Casado, M. and Martínez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3794–3803. PMLR. URL: http://proceedings.mlr.press/v97/lezcano-casado19a.html.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018a). Visualizing the loss landscape of neural nets. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf.

Li, H., Zhang, H., Qi, X., Yang, R., and Huang, G. (2019a). Improved techniques for training adaptive deep networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. URL: https://openaccess.thecvf.com/content_ICCV_2019/html/Li_Improved_Techniques_for_Training_Adaptive_Deep_Networks_ICCV_2019_paper.html.

Li, M., Li, Y., Tian, Y., Jiang, L., and Xu, Q. (2021). Appealnet: An efficient and highly-accurate edge/cloud collaborative architecture for dnn inference. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 409–414. URL: https://ieeexplore.ieee.org/document/9586176.

Li, S., Li, W., Cook, C., Gao, Y., and Zhu, C. (2019b). Deep independently recurrent neural network (indrnn). URL: https://arxiv.org/abs/1910.06251.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. (2018b). Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5457–5466. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Li_Independently_Recurrent_Neural_CVPR_2018_paper.html.

Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019.

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403. URL: https://www.sciencedirect.com/science/article/pii/S0925231221010894.

Lin, J., Chen, W.-M., Cai, H., Gan, C., and Han, S. (2021). Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 2346–2358. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2021/hash/1371bccec2447b5aa6d96d2a540fb401-Abstract.html.

Lin, J., Chen, W.-M., Lin, Y., cohn, j., Gan, C., and Han, S. (2020a). Mcunet: Tiny deep learning on iot devices. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11711–11722. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2020/hash/86c51678350f656dcc7f490a43946ee5-Abstract.html.

Lin, T., Jin, C., and Jordan, M. (2020b). On gradient descent ascent for nonconvex-concave minimax problems. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6083–6093. PMLR. URL: https://proceedings.mlr.press/v119/lin20a.html.

Linsley, D., Karkada Ashok, A., Govindarajan, L. N., Liu, R., and Serre, T. (2020). Stable and expressive recurrent vision models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10456–10467. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2020/file/766d856ef1a6b02f93d894415e6bfa0e-Paper.pdf.

Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. URL: https://arxiv.org/abs/1506.00019.

Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. (2017a). Progressive neural architecture search. *CoRR*, abs/1712.00559. URL: http://arxiv.org/abs/1712.00559.

Liu, H., Simonyan, K., and Yang, Y. (2019a). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1eYHoC5FX.

Liu, M., Liu, H., and Chen, C. (2017b). Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recognition*, 68:346 – 362. URL: http://www.sciencedirect.com/science/article/pii/S0031320317300936.

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Tan, K. C. (2021). A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*.

Liu, Z., Wang, Z., Liang, P. P., Salakhutdinov, R. R., Morency, L.-P., and Ueda, M. (2019b). Deep gamblers: Learning to abstain with portfolio theory. In *Advances in Neural Information Processing Systems*, pages 10622–10632. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/0c4b1eeb45c90b52bfb9d07943d855ab-Paper.pdf.

Lopez-Paz, D., Bottou, L., Schölkopf, B., and Vapnik, V. (2016). Unifying distillation and privileged information. In *International Conference on Learning Representations (Poster)*. URL: http://arxiv.org/abs/1511.03643.

Luo, W. and Yu, F. (2019). Recurrent highway networks with grouped auxiliary memory. *IEEE Access*, 7:182037–182049. URL: https://ieeexplore.ieee.org/document/8932404.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*. URL: https://openaccess.thecvf.com/content_ECCV_2018/html/Ningning_Light-weight_CNN_Architecture_ECCV_2018_paper.html.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics. URL: http://www.aclweb.org/anthology/P11-1015.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040.

McAuley, J. and Leskovec, J. (2013). Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 165–172, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2507157.2507163.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., and Kubica, J. (2013). Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., and Graves, A. (2021). Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=q3KSThy2GwB.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SyyGPP0TZ.

Mhammedi, Z., Hellicar, A., Rahman, A., and Bailey, J. (2017). Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2401–2409. PMLR. URL: http://proceedings.mlr.press/v70/mhammedi17a.htm.

Miller, J. and Hardt, M. (2019). Stable recurrent models. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Hygxb2CqKm.

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2022). Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press. ISBN: 9780262039406.

Mujika, A., Meier, F., and Steger, A. (2017). Fast-slow recurrent neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/e4a93f0332b2519177ed55741ea4e5e7-Paper.pdf.

Mujika, A., Meier, F., and Steger, A. (2018). Approximating real-time recurrent learning with random kronecker factors. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 6594–6603. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/dba132f6ab6a3e3d17a8d59e82105f4c-Paper.pdf.

Müller, R., Kornblith, S., and Hinton, G. (2019). When does label smoothing help? In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates

Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/f1748d6b0fd9d439f71450117eba2725-Paper.pdf.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress. URL: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf.

Nan, F. and Saligrama, V. (2017a). Adaptive classification for prediction under a budget. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 4727–4737. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/d9ff90f4000eacd3a6c9cb27f78994cf-Paper.pdf.

Nan, F. and Saligrama, V. (2017b). Dynamic model selection for prediction under a budget. *arXiv preprint arXiv:1704.07505*. URL: https://arxiv.org/abs/1704.07505.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

Neyshabur, B. (2017). Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*. URL: https://arxiv.org/abs/1709.01953.

Neyshabur, B., Tomioka, R., and Srebro, N. (2015). Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401. PMLR.

Ni, C., Charoenphakdee, N., Honda, J., and Sugiyama, M. (2019). On the calibration of multiclass classification with rejection. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 2586–2596. Curran Associates, Inc. URL: http://papers.nips.cc/paper/8527-on-the-calibration-of-multiclass-classification-with-rejection.pdf.

Niu, M. Y., Horesh, L., and Chuang, I. (2019). Recurrent neural networks in the eye of differential equations. *arXiv preprint arXiv:1904.12933*. URL: https://arxiv.org/abs/1904.12933.

Odema, M., Rashid, N., Demirel, B. U., and Faruque, M. A. A. (2021). Lens: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 403–408.

Ollivier, Y. and Charpiat, G. (2015). Training recurrent networks online without backtracking. *CoRR*, abs/1507.07680. URL: http://arxiv.org/abs/1507.07680.

OpenAI (2023). Gpt-4 technical report. arXiv:2303.08774 URL: https://arxiv.org/abs/2303.08774.

Park, E., Kim, D., Kim, S., Kim, Y.-D., Kim, G., Yoon, S., and Yoo, S. (2015). Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 124–132.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013a). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*. URL: https://arxiv.org/abs/1312.6026.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR. URL: https://proceedings.mlr.press/v28/pascanu13.html.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., and Dean, J. (2021). Carbon emissions and large neural network training. URL: https://arxiv.org/abs/2104.10350.

Pennington, J., Schoenholz, S., and Ganguli, S. (2017). Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4785–4795.

Rabanser, S., Thudi, A., Hamidieh, K., Dziedzic, A., and Papernot, N. (2022). Selective classification via neural network training dynamics. URL: https://arxiv.org/abs/2205.13532.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. URL: https://arxiv.org/abs/2212.04356.

Ramaswamy, H. G., Tewari, A., and Agarwal, S. (2018). Consistent algorithms for multiclass classification with an abstain option. *Electronic Journal of Statistics*, 12(1):530–554. URL: https://doi.org/10.1214/17-EJS1388.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. URL: https://arxiv.org/abs/2204.06125.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4405.

Rigollet, P. and Tong, X. (2011). Neyman-pearson classification, convexity and stochastic constraints. *Journal of Machine Learning Research*, 12(Oct):2831–2855.

Romero, A., Kahou, S. E., Montréal, P., Bengio, Y., Montréal, U. D., Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/abs/1412.6550.

Rosenblatt, F. (1962). *Principles of neurodynamics.* Spartan Books, Washington, D.C.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. (2019). Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907. URL: http://arxiv.org/abs/1907.03907.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. URL: https://arxiv.org/abs/1609.04747.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

Rusch, T. K. and Mishra, S. (2021a). Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=F3s69XzWOia.

Rusch, T. K. and Mishra, S. (2021b). Unicornn: A recurrent model for learning very long time dependencies. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9168–9178. PMLR. URL: http://proceedings.mlr.press/v139/rusch21a.html.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Ruthotto, L. and Haber, E. (2020). Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*. 62, pages:352–364. URL: https://doi.org/10.1007/s10851-019-00903-1.

Sadinle, M., Lei, J., and Wasserman, L. (2019). Least ambiguous set-valued classifiers with bounded error levels. *Journal of the American Statistical Association*, 114(525):223–234. URL: https://doi.org/10.1080/01621459.2017.1395341.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding. URL: https://arxiv.org/abs/2205.11487.

Sajjadi, M., Javanmardi, M., and Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/30ef30b64204a3088a26bc2e6ecf7602-Paper.pdf.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823. URL: https://openaccess.thecvf.com/content_cvpr_2015/html/Schroff_FaceNet_A_Unified_2015_CVPR_paper.html.

Sennrich, R., Haddow, B., and Birch, A. (2016). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 86–96, Berlin, Germany. Association for Computational Linguistics. URL: https://aclanthology.org/P16-1009.

Shahroudy, A., Liu, J., Ng, T.-T., and Wang, G. (2016). Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press.

Shekhar, S., Ghavamzadeh, M., and Javidi, T. (2019). Binary classification with bounded abstention rate. *arXiv preprint arXiv:1905.09561*. URL: https://arxiv.org/abs/1905.09561.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*. URL: http://arxiv.org/abs/1409.1556.

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.

Sohn, K., Berthelot, D., Carlini, N., Zhang, Z., Zhang, H., Raffel, C. A., Cubuk, E. D., Kurakin, A., and Li, C.-L. (2020). Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 596–608. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/06964dce9addb1c5cb5d6e3d9838f733-Paper.pdf.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *CoRR*, abs/1505.00387. URL: http://arxiv.org/abs/1505.00387.

Stanton, S. D., Izmailov, P., Kirichenko, P., Alemi, A. A., and Wilson, A. G. (2021). Does knowledge distillation really work? In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=7J-fKoXiReA.

Sun, K. and Sun, A. (2021). Dual descent alm and admm. arXiv:2109.13214 URL: https://arxiv.org/abs/2109.13214.

Sun, Y., Zhang, L., and Schaeffer, H. (2020). NeuPDE: Neural network based ordinary and partial differential equations for modeling time-dependent data. In Lu, J. and Ward, R., editors, *Proceedings of The First Mathematical and Scientific Machine Learning Conference*, volume 107 of *Proceedings of Machine Learning Research*, pages 352–372, Princeton University, Princeton, NJ, USA. PMLR. URL: http://proceedings.mlr.press/v107/sun20a.html.

Talathi, S. S. and Vartak, A. (2015). Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*. URL: https://arxiv.org/abs/1511.03771.

Tallec, C. and Ollivier, Y. (2018). Unbiased online recurrent optimization. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rJQDjk-0b.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.html.

Tan, M. and Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA. PMLR. URL: http://proceedings.mlr.press/v97/tan19a.html.

Tan, M. and Le, Q. (2021). Efficientnetv2: Smaller models and faster training. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR. URL: https://proceedings.mlr.press/v139/tan21a.html.

Tarvainen, A. and Valpola, H. (2017). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2022). Efficient transformers: A survey. *ACM Computing Survey*, 55(6). URL: https://doi.org/10.1145/3530811.

Teerapittayanon, S., McDanel, B., and Kung, H. T. (2017). Branchynet: Fast inference via early exiting from deep neural networks. URL: https://arxiv.org/abs/1709.01686.

The Theano Development Team, Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Bleecher Snyder, J., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P.-L., Cho, K., Chorowski, J., Christiano, P., Cooijmans, T., Côté, M.-A., Côté, M., Courville, A., Dauphin, Y. N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Ebrahimi Kahou, S., Erhan, D., Fan, Z., Firat, O., Germain, M., Glorot, X., Goodfellow, I., Graham, M., Gulcehre, C., Hamel, P., Harlouchet, I., Heng, J.-P., Hidasi, B., Honari, S., Jain, A., Jean, S., Jia, K., Korobov, M., Kulkarni, V., Lamb, A., Lamblin, P., Larsen, E., Laurent, C., Lee, S., Lefrancois, S., Lemieux, S., Léonard, N., Lin, Z., Livezey, J. A., Lorenz, C., Lowin, J., Ma, Q., Manzagol, P.-A., Mastropietro, O., McGibbon, R. T., Memisevic, R., van Merriënboer, B., Michalski, V., Mirza, M., Orlandi, A., Pal, C., Pascanu, R., Pezeshki, M., Raffel, C., Renshaw, D., Rocklin, M., Romero, A., Roth, M., Sadowski, P., Salvatier, J., Savard, F., Schlüter, J., Schulman, J., Schwartz, G., Vlad Serban, I., Serdyuk, D., Shabanian, S., Simon, É., Spieckermann, S., Ramana Subramanyam, S., Sygnowski, J., Tanguay, J., van Tulder, G., Turian, J., Urban, S., Vincent, P., Visin, F., de Vries, H., Warde-Farley, D., Webb, D. J., Willson, M., Xu, K., Xue, L., Yao, L., Zhang, S., and Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, page arXiv:1605.02688. URL: https://ui.adsabs.harvard.edu/abs/2016arXiv160502688T.

Tian, Y., Krishnan, D., and Isola, P. (2020). Contrastive multiview coding. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI*, page 776–794, Berlin, Heidelberg. Springer-Verlag. URL: https://doi.org/10.1007/978-3-030-58621-8_45.

Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Tong, X. (2013). A plug-in approach to neyman-pearson classification. *Journal of Machine Learning Research*, 14(56):3011–3040. URL: http://jmlr.org/papers/v14/tong13a.html.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. (2021). Training data-efficient image transformers & distillation through attention. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR. URL: https://proceedings.mlr.press/v139/touvron21a.html.

Trinh, T., Dai, A., Luong, T., and Le, Q. (2018). Learning longer-term dependencies in RNNs with auxiliary losses. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4965–4974. PMLR. URL: http://proceedings.mlr.press/v80/trinh18a.html.

Tung, F. and Mori, G. (2019). Similarity-preserving knowledge distillation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1365–1374, Los Alamitos, CA, USA. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/ICCV.2019.00145.

van den Oord, A., Li, Y., and Vinyals, O. (2019). Representation learning with contrastive predictive coding. URL: https://arxiv.org/pdf/1807.03748.pdf.

Vapnik, V. and Izmailov, R. (2015). Learning using privileged information: Similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16(61):2023–2049. URL: http://jmlr.org/papers/v16/vapnik15b.html.

Vapnik, V. N. (2000). *The Nature of Statistical Learning Theory*. Springer, New York. ISBN: 9781441931603 URL: https://doi.org/10.1007/978-1-4757-3264-1.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Volpi, R., Namkoong, H., Sener, O., Duchi, J. C., Murino, V., and Savarese, S. (2018). Generalizing to unseen domains via adversarial data augmentation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/1d94108e907bb8311d8802b48fd54b4a-Paper.pdf.

Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning (ICML)*, pages 3570–3578. URL: http://proceedings.mlr.press/v70/vorontsov17a.html.

Wang, W. and Howard, A. (2021). Mosaic: Mobile segmentation via decoding aggregated information and encoded context. URL: https://arxiv.org/abs/2112.11623.

Wang, X., Luo, Y., Crankshaw, D., Tumanov, A., Yu, F., and Gonzalez, J. E. (2018). Idk cascades: Fast deep learning by learning not to overthink. URL: https://arxiv.org/abs/1706.00885.

Wang, Y., Yao, Q., Kwok, J., and Ni, L. M. (2020). Generalizing from a few examples: A survey on few-shot learning. URL: https://arxiv.org/abs/1904.05046.

Warden, P. (2017). Speech commands: A public dataset for single-word speech recognition. Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.

Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv e-prints*, page arXiv:1804.03209. URL: https://arxiv.org/abs/1804.03209.

Wegkamp, M. (2007). Lasso type classifiers with a reject option. *Electronic Journal of Statistics*, 1:155–168. URL: https://doi.org/10.1214/07-EJS058.

Wegkamp, M. and Yuan, M. (2011). Support vector machines with a reject option. *Bernoulli*, 17(4):1368–1385. URL: https://doi.org/10.3150/10-BEJ320.

Weiss, N., Holmes, P., and Hardy, M. (2005). *A Course in Probability*. Pearson Addison Wesley.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

White, C., Neiswanger, W., and Savani, Y. (2021). Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*. URL: https://arxiv.org/abs/1910.11858.

Wiener, Y. and El-Yaniv, R. (2011). Agnostic selective classification. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24, pages 1665–1673. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/4b6538a44a1dfdc2b83477cd76dee98e-Paper.pdf.

Wightman, R. (2019). Pytorch image models. https://github.com/rwightman/pytorch-image-models.

Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, 2(4):490–501. URL: https://doi.org/10.1162/neco.1990.2.4.490.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. (2016). Full-capacity unitary recurrent neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4880–4888. Curran Associates, Inc. URL: http://papers.nips.cc/paper/6327-full-capacity-unitary-recurrent-neural-networks.pdf.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742. URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Wu_FBNet_Hardware-Aware_Efficient_ConvNet_Design_via_Differentiable_Neural_Architecture_Search_CVPR_2019_paper.pdf.

Wu, Y. and He, K. (2018). Group normalization. URL: https://openaccess.thecvf.com/content_ECCV_2018/html/Yuxin_Wu_Group_Normalization_ECCV_2018_paper.html.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*. URL: https://arxiv.org/abs/1609.08144.

Wu, Z., Xiong, Y., Yu, S. X., and Lin, D. (2018). Unsupervised feature learning via non-parametric instance discrimination. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3733–3742. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Wu_Unsupervised_Feature_Learning_CVPR_2018_paper.html.

Xie, Q., Dai, Z., Hovy, E., Luong, T., and Le, Q. (2020). Unsupervised data augmentation for consistency training. *Advances in neural information processing systems*, 33:6256–6268. URL: https://proceedings.neurips.cc/paper/2020/file/44feb0096faa8326192570788b38c1d1-Paper.pdf.

Xu, Z. E., Kusner, M. J., Weinberger, K. Q., Chen, M., and Chapelle, O. (2014). Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research*, 15:2113–2144. URL: http://jmlr.org/papers/v15/xu14a.html.

Yang, X., Song, Z., King, I., and Xu, Z. (2022). A survey on deep semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20. URL: https://arxiv.org/abs/2103.00550.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. (2018). Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HkwZSG-CZ.

Yelp, I. (2017). Yelp dataset challenge. URL: https://www.yelp.com/dataset/challenge.

You, Y., Gitman, I., and Ginsburg, B. (2017). Large batch training of convolutional networks. URL: https://arxiv.org/abs/1708.03888.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. (2020). Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Syx4wnEtvH.

Yu, A. W., Lee, H., and Le, Q. V. (2017). Learning to skim text. *CoRR*, abs/1704.06877. URL: http://arxiv.org/abs/1704.06877.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. (2019). Slimmable neural networks. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1gMCsAqY7.

Yuan, M. and Wegkamp, M. (2010). Classification methods with reject option based on convex risk minimization. *Journal of Machine Learning Research*, 11(5):111–130. URL: http://jmlr.org/papers/v11/yuan10a.html.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision (ICCV)*. URL: https://openaccess.thecvf.com/content_ICCV_2019/html/Yun_CutMix_Regularization_Strategy_to_Train_Strong_Classifiers_With_Localizable_Features_ICCV_2019_paper.html.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *British Machine Vision Conference (BMVC)*. URL: https://bmva-archive.org.uk/bmvc/2016/papers/paper087/index.html.

Zagoruyko, S. and Komodakis, N. (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Sks9_ajex.

Zeng, X. and Martinez, T. R. (2000). Using a neural networks to approximate an ensemble of classifiers. In *Neural Processing Letters*, page 2000. URL: https://doi.org/10.1023/A:1026530200837.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Sy8gdB9xx.

Zhang, J., Lei, Q., and Dhillon, I. (2018a). Stabilizing gradients for deep neural networks via efficient SVD parameterization. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5806–5814. PMLR. URL: http://proceedings.mlr.press/v80/zhang18g.html.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018b). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_ShuffleNet_An_Extremely_CVPR_2018_paper.html.

Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2018c). Deep mutual learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4320–4328. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_Deep_Mutual_Learning_CVPR_2018_paper.html.

Zhao, Z.-Q., Zheng, P., Xu, S.-T., and Wu, X. (2019). Object detection with deep learning: A review. URL: https://doi.org/10.1109/tnnls.2018.2876865.

Zhou, T., Wang, S., and Bilmes, J. (2020). Curriculum learning by dynamic instance hardness. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8602–8613. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2020/file/62000dee5a05a6a71de3a6127a68778a-Paper.pdf.

Zhu, P., Acar, D. A. E., Feng, N., Jain, P., and Saligrama, V. (2019). Cost aware inference for iot devices. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2770–2779. PMLR. URL: https://proceedings.mlr.press/v89/zhu19d.html.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4189–4198. PMLR. URL: https://proceedings.mlr.press/v70/zilly17a.html.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578. URL: http://arxiv.org/abs/1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html.

CURRICULUM VITAE

[redacted]

[redacted]

[redacted]