Applications and Methodologies of Machine Learning on Sequences

Aidan Nicholas Gomez

St. John's College

University of Oxford

Thesis submitted for the degree of

Doctor of Philosophy

Trinity 2023

For my father.

Acknowledgements

This thesis is the product of the ardour of hundreds of individuals who poured into me their time, knowledge, and affection. I am only able to write a few of their names here.

First and foremost, I thank my father, mother, and brother – the ones who raised me. My father taught me my love of science and truth. My mother taught me my love of art and beauty. My bother taught me my love of fun and good humour.

I thank my advisor Yarin Gal, who spent countless hours entertaining my awful ideas and helped point me in a productive direction. He afforded me the time, space, and opportunity to pursue every idea, adventure, and folly with unwavering support and encouragement.

I thank my lab mates who motivated and inspired me with many ideas and discussions.

I thank my cofounders Nick and Ivan, who covered for me at work while I worked on this thesis.

I thank Dameli, who has graciously and with immense patience endured my stress, my days and sometimes weeks away, and my working late into the night.

I thank my friends in Oxford who made my time here a joy. In particular, thank you Helio, Alessandro, Ania, Anna, and Claudia.

I thank my friends who were a bit further away but came to visit me or kept in contact from afar. In particular, thank you Yazan, Justin, Robyn, Victoria, Aaron, Aaron, and Nathan.

I thank Oxford – the university, the city, and their people – for making such a wonderful place to call home.

Contents

Chapter 1

Introduction

A rapidly emerging trend in machine learning is the consolidation of the field around a limited set of methods tailored towards scalability, versatility, and simplicity. The surge of sequence models is one of the most prominent architectural shifts that has been driven as a result of these priorities.

The growth in popularity of these models – alongside the realisation that larger models tend to exhibit better performance on tasks, as well as entirely new capabilities – has led to massive investment in systems to support training at scale. These easy-to-use and extensible toolkits for sequence models enabled rapid experimentation across the machine learning research community and has given rise to new instantiations of sequence models that extended beyond language and into domains like vision, audio, graph structured data, and tabular data.

Alongside the investment in systems and infrastructure, there has been an investment in methods for the collection and curation of large-scale datasets of sequence data. A similar observation of scale positively influencing performance on tasks and capabilities has been made in terms of the scale and cleanliness of data collected.

Key to realising the progress of recent years has been the pursuit of answering two questions:

1. How do we scale up our models?
2. How do we scale up our data?

This thesis contributes new methods for scaling sequence models as well as their data, and pursues their application on domains outside of the common webtext setting. It includes the following eight works, four of which have been published previously:

- Chapter 3 – Learning Sparse Networks Using Targeted Dropout

- Chapter 4 – Training Neural Networks in Low-Rank Subspaces
- Chapter 5 – SliceOut: An Efficient Dropout Alternative
- Chapter 6 – Asynchronous Reversible Component Networks
- Chapter 7 – Interlocking Backpropagation (JMLR, 2022)
- Chapter 8 – Inter-example Attention for Transformers Applied to Tabular Data (NeurIPS, 2021)
- Chapter 9 – Prioritized training on points that are learnable, worth learning, and not yet learned (ICML, 2022)
- Chapter 10 – Disease Variant Prediction with Deep Generative Models of Evolutionary Data (Nature, 2021)

The chapters are structured so that we begin by introducing methods focused on increasing the efficiency of training and serving to support scaling models (Chapters 3-7), and we close with methods focused on data efficiency and new data domains to support scaling data (Chapters 8-10). Most chapters are entirely distinct and introduce independent techniques, however Chapters 6 and 7 were written in sequence and the former inspired the development of the latter, which is discussed in the conclusion of the former and introduction of the latter.

The intent of this body of work is to accelerate the pace of compelling and coherent artificial intelligence by providing a collection of methods that can be exploited to enable faster and more effective scaling of both model and data.

Chapter 2

Background

In this chapter, I will give a brief tour of how the Transformer (Vaswani et al., 2017b) came to be and why the present-day preoccupation of the field has turned towards large models and unsupervised learning – setting the stage for the contributions of this thesis and motivating focus on advancing methods in modelling and data in the sequence learning regime.

Many commonly used modalities of data are naturally sequential in nature. Language, audio, video, and financial data all have sequential structure as a consequence of their temporality. However, up until fairly recently, the sequential structure of this data would most often be discarded and models would simply learn to perform tasks based on features that neglect item ordering.

These models – referred to as "bag-of-words" models because of all the words in a sequence being tossed into a bad and shaken up, losing their ordering – had been a difficult baseline to beat. Their simplicity and speed lent themselves extremely well to production scenarios with low latency tolerances, and so, as alternative methods that did incorporate temporal information arose they had to prove significant gains in task performance to justify the increase in computational cost associated with modelling these temporal dependencies.

In a domain such as language – where the placement or misplacement of a word can fundamentally alter the meaning of a sentence – bag-of-words models inevitably were not able to compete with modelling techniques that accounted for temporal dependence. The first category of neural network to incorporate temporal dependence and win popularity in the field were Recurrent Neural Networks (RNNs), and the most popular architecture within this category is the Long Short-Term Memory unit (LSTM) (Hochreiter & Schmidhuber, 1997). LSTMs set new records on many language tasks including classification and modelling, and established themselves as the standard model for language, audio, and time series data.

Despite these promising results, LSTMs had significant limitations that hindered their use and utility. First, LSTMs were tedious to train; they were sensitive to hyperparameters such as learning rate and initialisation scale and exhibited significant training instability; models would frequently diverge during training if parameters weren't properly chosen, or wouldn't train due to the vanishing gradients problem caused by the deep compute graphs of backpropagation through time. Second, the architecture cannot be naively parallelised across the length of an input sequence, instead processing each item in the sequence one-by-one, in order. Third, the architecture lacks a mechanism for directly referencing prior elements of a sequence, instead relying on an internal memory that the model must learn to utilise. While methods arose that sought to alleviate each of these limitations, their adoption was inconsistent and/or inadequate to fully resolve the friction of using LSTMs.

In late 2016, a series of papers by Nal Kalchbrenner and Aaron van den Oord, et al. thrust fully autoregressive models into the spotlight. Autoregressive models in this context refer to the class of neural network architectures which, at each point in a sequence, give the model direct access to all previous elements, i.e. modelling sequences by modelling the terms in the chain rule expansion of the sequence $p(X) = p(X_0)p(X_1|X_0)\cdots p(X_N|X_{N-1},\ldots,X_0)$. WaveNet (van den Oord et al., 2016a), ByteNet (Kalchbrenner et al., 2016a), and PixelCNN (van den Oord et al., 2016b) introduced methods for efficiently training autoregressive models in the domains of audio, language, and vision respectively.

A key innovation was the efficient implementation of a fully parallel decoding mechanism that combined teacher-forced decoding with masking to ensure full parallelisability without risking the model learning to cheat at the task by referencing the teacher-forcing targets. Teacher-forced autoregressive decoding is when the model is given access to the *ground-truth target sequence* Y and instead of conditioning the generation of each element on the previously predicted elements, the model conditions on the ground-truth sequence itself. That is, instead of modelling conditioned on the model's own predictions

$$\begin{aligned}x_0 &\sim p(X_0)\\ x_1 &\sim p(X_1|x_0)\\ \cdots\\ x_N &\sim p(X_N|x_0,\ldots,x_{N-1})\end{aligned}$$

we model conditioned on the target sequence $y_0,\ldots,y_N$

$$\begin{aligned}x_0 &\sim p(X_0)\\ x_1 &\sim p(X_1|y_0)\\ \cdots\\ x_N &\sim p(X_N|y_0,\ldots,y_{N-1}).\end{aligned}$$

This removes any dependencies between the samples and allows all to be executed in parallel.

In January of 2017, I joined Google Brain as an intern to work with Lukasz Kaiser. Lukasz had been closely following the progress of Nal and Aaron's projects and was eager to build better software frameworks to support the training of autoregressive models. In collaboration with Lukasz and many others, Tensor2Tensor (Vaswani et al., 2018) was written to be a domain-agnostic distributed neural network training framework incorporating the latest methods for improving model performance and optimisation stability.

Early on in the development of Tensor2Tensor, Lukasz and I were joined by colleagues Jakob Uszkoreit, Noam Shazeer, Ashish Vaswani, Niki Parmar, and Llion Jones who had begun exploring autoregressive attention models for text. Progress over the next three months resulted in the development of the Transformer model architecture and the publication of Attention Is All You Need (Vaswani et al., 2017b).

The Transformer architecture features a simple stack of identically structured layers; each layer is composed of an attention block followed by a depth-1 MLP, both wrapped in residual connections. The model is trained using teacher-forcing with autoregressive masking on the attention matrix, allowing the model to perform decoding fully-parallel in the sequence dimension during training.

Soon after the Transformer architecture had been released, its results in translation drew enough interest from the community to spur experimentation applying the architecture across a range of NLP problems including language modelling and representation learning. The two most well-known Transformer training methods are GPT (Radford et al., 2019a) and BERT (Devlin et al., 2018). GPT models are trained using a left-to-right autoregressive language modelling objective and are typically used to generate compelling language in production applications like assisted copywriting, summarisation, and code generation. BERT models are trained to predict masked inputs with no autoregressive constraints and are typically used to learn representations of language for use in downstream ML systems such as classifiers or search applications.

2.1 The scaling project

The original "big" Transformer models (intended for natural language translation) had approximately two-hundred million parameters and were trained on a single machine with eight Nvidia P100 GPUs for a few days. With the benefit of hindsight, this paltry deployment of compute highlights the limited support for model-based parallelism that existed at the time of the Transformer's development.

Not long after the model's release, Lukasz and Noam set about applying the Transformer to language modelling (Liu et al., 2018a) and proposed the decoder-only Transformer variant that would become the backbone of the GPT class of models. Around the same time, engineers within Google began collaboration with a team from

OpenAI to support their development of the first GPT. Noam assisted the OpenAI team in developing a training framework for large-scale models – back-boned by Mesh Tensorflow (Shazeer et al., 2018), a library for width-wise model parallelism developed by Noam for distributing a neural network across a large number of accelerators.

The result of this scaling, GPT (Radford et al., 2018), posited that unsupervised generative pretraining may allow the model to learn desirable "skills" from the data that enable the model to better solve downstream tasks. A hypothesis partially confirmed in the paper by exploring zero-shot performance of the pretrained model; and overwhelmingly confirmed by the follow-ups GPT-2 (Radford et al., 2019a) and GPT-3 (Brown et al., 2020a) which scaled to a maximum of 175B parameters – one thousand times larger than the original "big" Transformer model. This model scaling effort is the focus of Chapters 3-7.

In the years since GPT-3, the generality and power of these models has led to a wealth of applications and a surge of interest from industry and academia in continuing the scaling project and realising the full potential of large scale models trained on massive corpuses of web-scraped unsupervised data. The next section discusses the importance of this data.

2.2 Data's neglect

While model scaling has been heavily pursued and communicated by large industrial players, the web-scraped data these models are trained on have been scarcely discussed publicly. Despite the lack of attention, the quality and scale of data can have a dramatic impact on the quality of the trained model.

Web-scraped data is riddled with noise, redundancy, and toxicity. These pathologies present a challenge for models as they are compromised by noise that corrupts training gradients, redundancy that biases the model towards phrases that are rarely uttered outside of the web context, and toxicity that makes models unsafe to deploy in many scenarios.

Recent work (Hoffmann et al., 2022) has shown that most large models trained to date have been severely under-trained in terms of observed data. While most organisations were focussed on scaling models in terms of parameters to maximise performance, they neglected the impact of scaling data alongside scaling the model. These new observations spur a focus on data and the importance of its cleanliness, abundance, and diversity. The importance of data is the focus and motivation for Chapters 8-10.

Chapter 3

Learning Sparse Networks Using Targeted Dropout

Serving models with billions of parameters can incur significant cost due to the amount of accelerator memory necessary to hold the models. Sparse modelling techniques can substantially alleviate these memory burdens, however, they're often extremely complicated techniques that require significant modifications to the model and it's training. In this work, we present a regularisation scheme that is trivial to implement and is competitive with state of the art sparsity inducing techniques.

The following work is an excerpt from Gomez et al. (2019) and has received numerous citations since release. This is work done in collaboration with Ivan Zhang, Siddhartha Rao Kamalakara, Divyam Madaan, Kevin Swersky, Yarin Gal, and Geoffrey Hinton. In this project I was the source of the idea for a simpler pruning technique, led experiment design and execution, and supervised the experimental contribution of the rest of the team.

Neural networks are a powerful class of models that achieve the state-of-the-art on a wide range of tasks such as object recognition, speech recognition, and machine translation. One reason for their success is that they are extremely flexible models because they have a large number of learnable parameters. However, this flexibility can lead to overfitting, and can unnecessarily increase the computational and storage requirements of the network.

There has been a large amount of work on developing strategies to compress neural networks. One intuitive strategy is *sparsification*: removing weights or entire units from the network. Sparsity can be encouraged during learning by the use of sparsity-inducing regularisers, like L^1 or L^0 penalties. It can also be imposed by *post hoc* pruning, where a full-sized network is trained, and then sparsified according to some pruning strategy. Ideally, given some measurement of task performance, we would prune the weights or units that provide the least amount of benefit to the task. Finding the optimal set is, in general, a difficult combinatorial problem, and even a greedy strategy would require an unrealistic number of task evaluations, as there are often millions of parameters. Common pruning strategies therefore focus on fast approximations, such as removing weights with the smallest magnitude Han et al. (2015b), or ranking the weights by the sensitivity of the task performance with respect to the weights, and then removing the least-sensitive ones LeCun et al. (1990). The hope is that these approximations correlate well with task performance, so that pruning results in a highly compressed network while causing little negative impact to task performance, however this may not always be the case.

Our approach is based on the observation that dropout regularisation (Hinton et al., 2012; Srivastava et al., 2014b) itself enforces sparsity tolerance during training, by sparsifying the network with each forward pass. This encourages the network to learn a representation that is robust to a particular form of post hoc sparsification – in this case, where a random set of units is removed. Our hypothesis is that if we plan to do explicit post hoc sparsification, then we can do better by specifically applying dropout to the set of units that we a priori believe are the least useful. We call this approach *targeted dropout.* The idea is to rank weights or units according to some fast, approximate measure of importance (like magnitude), and then apply dropout primarily to those elements deemed unimportant. Similar to the observation with regular dropout, we show that this encourages the network to learn a representation where the importance of weights or units more closely aligns with our approximation. In other words, the network learns to be robust to *our choice* of post hoc pruning strategy.

The advantage of targeted dropout as compared to other approaches is that it makes networks extremely robust to the *post hoc* pruning strategy of choice, gives intimate control over the desired sparsity patterns, and is easy to implement, consisting of a two-line change for neural network frameworks such as Tensorflow (Abadi et al., 2015) or PyTorch (Paszke et al., 2017). The method achieves impressive sparsity rates on a wide range of architectures and datasets; notably 99% sparsity on the ResNet-32 architecture for a less than 4% drop in test set accuracy on CIFAR-10.

3.1 Background

In order to present targeted dropout, we first briefly introduce some notation, and review the concepts of dropout and magnitude-based pruning.

3.1.1 Notation

Assume we are dealing with a particular network architecture. We will use $\theta \in \Theta$ to denote the vector of parameters of a neural network drawn from candidate set Θ, with $|\boldsymbol{\theta}|$ giving the number of parameters. $\Omega_{\boldsymbol{\theta}}$ denotes the list of weight matrices in a neural network parameterised by $\boldsymbol{\theta}$, accordingly, we will denote $\mathrm{W} \in \Omega_{\boldsymbol{\theta}}$ as a weight matrix that connects one layer to another in the network. We will only consider weights, ignoring biases for convenience, and note that biases are not removed during pruning. For brevity, we will use the notation $w_o \equiv \mathrm{W}_{\cdot,o}$ to denote the weights connecting the layer below to the o^{th} output unit (i.e. the o^{th} column of the weight matrix), $N_{\text{col}}(\mathrm{W})$ to denote the number of columns in W, and $N_{\text{row}}(\mathrm{W})$ to denote the number of rows. Each column corresponds to a hidden unit, or feature map in the case of convolutional layers. Note that flattening and concatenating all of the weight matrices in $\Omega_{\boldsymbol{\theta}}$ would recover $\boldsymbol{\theta}$.

3.1.2 Dropout

Our work uses the two most popular Bernoulli dropout techniques, Hinton et al.'s unit dropout (Hinton et al., 2012; Srivastava et al., 2014b) and Wan et al.'s weight dropout (dropconnect) (Wan et al., 2013). For a fully-connected layer with input tensor $\boldsymbol{X}$, weight matrix W, output tensor $\boldsymbol{Y}$, and mask $\mathrm{M} \sim \mathrm{Bernoulli}(1-\alpha)$ we define both techniques below:

Unit dropout (Hinton et al., 2012; Srivastava et al., 2014b):

$$\boldsymbol{Y} = (\boldsymbol{X} \odot \mathrm{M})\mathrm{W}$$

Unit dropout randomly drops *units* (often referred to as neurons) at each training step to reduce dependence between units and prevent overfitting.

Weight dropout (Wan et al., 2013):

$$\boldsymbol{Y} = \boldsymbol{X}(\mathrm{W} \odot \mathrm{M})$$

Weight dropout randomly drops individual *weights* in the weight matrices at each training step. Intuitively, this is dropping connections between layers, forcing the network to adapt to a different connectivity at each training step.

3.1.3 Magnitude-based pruning

A popular class of pruning strategies are those characterised as *magnitude-based* pruning strategies. These strategies treat the top-k largest magnitude weights as important. We use argmax-k to return the top-k elements (units or weights) out of all elements being considered.

Unit pruning (Molchanov et al., 2016; Frankle & Carbin, 2018): considers the units (column-vectors) of weight matrices under the L^2-norm.

$$\mathcal{W}(\theta) = \left(\begin{cases} w_o, & \text{if } w_o \in \underset{1 \le j \le N_{\text{col}}(\mathrm{W})}{\underset{w_j}{\text{argmax-k}}} \|w_j\|_2 \\ 0, & \text{otherwise} \end{cases} \middle| \, 1 \le o \le N_{\text{col}}(\mathrm{W}), \mathrm{W} \in \Omega_\theta \right) \tag{3.1}$$

Weight pruning (Han et al., 2015b; Molchanov et al., 2016): considers the entries of each feature vector under the L^1-norm. Note that the top-k is with respect to the other weights within the same feature vector.

$$\mathcal{W}(\theta) = \left(\begin{cases} \mathrm{W}_{io}, & \text{if } \mathrm{W}_{io} \in \underset{1 \le j \le N_{\text{row}}(\mathrm{W})}{\underset{\mathrm{W}_{jo}}{\text{argmax-k}}} |\mathrm{W}_{jo}| \\ 0, & \text{otherwise} \end{cases} \middle| \, 1 \le i \le N_{\text{row}}(\mathrm{W}), 1 \le o \le N_{\text{col}}(\mathrm{W}), \mathrm{W} \in \Omega_\theta \right) \tag{3.2}$$

While weight pruning tends to preserve more of the task performance under coarser prunings (Han et al., 2015a; Ullrich et al., 2017; Frankle & Carbin, 2018), unit pruning allows for considerably greater computational savings (Wen et al., 2016; Louizos et al., 2017). In particular, weight pruned networks can be implemented using sparse linear algebra operations, which offer speedups only under sufficiently sparse conditions; while unit pruned networks execute standard linear algebra ops on lower dimensional tensors, which tends to be a much faster option for given a fixed sparsity rate.

3.2 Targeted Dropout

Consider a neural network parameterized by θ, and our importance criterion (defined above in Equations (3.1) and (3.2)) $\mathcal{W}(\theta)$. We hope to find optimal parameters θ^* such that our loss $\mathcal{E}(\mathcal{W}(\theta^*))$ is low, and at the same time $\|\mathcal{W}(\theta^*)\|_0 \le k$, i.e. we wish to keep only the k weights of highest magnitude in the network. A deterministic pruning implementation would select the bottom $|\theta| - k$ elements and drop them out. However, we would like for low-valued elements to be able to increase their value if they become important during training. Therefore, we introduce stochasticity into the process using a targeting proportion γ and a drop probability α. The targeting proportion means that we select the bottom $\gamma|\theta|$ weights as candidates for dropout, and of those we drop the elements independently with drop rate α. This implies that the expected number of units to keep during each round of targeted dropout is $(1 - \gamma \cdot \alpha)|\theta|$. As we will see below, the result is a reduction in the important subnetwork's dependency on the unimportant subnetwork, thereby reducing the performance degradation as a result of pruning at the conclusion of training.

3.2.1 Dependence Between the Important and Unimportant Subnetworks

The goal of targeted dropout is to reduce the dependence of the important subnetwork on its complement. A commonly used intuition behind dropout is the prevention of coadaptation between units; that is, when dropout is applied to a unit, the remaining network can no longer depend on that unit's contribution to the function and must learn to propagate that unit's information through a more reliable channel. An alternative description asserts that dropout maximizes the mutual information between units in the same layer, thereby decreasing the impact of losing a unit Srivastava et al. (2014b). Similar to our approach, dropout can be used to guide properties of the representation. For example, nested dropout (Rippel et al., 2014) has been shown to impose 'hierarchy' among units depending on the particular drop rate associated with each unit. Dropout itself can also be interpreted as a Bayesian approximation (Gal, 2016).

A more relevant intuition into the effect of targeted dropout in our specific pruning scenario can be obtained from an illustrative case where the important subnetwork is completely separated from the unimportant one. Suppose a network was composed of two non-overlapping subnetworks, each able to produce the correct output by itself, with the network output given as the average of both subnetwork outputs. If our importance criterion designated the first subnetwork as important, and the second subnetwork as unimportant (more specifically, it has lower weight magnitude), then adding noise to the weights of the unimportant subnetwork (i.e. applying dropout) means that with non-zero probability we will corrupt the network output. Since the important subnetwork is already able to predict the output correctly, to reduce the loss we must therefore reduce the weight magnitude of the unimportant subnetwork output layer towards zero, in effect "killing" that subnetwork, and reinforcing the separation between the important subnetwork and the unimportant one.

These interpretations make clear why dropout should be considered a natural tool for application in pruning. We can empirically confirm targeted dropout's effect on weight dependence by comparing a network trained with and without targeted dropout and inspecting the Hessian and gradient to determine the dependence of the network on the weights/units to be pruned. As in LeCun et al. (1990), we can estimate the effect of pruning weights by considering the second degree Taylor expansion of change in loss, $\Delta\mathcal{E} = |\mathcal{E}(\boldsymbol{\theta} - \boldsymbol{d}) - \mathcal{E}(\boldsymbol{\theta})|$:

$$\Delta\mathcal{E} = | - \nabla_{\boldsymbol{\theta}}\mathcal{E}^\top \boldsymbol{d} + {}^1\!/_2\, \boldsymbol{d}^\top \boldsymbol{H}\boldsymbol{d} + \mathcal{O}(||\boldsymbol{d}||^3)| \tag{3.3}$$

Where $d_i = \theta_i$ if $\theta_i \in \overline{\mathcal{W}(\boldsymbol{\theta})}$ (the weights to be removed) and 0 otherwise. $\nabla_{\boldsymbol{\theta}}\mathcal{E}$ are the gradients of the loss, and $\boldsymbol{H}$ is the Hessian. Note that at the end of training, if we have found a critical point $\boldsymbol{\theta}^*$, then $\nabla_{\boldsymbol{\theta}}\mathcal{E}(\boldsymbol{\theta}^*) = 0$, leaving only the Hessian term. In our experiments we empirically confirm that targeted dropout reduces the dependence between the important and unimportant subnetworks by an order of magnitude (See Fig. 3.1, and Section 3.4.1 for more details).

3.3 Related Work

The pruning and sparsification of neural networks has been studied for nearly three decades and has seen a substantial increase in interest due to their implementation on resource limited devices such as mobile phones and ASICs. Early work such as optimal brain damage (LeCun et al., 1990) and optimal brain surgeon (Hassibi & Stork, 1993), as well as more recent efforts (Molchanov et al., 2016; Theis et al., 2018), use a second order Taylor expansion of the loss around the weights trained to a local minimum to glean strategies for selecting the order in which to prune parameters. Han et al. (2015a) combine weight quantisation with pruning and achieve impressive network compression results, reducing the spatial cost of networks drastically. Dong et al. (2017) improve the efficiency of the optimal brain surgeon procedure by making an independence assumption between layers. Wen et al. (2016) propose using Group Lasso (Yuan & Lin, 2006) on convolutional filters and are able to remove up to 6 layers from a ResNet-20 network for a 1% increase in error.

A great deal of effort has been put towards developing improved pruning heuristics and sparsifying regularizers (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015a; Babaeizadeh et al., 2016; Molchanov et al., 2016; Dong et al., 2017; Louizos et al., 2017; Huang et al., 2018a; Theis et al., 2018). These are generally comprised of two components: the first is a regularisation scheme incorporated into training to make the important subnetworks easily identifiable to a post hoc pruning strategy; the second is a particular post hoc pruning strategy which operates on a pre-trained network and strips away the unimportant subnetwork.

The two works most relevant to our own are L^0 regularisation (Louizos et al., 2017) and variational dropout (Molchanov et al., 2017). Louizos et al. (2017) use an adaptation of concrete dropout (Gal et al., 2017) on the weights of a network and regularise the drop rates in order to sparsify the network. Similarly, Molchanov et al. (2017) apply variational dropout (Kingma et al., 2015) to the weights of a network and note that the prior implicitly sparsifies the parameters by preferring large drop rates. In addition to our methods being more effective at shrinking the size of the important subnetwork, targeted dropout uses two intuitive hyperparameters, the targeting proportion γ and the drop rate α, and directly controls sparsity throughout training (i.e., attains a predetermined sparsity threshold). In comparison, Louizos et al. (2017) uses the Hard-Concrete distribution which adds three hyperparameters and doubles the number of trainable parameters by introducing a unique gating parameter for each model parameter, which determines the Concrete dropout rate; while Molchanov et al. (2016) adds two hyperparameters and doubles the number of trainable parameters. In our experiments we also compare against L^1 regularization (Han et al., 2015b) which is intended to drive unimportant weights towards zero.

Another dropout-based pruning mechanism is that of Wang et al. (2017), where a procedure is used to adapt dropout rates towards zero and one (similar to Louizos et al. (2017) and (Molchanov et al., 2017)). We recommend Gale et al. (2019)'s rigorous analysis of recently proposed pruning procedures for a complete picture of the efficacy of recent neural network pruning algorithms; in particular, it challenges some of the recent claims suggesting pruning algorithms perform about as well as random pruning procedures (Crowley et al., 2018; Liu et al., 2018b).

Targeted dropout itself is reminiscent of nested dropout (Rippel et al., 2014) which applies a structured form of dropout: a chain structure is imposed on units, and children are deterministically dropped whenever their parent is dropped. In effect, each child unit gets a progressively higher marginal drop rate, imposing a hierarchy across the units; similar to both meProp (Sun et al., 2017) and excitation dropout (Zunino et al., 2018). Rippel et al. (2014) demonstrate the effect using an autoencoder where nested dropout is applied to the code; the result is a model where one can trade off reconstruction accuracy with compute by dropping lower priority elements of the code. Standout (Ba & Frey, 2013) is another similar variant of dropout; in standout, the activation value of a unit determines the drop rate, where high activations values lead to a higher keep probability and vice versa.

The Lottery Ticket Hypothesis of Frankle & Carbin (2018) demonstrates the existence of a subnetwork that – in isolation, with the rest of the network pruned away – both dictates the function found by gradient descent, and can be trained to the same level of task performance with, or without, the remaining network. In our notation, a prediction of this "winning lottery ticket" is $\mathcal{W}(\theta)$; and the effectiveness of our method suggests that one can reduce the size of the winning lottery ticket by regularising the network.

3.4 Experiments

Our experiments were performed using the original ResNet (He et al., 2016b), Wide ResNet (Zagoruyko & Komodakis, 2016), and Transformer (Vaswani et al., 2017b) architectures; applied to the CIAFR-10 (Krizhevsky & Hinton, 2009), ImageNet (Russakovsky et al., 2015), and WMT English-German Translation datasets. For each baseline experiment we verify our networks reach the reported accuracy on the appropriate test set; we report the test accuracy at differing prune percentages and compare different regularisation strategies. In addition, we compare our targeted dropout to standard dropout where the expected number of dropped weights is matched between the two techniques (i.e. the drop rate of standard dropout runs is set to $\gamma \cdot \alpha$, the proportion of weights to target times the dropout rate). We focus on pruning baselines and do not compare against the baseline of training a smaller model as this has already been shown to dramatically under-perform pruned networks (Molchanov et al., 2016).

For our pruning procedure, we perform the greedy layer-wise magnitude-base pruning described in Section 3.1.3 to all weight matrices except those leading to the logits. In our experiments we compare targeted dropout against the following competitive schemes:

L^1 **Regularization** (Han et al., 2015b): Complexity cost $\theta = ||\theta||_1$ is added to the cost function. The hope being that this term would drive unimportant weights to zero. In our table we denote this loss by L^1_β where β is the cost-balancing coefficient applied to the complexity term.

L^0 **Regularization** (Louizos et al., 2017): Louizos et al. apply an augmentation of Concrete Dropout (Gal et al., 2017), called Hard-Concrete Dropout, to the parameters of a neural network. The mask applied to the weights follows a Hard-Concrete distribution where each weight is associated with a gating parameter that determines the drop rate. The use of the Concrete distribution allows for a differentiable approximation to the L^0 cost, so we may directly minimise it alongside our task objective. When sparsifying these networks to a desired sparsity rate, we prune according to the learned keep probabilities ($\sigma(\log(\alpha))$ from (Louizos et al., 2017)), dropping those weights with lowest keep probabilities first.

Variational Dropout (Kingma et al., 2015; Molchanov et al., 2017): Similar to the technique used for L^0 regularisation, Molchanov et al. (2017) apply Gaussian dropout with trainable drop rates to the weights of the network and interprets the model as a variational posterior with a particular prior. The authors note that the variational lower bound used in training favors higher drop probabilities and experimentally confirm that networks trained in this way do indeed sparsify.

Smallify (Leclerc et al., 2018): Leclerc et al. use trainable gates on weights/units and regularise gates towards zero using L^1 regularisation. Crucial to the technique is the online pruning condition: Smallify keeps a moving variance of the sign of the gates, and a weight/unit's associated gate is set to zero (effectively pruning that weight/unit) when this variance exceeds a certain threshold. This technique has been shown to be extremely effective at reaching high prune rates on VGG networks (Simonyan & Zisserman, 2014).

Specifically, we compare the following techniques:

dropout_α: Standard weight or unit dropout applied at a rate of α.

$\text{targeted}_{\alpha,\gamma}$: Targeted dropout (the weight variant in 'a)' tables, and unit variant in 'b)' tables) applied to the $\gamma \cdot 100\%$ lowest magnitude weights at a rate of α.

variational: Variational dropout (Kingma et al., 2015; Molchanov et al., 2017) applied with a cost coefficient of $0.01/50,000$.

L^0_β: L^0 regularisation (Louizos et al., 2017) applied with a cost coefficient of $\beta/50,000$.

L^1_β: L^1 regularisation (Han et al., 2015b) applied with a cost coefficient of β.

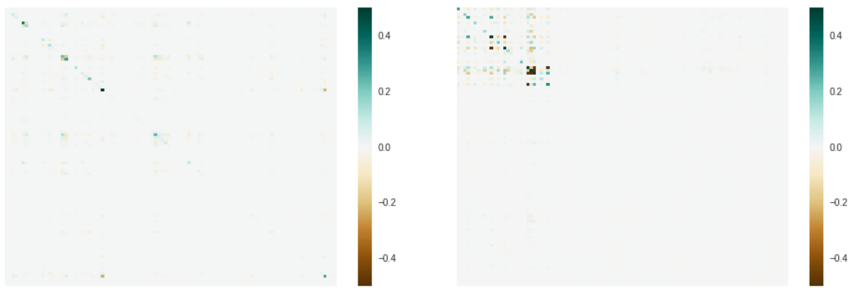


Figure 3.1: A comparison between a network without dropout (top) and with targeted dropout (bottom) of the matrix formed by $\boldsymbol{\theta}^{\top} \odot \boldsymbol{H} \odot \boldsymbol{\theta}$. The weights are ordered such that the last 75% are the weights with the lowest magnitude (those we intend to prune). The sum of the elements of the lower right hand corner approximates the change in error after pruning (Eqn. (3.3)). Note the stark difference between the two networks, with targeted dropout concentrating its dependence on the top left corner, leading to a much smaller error change after pruning (given in Table 3.1).

Table 3.1: Comparison of the change in loss ($|\Delta\mathcal{E}|$ of Equation (3.3)) for dense networks.

Regularisation	$\lvert\Delta\mathcal{E}\rvert$	Unpruned Accuracy	Pruned Accuracy
None	0.120698	38.11%	26.13%
Targeted Dropout	0.0145907	40.09%	40.14%

smallify: λ	Smallify SwitchLayers (Leclerc et al., 2018) applied with a cost coefficient of λ, exponential moving average decay of 0.9, and a variance threshold of 0.5.

3.4.1 Analysing the Important Subnetwork

In order to analyze the effects of targeted dropout we construct a toy experiment with small dense networks to analyse properties of the network's dependence on its weights. The model we consider is a single hidden layer densely connected network with ten units and ReLU activations (Nair & Hinton, 2010). We train two of these networks on CIFAR-10; the first unregularised, and the second with targeted dropout applied to the $\gamma = 75\%$ lowest-magnitude weights at a rate of $\alpha = 50\%$. The networks are both trained for 200 epochs at a learning rate of 0.001 using stochastic gradient descent without momentum.

We then compute the gradient and Hessian over the test set in order to estimate the change in error from Equation 3.3 (see Table 3.1). In addition, we compute the

Table 3.2: ResNet-32 model accuracies on CIFAR-10 at differing pruning percentages and under different regularisation schemes. The top table depicts results using the weight pruning strategy, while the bottom table depicts the results of unit pruning (see Sec. 3.1.3)

Weight Dropout/Pruning

prune percentage	none	dropout $\alpha=0.25$	targeted $\alpha=0.5,\gamma=0.5$	targeted $\alpha=0.33,\gamma=0.75$	targeted $\alpha=0.66,\gamma=0.75$	targeted $\alpha=0.75,\gamma=0.90$	variational	$L^1_{0.1}$	$L^0_{0.1}$
0 %	93.71	93.62	93.03	89.88	92.64	92.53	92.09	92.80	88.83
10%	93.72	93.63	93.04	89.80	92.62	92.55	92.00	92.72	90.66
20%	93.77	93.66	93.02	89.93	92.63	92.48	92.02	92.84	88.64
30%	93.59	93.58	92.98	89.89	92.66	92.53	92.07	92.63	87.16
40%	93.09	93.45	93.03	89.75	92.70	92.63	92.12	92.80	85.31
50%	92.20	93.07	92.99	89.72	92.65	92.54	91.84	92.29	80.94
60%	90.46	90.81	92.66	89.84	92.70	92.55	91.48	91.20	69.48
70%	81.88	72.29	92.22	89.80	92.66	92.56	90.23	86.30	46.19
80%	32.02	19.84	84.03	85.80	91.86	92.54	83.44	63.00	23.71
90%	14.63	10.05	28.27	27.04	67.58	92.48	15.16	21.08	12.55

Unit Dropout/Pruning

prune percentage	none	dropout $\alpha=0.25$	targeted $\alpha=0.5,\gamma=0.5$	targeted $\alpha=0.33,\gamma=0.75$	targeted $\alpha=0.66,\gamma=0.75$	targeted $\alpha=0.90,\gamma=0.75$	variational	$L^1_{0.01}$	$L^0_{0.01}$
0 %	93.69	92.43	92.21	90.46	89.38	89.78	93.14	93.31	93.35
10%	90.05	67.52	91.96	88.44	89.48	90.18	92.91	91.03	83.01
20%	80.34	25.05	91.63	83.55	88.89	89.79	90.38	85.63	54.59
30%	59.94	13.47	91.30	69.82	88.84	89.88	86.38	72.19	21.34
40%	35.40	10.02	89.89	54.42	87.54	89.98	83.59	46.41	10.82
50%	12.63	9.97	88.41	28.88	84.86	90.05	65.79	26.72	15.04
60%	10.65	9.99	26.55	18.55	81.98	90.08	41.05	12.11	9.46
70%	11.70	10.01	17.41	17.84	75.47	90.03	19.36	11.81	10.02
80%	9.99	9.95	10.63	10.87	28.99	34.18	9.56	14.73	14.88
90%	9.85	9.98	9.30	10.29	9.97	10.04	10.41	10.22	9.98

Hessian-weight product matrix formed by typical element $[\boldsymbol{\theta}^\top \odot \boldsymbol{H} \odot \boldsymbol{\theta}]_{ij} = \boldsymbol{\theta}_i \boldsymbol{H}_{ij} \boldsymbol{\theta}_j$ as an estimate of weight correlations and network dependence (see Figure 3.1). This matrix is an important visualisation tool since summing the entries associated with weights you intend to delete corresponds to computing the second term in Equation (3.1) – this becomes the dominant term towards the end of training, at which time the gradient is approximately zero.

Figure 3.1 makes clear the dramatic effects of targeted dropout regularisation on the network. In the Figure, we reorder the rows and columns of the matrices so that the first 25% of the matrix rows/columns correspond to the 25% of weights we identify as the important subnetwork (i.e. highest magnitude weights), and the latter 75% are the weights in the unimportant subnetwork (i.e. lowest magnitude weights). The network trained with targeted dropout relies nearly exclusively on the 25% of weights with the largest magnitude at the end of training. Whereas, the network trained without regularisation relies on a much larger portion of the weights and has numerous dependencies in the parameters marked for pruning.

Table 3.3: ResNet-102 model accuracies on ImageNet. Accuracies are top-1, single crop on 224 by 224 pixel images.

	Weight Dropout/Pruning			**Unit Dropout/Pruning**		
prune percentage	none	targeted $\alpha=0.5,\gamma=0.5$	$L^1_{10^{-5}}$	none	targeted $\alpha=0.5,\gamma=0.5$	$L^1_{0.001}$
0 %	75.9	75.7	70.6	75.7	74.3	75.7
10%	75.9	75.7	70.4	34.5	67.2	66.6
20%	74.9	75.3	69.8	1.8	59.4	12.6
30%	71.9	74.4	65.7	0.4	33.0	0.4
40%	64.4	73.5	62.1	0.1	6.4	0.2
50%	45.0	68.8	53.4	0.1	0.6	0.1
60%	8.6	50.5	38.3	0.1	0.2	0.1
70%	0.7	14.8	17.6	0.1	0.1	0.1
80%	0.2	0.4	1.4	0.1	0.1	0.1
90%	0.1	0.1	0.4	0.1	0.1	0.1

3.4.2 ResNet

We test the performance of targeted dropout on Residual Networks (ResNets) (He et al., 2016a) applied to the CIFAR-10 dataset, to which we apply basic input augmentation in the form of random crops, random horizontal flipping, and standardisation. This architectural structure has become ubiquitous in computer vision, and is gaining popularity in the domains of language (Kalchbrenner et al., 2016b), and audio (Van Den Oord et al., 2016). Our baseline model reaches over 93% final accuracy after 256 epochs, which matches previously reported results for ResNet-32 (He et al., 2016a).

Our weight pruning experiments demonstrate that standard dropout schemes are comparatively weak compared to their targeted counterparts; standard dropout performs worse than our no-regularisation baseline. We find that a higher targeted dropout rate applied to a larger portion of the weights results in the network matching unregularised performance with only 40% of the parameters.

Variational dropout seems to improve things marginally over the unregularised baseline in both weight and unit pruning scenarios, but was still outperformed by targeted dropout. L^0 regularisation was fairly insensitive to its complexity term coefficient; we searched over a range of $\beta \in [10^{-6}, 10^{1}]$ and found that values above 10^{-1} failed to converge, while values beneath 10^{-4} tended to show no signs of regularisation. Similarly to variational dropout, L^0 regularisation does not prescribe a method for achieving a specific prune percentage in a network, and so, an extensive hyperparameter search becomes a requirement in order to find values that result in the desired sparsity. As a compromise, we search over the range mentioned above and select the setting most competitive with targeted dropout; next, we applied magnitude-based pruning to the estimates provided in Equation 13 of Louizos et al. (2017). Unfortunately, L^0 regularisation seems to force the model away from conforming to our assumption of importance being described by parameter magnitude.

In Table 3.3 we present the results of pruning ResNet-102 trained on ImageNet. We observe similar behaviour to ResNet applied to CIFAR-10, although it's clear that the task utilises much more of the network's capacity, rendering it far more sensitive to pruning relative to CIFAR-10.

3.4.3 Wide ResNet

In order to ensure fair comparison against the L^0 regularisation baseline, we adapt the authors own codebase[1] to support targeted dropout, and compare the network's robustness to sparsification under the provided L^0 implementation and targeted dropout. In Table 3.4 we observe that L^0 regularisation fails to truly sparsify the network, but has a strong regularising effect on the accuracy of the network (confirming the claims of Louizos et al.). This further verifies the observations made above, showing that L^0 regularisation fails to sparsify the ResNet architecture.

Unit Dropout/Pruning

prune percentage	none	targeted $\alpha=0.33,\gamma=0.75$	$L^0_{10^{-6}}$
0 %	92.21	92.24	94.15
10%	89.76	92.09	88.05
20%	82.37	91.55	65.03
30%	52.20	90.09	13.34
40%	18.48	87.47	10.01
50%	10.53	82.09	10.00
60%	10.04	69.58	10.00
70%	10.00	44.05	10.00
80%	10.00	16.94	10.00
90%	10.00	10.43	10.00

Table 3.4: Wide ResNet (Zagoruyko & Komodakis, 2016) model classification accuracy on CIFAR-10 test set at differing prune percentages.

3.4.4 Transformer

The Transformer network architecture (Vaswani et al., 2017b) represents the state-of-the-art on a variety of NLP tasks. In order to evaluate the general applicability of our method we measure the Transformer's robustness to weight-level pruning without regularisation, and compare this against two settings of targeted dropout applied to the network.

The Transformer architecture consists of stacked multi-head attention layers and feed-forward (densely connected) layers, both of which we target for sparsification;

[1]the original L^0 PyTorch code can be found at: `github.com/AMLab-Amsterdam/L0_regularization`

Weight Dropout/Pruning

prune percentage	none	targeted $\alpha=\frac{2}{3},\gamma=\frac{3}{4}$	targeted $\alpha=\frac{2}{3},\gamma=\frac{9}{10}$
0 %	26.01	26.52	25.32
10%	26.05	26.44	25.32
20%	25.90	26.48	25.19
30%	25.91	26.30	25.27
40%	25.81	26.20	24.97
50%	25.08	26.03	24.93
60%	23.31	25.62	24.27
70%	8.89	24.07	22.41
80%	0.24	12.39	10.57
90%	0.01	0.07	0.64

(a) Transformer model uncased BLEU score.

Weight Dropout/Pruning

prune percentage	none	targeted $\alpha=\frac{2}{3},\gamma=\frac{3}{4}$	targeted $\alpha=\frac{2}{3},\gamma=\frac{9}{10}$
0 %	62.29	58.31	57.41
10%	62.54	59.00	58.10
20%	62.21	59.39	58.52
30%	62.33	58.66	57.86
40%	61.81	59.39	58.67
50%	60.82	57.71	57.08
60%	58.13	58.42	57.96
70%	48.40	55.39	54.85
80%	25.80	47.09	46.63
90%	6.90	21.64	27.02

(b) Transformer model per-token accuracy.

Table 3.5: Evaluation of the Transformer Network under varying sparsity rates on the WMT newstest2014 EN-DE test set.

within the multihead attention layers, each head of each input has a unique linear transformation applied to it, which are the weight matrices we target for sparsification.

Table 3.5a details the results of pruning the Transformer architecture applied to the WMT newstest2014 English-German (EN-DE). Free of any regularisation, the Transformer seems to be fairly robust to pruning, but with targeted dropout we are able to increase the BLEU score by 15 at 70% sparsity, and 12 at 80% sparsity; further confirming target dropout's applicability to a range of architectures and datasets.

3.4.5 Scheduling the Targeting Proportion

Upon evaluation of weight-level Smallify (Leclerc et al., 2018) we found that, with tuning, we were able to out-perform targeted dropout at very high pruning percentages (see Table 3.6). One might expect that a sparsification scheme like Smallify – which allows for differing prune rates between layers – would be more flexible and better suited to finding optimal pruning masks; however, we show that a variant of targeted dropout we call *ramping targeted dropout* is capable of similar high rate pruning. Moreover, ramping targeted dropout preserves the primary benefit of targeted dropout: fine control over sparsity rates.

Ramping targeted dropout simply anneals the targeting rate γ from zero, to the specified final γ throughout the course of training. For our ResNet experiments, we anneal from zero to 95% of γ over the first forty-nine epochs, and then from 95% of γ to 100% of γ over the subsequent forty-nine. In a similar fashion, we ramp α from 0% to 100% linearly over the first ninety-eight steps.

Using ramping targeted dropout we are able to achieve sparsity of 99% in a ResNet32 with accuracy 87.03% on the CIFAR-10 datatset; while the best Smallify run achieved intrinsic sparsity of 98.8% at convergence with accuracy 88.13%, when we perform pruning to enforce equal pruning rates in all weight matrices, the

Weight Dropout/Pruning

prune percentage	targeted α=0.66,γ=0.75	smallify λ=0.00001
0 %	92.64	90.16
10%	92.62	90.13
20%	92.63	90.16
30%	92.66	90.06
40%	92.70	90.17
50%	92.65	90.20
60%	92.70	90.12
70%	92.66	90.10
80%	91.86	90.15
90%	67.58	90.16

prune percentage	smallify λ=0.00001	ramp targ α=0.99,γ=0.99
90%	90.20	89.03
91%	90.33	89.16
92%	90.30	89.14
93%	90.27	89.03
94%	89.46	89.05
95%	89.41	89.05
96%	88.55	89.02
97%	86.35	89.05
98%	59.27	89.05
99%	13.83	88.97

prune percentage	ramp targ α=0.99,γ=0.99
98.5%	89.03
98.6%	89.08
98.7%	89.00
98.8%	89.05
98.9%	88.99
99.0%	89.10
99.1%	88.35
99.2%	79.88
99.3%	77.35
99.4%	16.55

Unit Dropout/Pruning

prune percentage	targeted α=0.66,γ=0.75	smallify λ=0.0001	ramp targ α=0.99,γ=0.90
0 %	90.55	90.20	85.98
10%	90.83	90.33	86.12
20%	89.88	90.30	86.01
30%	87.35	90.27	86.10
40%	85.39	89.46	85.98
50%	80.84	89.41	86.13
60%	71.97	88.55	86.02
70%	55.98	86.35	86.08
80%	10.02	59.27	85.95
90%	10.07	13.83	85.99

Table 3.6: Comparing Smallify to targeted dropout and ramping targeted dropout. Experiments on CIFAR10 using ResNet32.

network degrades rapidly (see Table 3.6).

3.4.6 Fixed Filter Sparsity

We also propose a variation of Ramping targeted dropout (Section 3.4.5), where each layer is assigned a γ such that only a fixed number of weights are non-zero by the end of training (for example, three parameter per filter). We refer to this as Xtreme dropout. ResNet32 when trained with Xtreme-3 (3 weights per filter are non-zero) was able to achieve an accuracy of 84.7% on the CIFAR-10 datatset at a sparsity level of 99.6% while Xtreme-4 was able to achieve 87.06% accuracy at a sparsity level of 99.47%. An interesting observation of Xtreme pruning is that when trained on ResNet18, it achieves 82% accuracy at a sparsity level of 99.8%. When translated to the number of parameters, it has only 29,760 non-zero parameters (includes BatchNorm) which is less than the number of parameters in a network consisting of a single dense layer with 10 output units.

3.5 Exploring Recent Discussions and Concerns

A line of work (Crowley et al., 2018; Liu et al., 2018b) has suggested that post hoc pruning with fine-tuning is not as effective as it could be. They propose using the sparsity patterns derived from a pruned model to define a smaller network (where the remaining, unpruned weights are reinitialised randomly) which is then trained from scratch, yielding better final task performance than fine-tuning the pruned model's weights.

A similar question arose in our own work as we pondered how early in the training procedure the important subnetwork could be decided. In the ideal case, the important subnetwork would be arbitrary and we could blindly select any subnetwork at the beginning of training, delete the remaining network, and recover similar accuracy to a much more complicated pruning strategy. While in the worst case, the important subnetwork would be predestined, and would remain difficult to identify until the very end of training.

While Crowley et al. (2018); Liu et al. (2018b) rely on sparsity patterns derived from pruned models, in this paper we are concerned with pruning schemes that achieve sparsity in a single execution of the training procedure; and so, in order to evaluate the more general claim that training smaller networks from scratch can match (or even out-perform) pruning, we compare the following two methods:

- **Random-pruning:** Before training, prune away a random subnetwork.
- **Targeted Dropout (Ramping TD):** Apply ramping targeted dropout throughout the course of training.

The results of our experiment are displayed in Table 3.7. It is clear that – although Crowley et al. (2018); Liu et al. (2018b)'s results show that knowing a *good* sparsity pattern in advance allows you to achieve competitive results with pruning – simply training a smaller subnetwork chosen at random does not compete with a strong regularisation scheme used over the course of training. Similar observations that contradict the conclusions of Crowley et al. (2018); Liu et al. (2018b) have been made in both Frankle et al. (2019a) and Gale et al. (2019).

3.6 Conclusion

We propose *targeted dropout* as a simple and effective regularisation tool for training neural networks that are robust to *post hoc* pruning. Among the primary benefits of targeted dropout are the simplicity of implementation, intuitive hyperparameters, and fine-grained control over sparsity - both during training and inference. Targeted

	Prune %			
Type	50%	75%	90%	99%
Random-prune	92.58	92.32	90.66	80.86
Ramping TD	**93.29**	**92.72**	**92.51**	**88.80**

(a) Comparison of weight-level pruning methods using ResNet-32 trained on CIFAR-10.

	Prune %			
Type	75%	85%	90%	95%
Random-prune	90.50	88.52	84.98	79.09
Ramping TD	**90.84**	**88.59**	**86.45**	**80.65**

(b) Comparison of unit-level pruning methods using ResNet-32 trained on CIFAR-10.

	Prune %			
Type	75%	85%	90%	95%
Random-prune	48.98 (0.62)	45.58 (1.25)	40.50 (2.03)	31.44 (1.64)
Ramping TD	**52.64** (0.61)	**49.20** (0.10)	**45.03** (0.83)	30.15 (1.72)

(c) Comparison of unit-level pruning methods using VGG-16 trained on CIFAR-100. Results are the average of five independent training runs followed by one standard deviation reported in brackets.

Table 3.7: Comparison between random pruning at the beginning of training and regularising with targeted dropout throughout the course of training, followed by post hoc pruning.

dropout performs well across a range of network architectures and tasks, demonstrating is broad applicability. Importantly, like Rippel et al. (2014), we show how dropout can be used as a tool to encode prior structural assumptions into neural networks. This perspective opens the door for many interesting applications and extensions.

Chapter 4

Optimising Neural Networks in Low-rank Subspaces

The standard building block of a neural network is a vector-matrix multiplication between an intermediate representation and a weight matrix, followed by an activation function. One straight-forward approach to reducing both the memory and the compute requirements of a neural network would be to factorise the weight matrix and truncate its spectrum to be low-rank. If such a reduction in rank does not limit the expressiveness of the neural network to the extent that it learns a worse solution, then the method could provide both memory and compute benefits. However, in practice it's generally observed that optimising low-rank factorisations of neural networks leads to inferior models that under-perform their full-rank equivalents. In the following work we explore whether this degradation in performance is due to the network's reduced complexity, or whether it is the result of poor transfer of hyperparameters from the full-rank to low-rank setting.

The following is an excerpt from Kamalakara et al. (2022). This is work done in collaboration with Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, and Yarin Gal. In this project I was the source of the idea for the method, led experiment design, and supervised the project alongside Jimmy Ba and Yarin Gal.

Recent developments in training very large vision and language models Brown et al. (2020a); Fedus et al. (2021); Dosovitskiy et al. (2020) have led to an increasing need for efficient training paradigms. Low rank matrix factorisation of layers in a deep neural network can offer significant training speedups (up to 2x) and consumes less memory when compared to its unfactorised counterpart. Matrix factorisation has been studied extensively in the context of linear networks and their applications to matrix sensing and matrix completion problems. In deep neural networks, the effects of factorised layers on optimisation are non-trivial. Hence, prior work in this space predominantly focused on low-rank training with additional training objectives, or involved computing factorised approximations *post-training*. There has been limited prior work that focused on training dynamics for low rank deep neural networks.

Our contributions: we examine the recent developments in training low rank networks and question existing beliefs about why techniques like singular value decomposition (SVD) based initialisation and modified L_2 regularisation are effective. We start with SVD based initialisation techniques which have been found to be effective in both low-rank and sparsity literature Lee et al. (2019b). We look to random matrix theory to formally define the distribution of singular values at initialisation in modern neural networks and challenge prior assumptions on their importance. We reveal novel empirical insights about the dynamics of singular values during training of an L_2 regularised network and present a hypothesis about why L_2 regularisation on the re-composed matrix works better than L_2 regularisation on its factors. We also investigate currently held beliefs about effective step size and its correlation with performance. Moreover, we analyse and present experiments with pre-training as a strategy to train better performing low-rank networks. We present a wide array of experiments to support our arguments and to demonstrate the effectiveness and practicality of training low-rank neural networks.

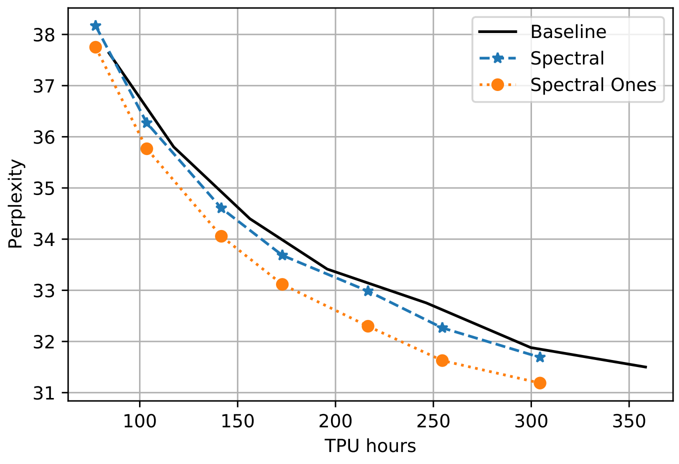


Figure 4.1: TPU Compute hours vs Performance of GPT-2 on LM1B as the model is scaled up. Each point on the line corresponds to a different model size starting from 1024 hidden dimensions (on the top left) to 2560 (in the bottom right) with increments of 256.

4.1 Related Work

Most works in the low rank space that focus on efficiency and speedups looked at post-hoc approximation of trained networks. Yu et al. (2017) took an SVD free approach to reconstruct feature maps by minimising an objective that imposes sparse low rank structure. Jaderberg et al. (2014) also considered a trained network upon which a low rank structure is imposed through filter and data reconstruction objectives. Tai et al. (2016) focused on low rank training of CNNs from scratch; they proposed a horizontal and vertical filter decomposition of a convolutional kernel and reproject into orthogonal vectors at every step. One of the reasons why many related works have focused on post-training low rank approximations is that training dynamics of neural networks are poorly understood. To resolve this to an extent, many recent works have attempted to understand the implicit bias of gradient descent (GD) in matrix factorisation in both linear and non-linear networks. Arora et al. (2019) investigated the behaviour of GD in deep linear networks and found that as the depth of factorisation increases, GD tends to find low rank solutions. They also present evidence for the hypothesis that the language of norms such as nuclear norm, Frobenius norm, etc, may not be enough to describe the behaviour of GD. Martin & Mahoney (2018) presented an empirical analysis of commonly used architectures and characterised the dynamics of GD in deep non-linear networks in terms of Empirical Spectral Distributions (ESD) and phases of training. They define a set of rank measures, which we use in our work to analyse low rank training juxtaposed with analysis on unfactored training. Wang et al. (2021) used low rank training with unfactorised pretraining in the context of efficient communication in a distributed setting. Khodak et al. (2021) proposed a low rank training procedure by investigating initialisation and regularisation in factorised layers. They analysed SVD based initialisation (Spectral Initialisation) and properties of L_2 regularisation which we study independently in our work. They conjecture that there is an interplay between normalisation and weight decay and formalise this behaviour through factorised update equations.

4.2 Low Rank Training

4.2.1 Factorisation

In all our experiments and analyses, we factorise a weight matrix $\boldsymbol{W}$ at each layer into two components $\boldsymbol{U}$ and $\boldsymbol{V}$ such that $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^\top$.

We focus on a factorisation depth of 2, taking into consideration memory-speedup tradeoffs: As the depth of factorisation at each layer increases, more activations need to be stored in-memory for backpropagation. A depth of two provides speedups across all our experiments while ensuring minimal activation memory overhead.

Consider the difference between the vanilla gradient descent update (unfactorised)

$W_{t+1} = W_t - \alpha \nabla W$ and the update performed in the factorised setting:

$$\begin{aligned} W_{t+1} &= U_{t+1} V_{t+1}^\top \\ W_{t+1} &= (U_t - \alpha \nabla U)(V_t - \alpha \nabla V)^\top \\ W_{t+1} &= W_t - \alpha \underbrace{(\nabla W_t V_t V_t^\top + U_t U_t^\top \nabla W_t)}_{\nabla_t} \\ &\quad + \alpha^2 \nabla W_t W_t \nabla W_t^\top \end{aligned} \tag{4.1}$$

Khodak et al. (2021) extend the update equation above to normalised layers. Most modern architectures rely on normalisation layers to train networks that generalise well. This includes BatchNorm in ResNets and LayerNorm in Transformers. We refer the reader to Khodak et al. (2021) for a more detailed discussion on the type and role of normalisation in factorised layers and use their formulation of the normalised update equation, which is given by

$$\begin{aligned} \hat{w}_{t+1} &= \hat{w}_t - \frac{\alpha}{\| W\|_F^2}(I_{mn} - \hat{w}_t \hat{w}_t{}^\top)\mathrm{vec}(\hat{\nabla}_t) \\ &\quad + \mathcal{O}(\alpha^2) \end{aligned} \tag{4.2}$$

where $\hat{\nabla}_t$ is ∇_t with gradients taken with respect to the normalised weight matrix $\hat{W} = \frac{W}{\| W\|_F}$ and $\hat{w} = \mathrm{vec}(\hat{W})$.

We see that gradient descent in the factorised setting does not perfectly align with the vanilla gradient descent update. In the subsequent sections, we empirically explore and work to overcome the implicit biases of this factorised update so that we can make low rank training an effective and efficient training method.

4.2.2 Spectral Initialisation

Khodak et al. (2021) investigated the usefulness of spectral initialisation in low rank formulations of deep learning architectures and proposed a few hypotheses for why it seems to improve optimisation. We use the same truncated SVD initialisation scheme, which is as follows

$$\begin{aligned} \mathrm{SVD}_r(W) &= \hat{U}_{:r} \Sigma_r \hat{V}_{:r}^\top, \\ U &= \hat{U}_{:r} \sqrt{\Sigma_r}, \\ V &= \hat{V}_{:r} \sqrt{\Sigma_r}, \end{aligned} \tag{4.3}$$

where W is a matrix of shape $N \times M$, U of shape $N \times r$, V of shape $M \times r$, Σ is the diagonal matrix of singular values and r is the rank we choose for the factorisation. We note that U and V are rectangular matrices unless specified otherwise.

Khodak et al. (2021) analysed SVD based initialisation in the context of the update equation 4.1 and make an incorrect assumption for why this initialization is effective: That $U_0 U_0^\top = V_0 V_0^\top = \Sigma_r$. In the low rank context, U and V are rectangular matrices obtained from truncated SVD which makes U and V column-wise orthogonal matrices. Therefore, we point out that $UU^\top$ and $VV^\top$ *cannot* be equal to Σ_r and $\nabla W_t V_t V_t^\top + U_t U_t^\top \nabla W_t$ terms in the equation 4.1 cannot be simplified.

We believe that spectral initialisation works for reasons other than the ones stated in prior work. In section 4.3.1, we present an ablation experiment that hints at why this initialisation scheme performs better.

4.2.3 L_2 Regularisation

Many architectures rely on L_2 regularisation for better generalisation. The straightforward approach to impose L_2 regularisation in a factorised network is to apply the Frobenius norm penalty to the factors U and V – that is, $\frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)$. Srebro & Shraibman (2005) showed that this penalty actually minimises the nuclear norm of the recomposed matrix $UV^\top$.

To address this, Khodak et al. (2021) propose penalising the Frobenius norm of the recomposed matrix $UV^\top$, which they refer to as, Frobenius decay. They argue that Frobenius decay helps in keeping the effective step size high through out training where effective step size is the term $\frac{\eta}{\|W\|_F^2}$ in equation 4.2. We show, through an ablations study, that effective step size is an inadequate argument to justify the effectiveness of Frobenius decay over L_2 regularization. We point out that the dynamics of low-rank training with L_2 regularisation cannot be understood by only considering the normalised update equation 4.2. This ignores the $\eta\lambda \approx \mathcal{O}(\eta^2)$ terms arising from Frobenius norm penalty which have a non-trivial impact on the optimisation. We find that the effectiveness of Frobenius decay over L_2 regularisation can be better explained by examining the effective rank of the network. We use the rank measure proposed in Martin & Mahoney (2018) which defines effective rank as the nuclear norm divided by the operator norm i.e $\frac{\|UV^\top\|_*}{\|UV^\top\|_{op}}$.

4.2.4 Pre-training

The initial stages of training are widely believed to be important for good performance in neural networks Achille et al. (2017) Frankle et al. (2019b). This motivates us to explore training for a fraction of the total training steps in the unfactorised space before switching to low rank substitutions of these unfactorised layers. We apply the truncated SVD scheme descibed in equation 4.3 to the partially trained weights to obtain the factors of the layer. Section 4.3.3 describes the impact of pre-training on performance across our vision and language experiments and analyses the nature of the solutions found with pre-training when compared to solutions found by low rank networks trained from scratch Evci et al. (2019) Frankle et al. (2019c).

4.3 Experiments and Results

We conduct extensive experiments on both vision and language models. For vision models, we use a Wide-ResNet-28 on CIFAR-100 and a ResNet-50 on the ImageNet dataset. For the language modelling task, we conduct experiments on one million word benchmark dataset (LM1B) Chelba et al. (2013b) and use the GPT-2 Radford et al. (2019a) architecture. Details on our complete experimental setup can be found in the supplementary material. In the following sections, we compare different initialisation schemes and study the effects of applying L_2 regularisation to the factors U, V and $UV^\top$ in normalised neural networks. Finally, we demonstrate the effectiveness of — and analyse the nature of solutions found by — pre-training.

4.3.1 Initialisation

We show that spectral initialisation offers commensurate performance when compared to traditional initialisation schemes. Then, we show empirically that the singular values do not play a major role in improving performance and that it is the direction of the singular vectors that matters. This finding is in contrast with prior beliefs Khodak et al. (2021) about the role of singular values in retaining the scale of initialisation. We establish this by setting the singular values to ones in equation 4.3. Tables 4.2, 4.3, 4.4 compare the results across initialisation schemes on CIFAR100, ImageNet and LM1B respectively. We observe that spectral ones leads to a better accuracy on CIFAR-100, lower perplexity on LM1B and a commensurate performance on ImageNet. While we offer no concrete explanation for why discarding the singular values is beneficial to optimization, one relevant note is that in regular neural networks, it is important to initialize all neurons with a similar magnitude; in spectral initialization, the columns of U and rows of V are being scaled by values that can differ dramatically, potentially hampering optimization behaviour. We leave a proper analysis of the subject to future work.

4.3.2 L_2 Regularisation

We investigate the effective step size hypothesis by training two networks, one with learning rate η and the other with $\frac{\eta}{2}$. So, the effective step size of these networks is $\frac{\eta}{||W||_F^2}$ and $\frac{\eta}{2||W||_F^2}$ respectively, based on equation 4.2. If the hypothesis that a

Model	Dataset	Frobenius decay	L_2
WRN	CIFAR-100	39.87	16.4
ResNet-50	ImageNet	68.72	58.00
Transformer	LM1B	206.93	205.70

Table 4.1: Effective rank measures for different models

higher effective step size leads to better performance were true, we should see that halving the effective step size should lead to a lower performance but we find that $\frac{\eta}{2}$ leads to models that are atleast as good as models trained with learning rate η.

Tables 4.5, 4.6 and 4.7 compare the impact of effective step size on performance across CIFAR-100, ImageNet and LM1B respectively. Analysing the evolution of singular values in networks trained with L_2 regularisation and Frobenius decay revealed that singular values are disproportionately affected in the case of L_2 regularisation. We observed a "rich get richer, poor get poorer" phenomenon in L_2 regularised networks which caused the effective rank $\frac{\|UV^\top\|_*}{\|UV^\top\|_{op}}$ of the network to drop because of the disproportionate increase in the operator norm of each layer. We report the averaged (across layers) effective rank at the end of training for our experiments in Table 4.1

4.3.3 Pre-training

We investigate pre-training networks for a fraction of the total training steps and observe that this leads to significantly improved performance in our language model experiments shown in Figure 4.1. We pre-train in the unfactorised space for 40,000 steps and continue training in the factorised space for 200,000 steps. We combine pre-training with the techniques aforementioned *viz* Frobenius decay and resuming with decompositions obtained from Spectral and Spectral ones as described in 4.2.4. We find that pre-training does not offer improved performance compared to low-rank network trained from scratch in our vision experiments as shown in Tables 4.8 and 4.9. Furthermore, we notice that the solutions found with pre-training are closer in the parameter space to their corresponding baseline (unfactorised) models. We demonstrate this by performing linear interpolation between pre-training and baseline weights by using the following equation: $\theta = (1-t)\theta_b + (t)\theta_l$ for $t \in [0.0, 1.0]$ with increments of 0.1 where t is the interpolation coefficient, θ_b is the parameter from the baseline model and θ_l is the parameter from the low rank model with pre-training.

4.3.4 Experiment Details

For the language modelling task, we conduct our experiments on one million word benchmark dataset (LM1B) (Chelba et al., 2013b) and use the following set up: input

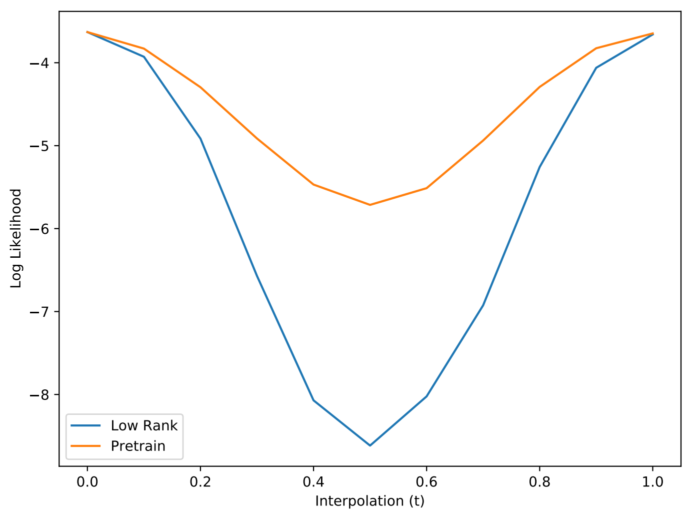


Figure 4.2: Comparison of interpolation of low rank and pretrained networks for LM

sequence length is fixed at 256 and 1152 tokens for training and evaluation respectively and the vocab size is limited to 32K subwords and train all the models to 240K steps. We implemented transformer language model on Tensorflow and run all our experiments on cloud TPUs. To have better savings on compute and memory we combine the query, key value generation into one weight matrix. For each transformer layer, we decompose three matrix operations; Q,K,V generation and the two fully connected layers. We skip factorising the output projection layer and the combiner layer that combines the outputs of attention (this is a square matrix and we see memory and compute benefit only for very small ranks). For all transformer runs, we choose a rank of 62.5% and half its baseline learning rate. For pre-training, we train unfactored for 40K steps then switch to low rank factorised training for the remaining 200K steps and halving the learning rate.

For the image classification task, we conduct experiments with CIFAR-100 and ImageNet. For CIFAR-100 we use the standard training/test split with a simple augmentation scheme – Random Crop and Horizontal Flips. We train a WideResNet-28 (Zagoruyko & Komodakis, 2016) for 200 epochs with SGD with momentum (0.9) and a batch size of 128. For regularisation, we a weight decay coefficient of 5e-4 and no dropout. For the low rank training runs, we factorised every convolutional layer other than the first according to our factorisation scheme describe above and the chosen rank. For ImageNet experiments, we use a standard ResNet-50 architecture and train on a TPU v2-8 with a per-core batch size of 128 and follow the same hyperparameters and learning rate schedule described in He et al. (2016b).

Rank	**Initialisation**	**Accuracy**
Baseline (N/A)	He	81.08
0.1	He	77.94
	spectral	79.84
	spectral ones	79.07
0.2	He	80.37
	spectral	81.35
	spectral ones	81.27
0.3	He	80.87
	spectral	81.53
	spectral ones	81.61

Table 4.2: Initialization results of Wide Resnets on Cifar-100

Rank	**Initialisation**	**Top-1**	**Top-5**
Baseline (N/A)	He	76.39	93.21
0.3	He	75.26	92.56
	spectral	75.77	92.87
	spectral ones	75.71	92.82
0.5	He	75.97	92.84
	spectral	76.13	93.09
	spectral ones	75.98	92.97

Table 4.3: Initialization results of ResNet on Image Net

Rank	**Initialisation**	**Perplexity**
Baseline (N/A)	He	37.67
0.62	He	39.6
	spectral	38.78
	spectral ones	38.47

Table 4.4: Initialization results of Transformers on LM1B

Rank	**Regularisation**	**lr scaling**	**Accuracy**
0.1	L2	0.5	73.12
		1.0	72.59
	Frobenius Decay	0.5	79.84
		1.0	79.79
0.2	L2	0.5	78.22
		1.0	77.56
	Frobenius Decay	0.5	81.35
		1.0	81.61

Table 4.5: Comparison between Frobenius Decay and L2 regularisation on Cifar-100

Rank	**Regularization**	**lr scaling**	**Top-1**	**Top-5**
0.3	L2	0.5	75.11	92.42
		1.0	74.9	92.24
	Frobenius Decay	0.5	75.22	92.49
		1.0	75.77	92.87
0.5	L2	0.5	75.04	92.36
		1.0	74.83	92.25
	Frobenius Decay	0.5	75.97	92.85
		1.0	76.13	93.09

Table 4.6: Comparison between Frobenius Decay and L2 regularisation on Imagenet

Rank	**Regularisation**	**lr scaling**	**Perplexity**
0.62	L2	0.5	38.87
		1.0	39.01
	Frobenius Decay	0.5	38.78
		1.0	39.2

Table 4.7: Comparison between Frobenius Decay and L2 regularisation on LM1B

Rank	**Pre-training Epochs**	**Accuracy**
0.2	0	81.35
	15	81.33
	30	81.56
	40	81.53
	50	81.39
	75	81.53
0.3	0	81.53
	15	81.73
	30	81.51
	40	81.67
	50	82.0
	75	81.44

Table 4.8: Pre-training results for Wide ResNets on CIFAR-100

Rank	# Pretrain epochs	Top-1	Top-5
0.5	5	76.07	92.88
	10	75.96	93.04
	15	76.12	92.96
	20	76.08	92.94
	25	76.15	93.00
	30	76.05	92.9
	35	76.24	93.06
	40	76.21	93.09
	45	76.29	93.12

Table 4.9: Pre-training results for ResNet50 on ImageNet

Chapter 5

Improving Training Efficiency Using SliceOut

Dropout is a widely utilised regularisation strategy that stochastically zeros (drops) dimensions of intermediary activations. By zero-ing dimensions, the subsequent vector-matrix multiplication between the activation and the subsequent layer's weight will result in many redundant FLOPs where the activation's dimension is zero. SliceOut seeks to save these wasted FLOPs by changing the distribution of dropped activations so that they are always contiguously arranged. This way, a simple slice operation can be used to eliminate the dropped dimension; saving both compute and memory during training.

The following is an excerpt from Notin et al. (2020) and is under review with the Journal of Machine Learning Research. This is work done in collaboration with Pascal Notin, Joanna Yoo, and Yarin Gal. In this project I was the source of the idea for a fast dropout variant using slice operations, led experiment design, and supervised the experimentation.

The success of deep learning over the past two decades has relied heavily on algorithmic and hardware innovations to support ever increasing computational workloads. While several methods have been recently introduced to achieve step-improvements in efficacy at inference time (e.g., quantisation, pruning), translating these benefits to training as been a more challenging endeavour given the impact they may have on the training dynamics. When dealing with a fixed compute budget, the ability to train the same models more rapidly supports shorter research iteration cycles, more extensive hyperparameter or architecture searches, or a reduction in the required energy consumption and the corresponding carbon footprint. In applications that require regular model re-training (e.g., active learning, continual learning), faster training translates into more regular updates and subsequently stronger task performance with the same resources.

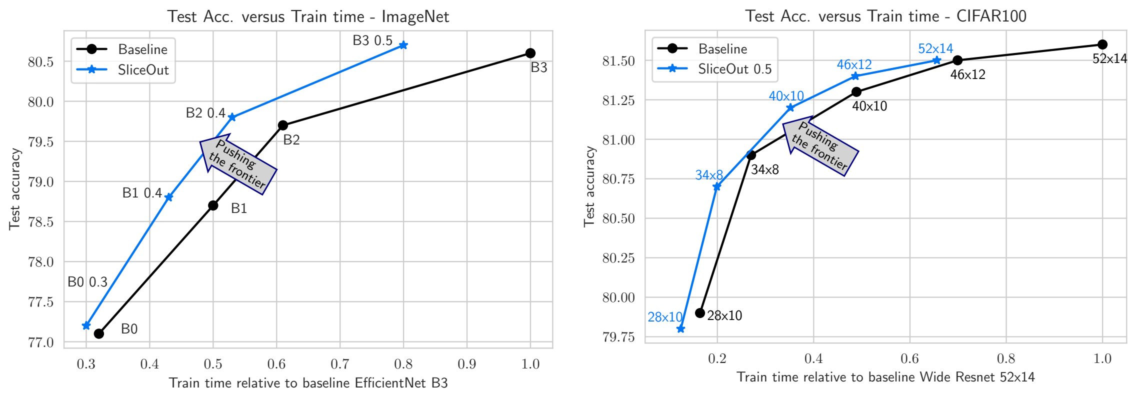


Figure 5.1: Compute efficacy frontiers: Leveraging SliceOut allows to achieve a more desirable compute efficacy frontier for EfficientNets on Imagenet (left) and Wide ResNets on CIFAR100 (right). Each curve represents the test accuracy Vs required training time (relative to train time of the larger net) for networks trained with SliceOut (blue curves) and without (black curves). See detailed results in tables 5.2 and 5.3.

In this work, we introduce an architecture-agnostic method to train neural networks faster without compromising on final test accuracy, thereby achieving a more desirable compute efficacy frontier (see Fig. 5.1).

Our proposed method, SliceOut (§5.2), draws inspiration from dropout (Hinton et al., 2012; Srivastava et al., 2014a), a regularisation technique widely used in large neural networks. We show that the scheme can be used as an alternative to standard dropout that simultaneously preserves its regularisation benefits while achieving speedups and memory gains at train time. More generally, we demonstrate it can be also leveraged to achieve training speedups in architectures where no dropout was used in the first place.

SliceOut introduces structure to dropout by slicing contiguous memory segments, i.e., selecting a contiguous range of neighboring neurons and slicing feature tensors or weight matrices row/column-wise (Fig. 5.3c), as opposed to selecting neurons uniformly at random. From the computational perspective, this strategy takes advantage of GPU memory layout as the operation requires a single access to contiguous memory. From the memory perspective, the zero units, that would physically remain in memory with standard dropout, are removed from memory overhead by the slicing operation. This implies a smaller memory footprint for weight gradients and activations throughout the network, and also results in matrix multiplications with smaller tensors compared to standard dropout. This in turn allows us to fit larger models in memory than would otherwise be possible, or conversely, to train a model of similar size with fewer computing resources. The relative simplicity of the approach as a constrained-form of dropout facilitates its implementation across architectures and deep-learning frameworks. Lastly, SliceOut helps prevent some of the issues that standard dropout has when applied to CNNs (§5.2.4 and Fig. 5.4).

Our experiments are carried in three settings (§5.3): the first consists of relatively small neural networks applied to MNIST and FashionMNIST, to illustrate the benefits of the approach in a simple setting; the second is Wide ResNets applied to CIFAR-10/100 and EfficientNets applied to CIFAR-10/100 and ImageNet, demonstrating significant memory and speedup gains due to the large reduction in ops on the high dimensional feature vectors of CNNs; and the final setting is language modelling with Transformers applied to LM1B, demonstrating the applicability of our method beyond vision tasks. In all our settings we find that SliceOut performs comparatively (or out-performs) standard dropout in terms of test accuracy, while achieving memory and compute savings of 10-40%, depending on the model architecture and dropout rate considered.

Our contributions are as follows:

- We introduce SliceOut, a general-purpose scheme to train neural networks faster without impacting final test accuracy
- We derive various sampling and normalisation schemes for the method which preserve (exactly or approximately) the first and second moments of the layers' output, allowing for efficient deterministic approximations at inference time
- We implement this new scheme across a diverse set of network architectures - from regular MLPs, to Wide ResNets, EfficientNets and Transformers
- We quantify the relative speedups and memory gains between the different dropout schemes across experimental setups, demonstrating practical gains with SOTA models with minimal to no impact on test accuracy

5.1 Background

5.1.1 Compute efficacy frontiers

In the past few years we have observed an unprecedented race to training ever larger neural networks via massive compute resources with the ultimate objective to squeeze in the most parameters possible for a fixed amount of compute – the latest example being the GPT-3 model with a total of 175 billion parameters (Brown et al., 2020b). Significant progress has also been made towards the ability to train large deep networks very rapidly – with several teams competing to train high accuracy models on ImageNet in a few minutes (Jia et al., 2018; Goyal et al., 2017). McCandlish et al. (2018) investigate the relationship between compute resources and total training time to achieve a fixed test accuracy, and observe Pareto frontiers connecting the two, for example by training a model to solve the Atari Breakout game.

The aforementioned examples demonstrate the intricate relationships between amount of compute available, overall training duration, and final test accuracy. A

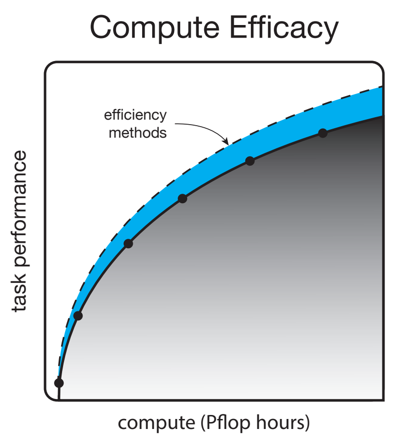


Figure 5.2: The purpose of developing efficiency in ML is pushing forward the *compute efficacy frontier*: the frontier describing the best model one can obtain using a given amount of compute.

convenient way to conceptualise these relationships and describe the objective of progress in ML efficiency is to consider the *compute efficacy frontier*. The compute efficacy frontier defines the utility of a given amount of compute; that is, given X accelerators operating for Y hours, the frontier describes the best performing model one can obtain (See Figure 5.2). In our work, we introduce a method that pushes the frontier forward by reducing the compute while preserving the task performance. We show that this method is effective across model architectures and across task domains. Importantly, we show that even in highly efficient and optimised settings – like EfficientNet models for ImageNet (Figure 5.1) – our method has a dramatic impact on compute efficacy.

5.1.2 Related dropout variants

Standard dropout randomly "turns off" at train time the neurons of a given layer and, implicitly, the weights connected to them. This prevents co-adaptation between neurons (Srivastava et al., 2014a), and empirically results in improved generalisation across a wide range of architectures and tasks (Labach et al., 2019). Standard dropout may also be interpreted as sampling a "thinned" architecture from an exponential number of related networks (2^d if the layer width is d) during training, and approximately ensembling these architectures at test time through first-moment propagation (Gal & Ghahramani, 2015).

Since the seminal dropout paper (Hinton et al., 2012), many alternative dropout schemes have been proposed to improve the efficiency of the technique across a wide range of different neural network architectures. We review the most relevant approaches related to our work.

Standard dropout. At each training step, the activations from neurons at a layer where dropout is applied are zeroed out with a probability p – the dropout probability for that layer – with the forward and backward passes being then performed as usual (Fig. 5.3a). During testing, all units of the original architecture are kept to perform the forward pass. Because a fraction p of units are dropped during training, activations need to be renormalised to preserve the expected value of pre-activations of subsequent layers between train and test, preserving the first and second moments of the layer's output. This normalisation may be performed at test time ("weight scaling inference rule", Goodfellow et al. (2016)), or during training ("inverted dropout"). The latter is the most popular approach used nowadays and consists of dividing each neuron at a layer where dropout is applied by the probability of it being kept (i.e., divided by $(1 - p)$).

Controlled dropout. Controlled dropout (ByungSoo Ko et al., 2017; Ko et al., 2017) was suggested to speed up the training of fully connected networks based on the observation that storing zeroed activations throughout the forward and backward pass leads to computational inefficiencies. The authors propose to keep a random subset of rows or columns of the activation tensors by performing a set of 'gather' operations (*gather ops*) on the corresponding network weights (Fig. 5.3). The gather ops select specific weight rows/columns, and allocate new memory into which these rows/columns are copied, so that subsequent multiplications in the forward and backward passes involve smaller tensors. Although this approach helps avoid unnecessary multiplications, the gather ops' memory allocations introduce significant overhead. More specifically, the GPU needs to perform a quadratic number of reads and writes in order to create the required reduced tensors. This is not only slow to perform, but also results in duplicating the gathered weight tensors data in memory (Table 5.1).

DropBlock & SpatialDropout. Convolutional neural networks require a different scheme than standard dropout to perform effective regularisation (Tompson et al., 2014; He et al., 2015). This is both due to the strong correlations between adjacent pixels present in natural images (and preserved in subsequent feature maps) and the fact convolution kernels operate on nearby pixels. Consequently, when a given pixel is zeroed out, information can still propagate through neighboring pixels as if no dropout had been applied. Several schemes have been proposed to circumvent this limitation, for example by zeroing out contiguous regions of the feature maps (Ghiasi et al., 2018) or zeroing out entire convolution filters (Tompson et al., 2014).

Further parallels may be drawn between SliceOut and Nested Dropout (Rippel et al., 2014), in which coherent nested sets of hidden units are dropped in order to learn ordered representations, and with DropEdge (Rong et al., 2019), in which a certain number of edges are removed from the input graph at each training epoch.

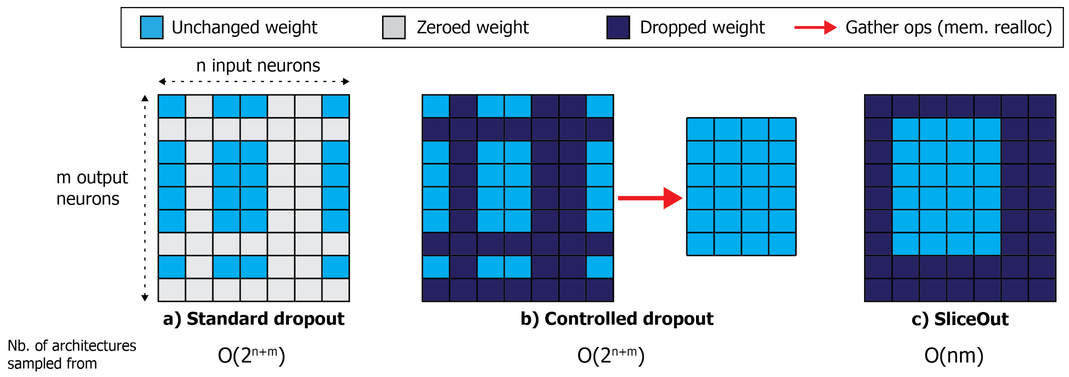


Figure 5.3: Representation of the effective weight dropout masks for different dropout schemes in a fully connected network. **a) Standard dropout:** entire rows/columns are set to zero (in practice we typically zero out the output tensor as opposed to the weight) **b) Controlled dropout:** similar to standard dropout, except non-zero weights are gathered and reallocated in new memory. **c) SliceOut:** structured weight dropout keeps contiguous set of rows/columns of weight tensors in-place.

5.2 SliceOut

SliceOut is a structured weight dropout scheme aimed at speeding up computations and reducing cached memory footprint, while preserving the regularisation benefits of standard dropout. We first convert the dropout rate into an expected number of nodes that should be kept at a layer where SliceOut is applied, i.e. the "slice width". During training, we uniformly sample the starting index of the slice (restricting to a subset of eligible positions), then "slice" (see next paragraph) the relevant rows and columns of the weights and biases that precede / follow the layer(s) where SliceOut is applied (Fig. 5.3). We then perform the forward and backward passes with the sliced weights and biases, updating the corresponding slice(s) of the original weight matrices in-place. We repeat this end-to-end process, sampling different slices at each step, until convergence (Algorithm 17). At test time, we use the full network without dropping any weights or biases, similar to standard dropout.

5.2.1 The slice op

Slicing is a fast and memory efficient operation: it selects the tensor elements of interest with a single memory access, and performs tensor operations with the logical tensors in-place (Harris et al., 2020; Paszke et al., 2019). The slice operation (*slice op*) only changes the logical view into the memory, but not the physical memory. When a GPU matmul or conv kernel (both GPU functions) is called, it only sees the weights within that view, and does its operation with those weights without having to move anything in memory. SliceOut enjoys speed-ups from performing forward and backward passes with smaller tensors; Furthermore, as we now need only keep

Table 5.1: Comparison of memory usage & No. of basic operations for different dropout schemes with b the batch size, n & m the No. of neurons in the input & output layers resp. & p the dropout probability applied to both input and output layers (with 0<p<1). **SliceOut** benefits from the same computation savings as **Controlled dropout**, without the memory reallocation overhead.

Metric	Standard dropout	Controlled dropout	SliceOut
No. extra read/writes to manipulate weights	–	$O((1-p)^2 * n * m)$	$O(1)$
Extra memory usage due to weight copy	–	$(1-p)^2 * n * m$	–
No. basic operations for weight multiply	$O(b * n * m)$	$O((1-p)^2 * n * m * b)$	$O((1-p)^2 * n * m * b)$
Size of output activations tensor	$m * b$	$(1-p) * m * b$	$(1-p) * m * b$

the smaller sliced activation tensors in memory to perform the backward pass at train time, we save on activation storage.

At test time we use the full network, and therefore there is no difference in memory usage to a network trained with standard dropout. However, the memory bottleneck for large networks is typically at train time since we are required to store intermediate activations to compute gradients on the backward pass.

5.2.2 Normalisation

After applying dropout, it is necessary to re-normalise activations in order to preserve the moments of their distributions and avoid the network outputs exploding or collapsing to zeros. We experimented with different approaches to normalise activations after dropout, and describe here the two that lead to the best results in experimental settings:

- **Flow normalisation:** We divide activations by the *expected proportion* of nodes kept at that layer during training (i.e., the ratio of the slice width to the full layer width). Intuitively, this helps keep constant the expected values of pre-activations at subsequent layers.
- **Probabilistic normalisation:** We divide each node by the probability of *this specific node* being kept during training. This helps ensure that, on average during training, the activations stemming from this particular node are equal to what they would be at test time.

These two normalisations coincide in the standard dropout case, where the expected proportion of nodes kept at a given layer is exactly equal to the probability of each node to be kept during training. This is not the case in SliceOut, as we impose constraints on eligible slices during sampling to avoid memory re-allocations and keep the size of tensors constant throughout training: nodes around the edges are less likely to be selected at a given training step.

5.2.3 Regularisation and ensembling

While standard dropout samples a “thinned” network from an exponential number of possible architectures, SliceOut samples from a linear or quadratic number of architectures.[1] As a result SliceOut can be seen as a milder regularisation scheme (for a fixed dropout probability value). We observe in several experimental settings that, beyond a certain dropout probability threshold, the performance drops more sharply in standard dropout than in SliceOut. This increased stability makes SliceOut less sensitive to the chosen dropout probability, enabling higher drop rates.

Algorithm 1 Slice dropout algorithm - Simple FFN with L hidden layers

```
Let $W_l$, with $l \in [1 - L]$, be the weights tensor of the $l^{th}$ hidden layer
Let $f_l(.)$ be the non-linearity applied at the $l^{th}$ layer
for $training\_step \leftarrow 1$ to $T$ do
    Sample mini-batch $(x, y)$
    for $layer_l \leftarrow 1$ to $L$ do
        Sample slice: $Slice_l = (start_l, end_l)$
    end for
    for $layer_l \leftarrow 1$ to $L$ do
        Slice weights:
        $W_{l\_slice} = W_l[(start_l, end_l), (start_{l-1}, end_{l-1})]$ ▷ where $(start_0, end_0)$ selects the full input
    end for
    Perform forward pass with sliced weights:
    for $layer_l \leftarrow 1$ to $L$ do
        $x \leftarrow f_l(norm(W_{l\_slice} \cdot x))$ ▷ where norm(.) is the activation normalisation applied post dropout
    end for
    Perform backward pass with sliced weights
end for
```

5.2.4 SliceOut and CNNs

Our SliceOut schemes for CNNs (Fig. 5.4) draw inspiration from the prior dropout schemes tailored to CNNs discussed in §5.1 (Tompson et al., 2014; Ghiasi et al., 2018):

- **Channel-SliceOut**: slicing contiguous sets of channels for a given convolution kernel

[1]If SliceOut is applied at only one layer, we only take slices row-wise of the corresponding weight vector (and column-wise of the subsequent weight vector), thereby sampling from a linear number of architectures. If SliceOut is applied at two consecutive layers, we slice the second weight matrix row and column wise, thereby sampling from a quadratic number of architectures (Appendix ??).

- **Patch-SliceOut**: slicing contiguous 2D chunks of the input activation tensors, and then performing the convolution

Channel-SliceOut builds on the SpatialDropout scheme (Tompson et al., 2014), with the critical difference that we directly slice the convolution kernels instead of zeroing out feature maps of the output activation tensor. This results in smaller output tensors and helps avoiding performing tensor operations for which the outcome will be ultimately be set to zero. Patch-SliceOut can be seen as performing the complement operation to what is done in Cutout (DeVries & Taylor, 2017) (on the input image), or more generally in DropBlock (Ghiasi et al., 2018), where units in a contiguous region of a feature map are dropped together, except that we slice out zeros instead of carrying them around.

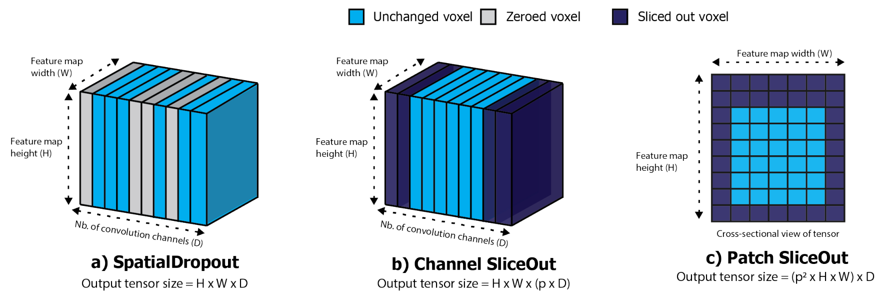


Figure 5.4: Comparison of the feature tensor of a convolution layer where different dropout schemes are applied
a) SpatialDropout: randomly sets entire convolution channels to zero. **b) Channel SliceOut** randomly selects a contiguous set of convolution channels, resulting in a more compact feature tensor (other channels are never allocated in memory) **c) Patch SliceOut:** selects a contiguous block of the input tensor across feature maps, then performs the convolution.

5.2.5 SliceOut and Transformers

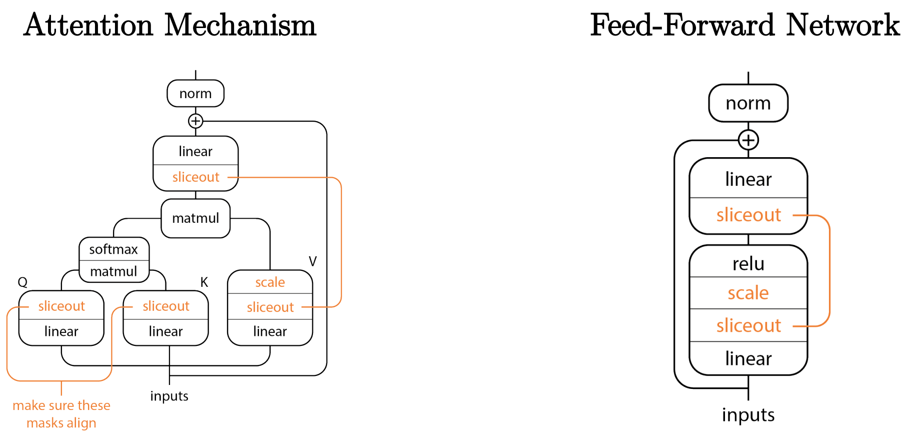


Figure 5.5: Transformer architecture with SliceOut

Transformers (Vaswani et al., 2017c) represent the state of the art across a host of natural language benchmarks and have seen adoption across academia and industry. One short-coming of the architecture is the considerable memory requirements demanded by the model architecture since Transformers tend to improve their performance dramatically with the number of parameters they are given. This observation has lead to several strategies to construct larger and better models (a quick overview of the Transformer architecture is given in Fig. 5.5).

SliceOut represents a complementary technique to the standard model-scaling measures taken in the literature (e.g., distributed data and model-parallelism, memory efficiency-focused optimisers (Shazeer & Stern, 2018)) and can be used in conjunction with them.

In our implementation of SliceOut in Transformers we do not normalise the queries and keys as in §5.2.2. Instead, we modify the temperature value (α) used in the attention weights:

$$W_{attn} = \text{softmax}\left(\frac{QK^\top}{\sqrt{\alpha}}\right)$$

In a Transformer α is generally set to the dimensionality of the vectors in the queries and keys, but in our case, SliceOut changes the dimensionality of those vectors during training, and so we adjust α to be the new dimensionality of these vector after SliceOut. We do still perform normalisation (§5.2.2) on the values and within the feed-forward networks (Fig. 5.5; Note: In the figure, normalisation is denoted "scale" while "norm" refers to layer normalisation, as in the original Transformer paper).

Since there is a dot product taken between each of the queries and keys, it is necessary that the sliced out indices of those vectors are aligned. That is, SliceOut slices out some contiguous set of elements from a query vector $Q_{\text{sliced}} = (q_i, \ldots, q_{q+d})$; it is, of course, extremely important than these indices are the same for the sliced keys $K_{\text{sliced}} = (k_i, \ldots, k_{q+d})$. Similarly, when slicing weight matrices we must ensure that the slices made along the leading dimension align with the slices applied to the incoming activation vector. See the orange lines in Fig. 5.5 for a pictorial description of indices that must be aligned.

5.3 Experimental results

We quantify the benefits of SliceOut across several neural network architectures and application domains: fully connected networks on MNIST and FashionMNIST datasets (§5.3.1), Wide ResNets on the CIFAR-10 and CIFAR-100 datasets (§5.3.2, EfficientNets on CIFAR-10/100 and ImageNet (5.3.3), and Transformers on the LM1B dataset (§5.3.4). For each experiment, we train the different networks until convergence, measure speedups based on the train time per epoch, and memory gains via the maximum GPU memory managed by the caching allocator at each epoch.

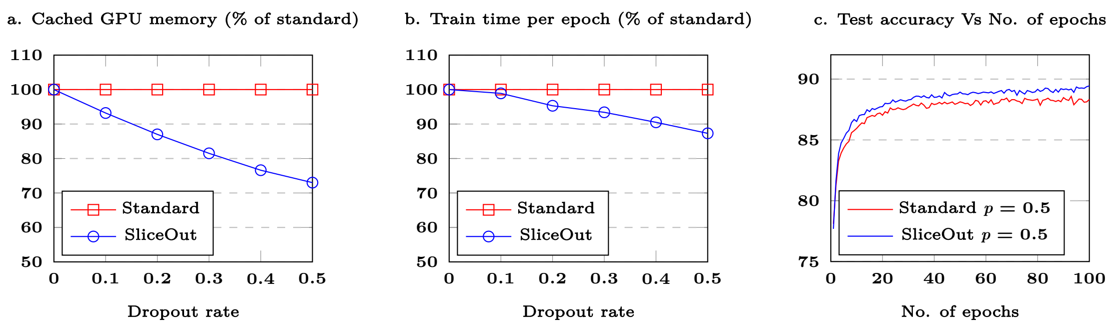


Figure 5.6: FashionMNIST - We achieve 30% memory savings (a.) and 15% training speedups (b.) from replacing standard dropout with SliceOut in a simple fully connected network with 3 hidden layers, converging slightly faster and to a higher test accuracy value (c., here for dropout rate = 0.5, although similar trends were observed for any dropout rate ≤ 0.5). Results were averaged over 4 independent runs

5.3.1 Fully connected networks

This first set of experiments is performed in a simpler setting aimed at studying the properties of our method with fully connected networks on the MNIST (Lecun et al., 1998) and FashionMNIST (Xiao et al., 2017) datasets.

In the FashionMNIST experiments, we observe not only speedups (up to 15%) and cached GPU memory savings (up to 30%) with SliceOut, we also converge faster and to a higher test accuracy value (Fig. 5.6) when typical dropout rates are applied (i.e., $p \leq 0.5$). The highest test accuracy obtained with SliceOut across all hyperparameter settings tested was $90.0 \pm 0.03\,\%$ (obtained with $p = 0.1$), while the highest with standard dropout (also for $p = 0.1$) was $89.6 \pm 0.08\,\%$ (no dropout lead to a test accuracy similar to the latter)

In the MNIST experiments, we observe similar speedups and memory gains from SliceOut, although there was no statistically significant difference in terms of top test accuracy.

Across both experiments, controlled dropout was converging to similar test accuracy values, but was systematically slower and less memory efficient than SliceOut.

5.3.2 Wide ResNets

Wide ResNets (Zagoruyko & Komodakis, 2016) are a variant of the original ResNet architecture that achieve higher test accuracy by simultaneously reducing the depth of the network and increasing the number of convolution filters in each residual block. The architecture strings together several "Wide-dropout" blocks, progressively increasing the number of channels and reducing the height & width of the activation tensors. Standard dropout is used critically in each residual block between the two

3x3 convolutions, to prevent potential overfitting resulting from the channel widening.

We remove the standard dropout layer in the original "Wide-dropout" block and experiment with our two SliceOut schemes for CNNs (see Fig. 5.4 and architecture diagram on Fig. 5.7):

- **Channel-SliceOut:** we apply SliceOut on the first 3x3 convolution across all residual blocks. It is critical to ensure that we operate on the same slice at the subsequent convolution layer, and the batch norm in-between;
- **Patch-SliceOut:** we apply Patch-SliceOut on the input tensor to the first 3x3 convolution, across all blocks.

For both schemes, performing the normalisation after the second batch norm and right before the final projection convolution ("delayed normalisation", Fig. 5.7) helps further increase test accuracy. We observe higher performance when using the Probabilistic normalisation scheme over the Flow normalisation (§5.2.2), and for Channel-SliceOut over Patch-SliceOut.

When using SliceOut across a range of Wide Resnet architectures on CIFAR-10/100 (Table. 5.2), we obtain training speedups of up to 35% and memory gains of up to 25% with no impact on test accuracy. This translates into a superior compute efficiency frontier (Fig. 5.1). For example, we are able to train a 46x12 architecture with SliceOut as fast as a 40x10 architecture without SliceOut, and achieve a higher test accuracy as a result.

Table 5.2: Wide ResNets results. Training time & Max cached GPU memory are respectively the relative train time speedups per epoch for a network trained with SliceOut Vs standard dropout, and the maximum cached GPU memory during training. Results are averaged over 5 independent runs. Reported baseline values (standard dropout) are obtained via an hyperparameter search over dropout rates and selecting the value yielding the highest test accuracy. SliceOut results are obtained with a 0.5 rate, Channel-SliceOut and Probabilistic normalisation.

Dataset	Architecture	Test accuracy Standard dropout	Test accuracy SliceOut	Training speedups	Max cached memory gains
CIFAR-10	28x6	96.1%	96.0%	-25%	-21%
	34x8	96.2%	96.2%	-26%	-21%
	40x10	96.3%	96.2%	-30%	-20%
	46x12	96.4%	96.2%	-29%	-25%
	52x14	96.4%	96.2%	-32%	-22%
CIFAR-100	28x6	79.9%	79.8%	-24%	-21%
	34x8	80.9%	80.7%	-26%	-21%
	40x10	81.3%	81.2%	-28%	-20%
	46x12	81.5%	81.4%	-30%	-25%
	52x14	81.6%	81.5%	-34%	-22%

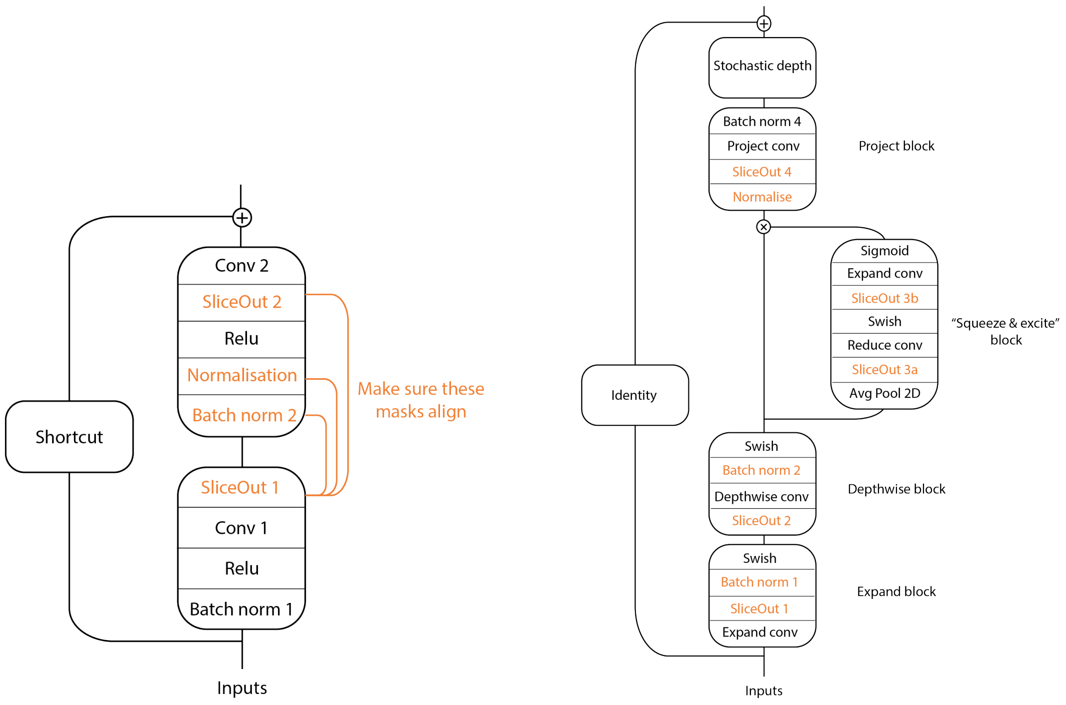


Figure 5.7: Wide ResNet residual block and EfficientNets MBConv block with SliceOut. The selected slices for the items in orange need to be aligned for a given forward/backward pass.

5.3.3 EfficientNets

EfficientNets (Tan & Le, 2019) achieve state of the art performance on several vision datasets including ImageNet (Russakovsky et al., 2014), while being more compute efficient than prior architectures at test time. The purpose of our EfficientNets experiments is two-fold: first, we demonstrate the scalability and generalisability of the SliceOut scheme to larger datasets and more complex architectures; second, we show that SliceOut can also be thought of as a method to accelerate model training, even when dropout is not used in the original architecture. In EfficientNets, dropout is not used in any of the mobile inverted bottleneck (MBConv) blocks that form the backbone of the architecture. [2] We use Channel-SliceOut to operate on the "expand convolution" (Fig. 5.7) of the first three stages of MBConv blocks, as this is where the largest tensors are created. Similar to what we observed with Wide ResNets, "delayed normalisation" (right before the final "projection" convolution) leads to higher test accuracy. We hypothesize that normalising earlier in the block leads to worse performance as it perturbs the statistics computed at train time for a given slice by the intermediate Batch normalisation layers. Since we add SliceOut in parts of the network where no regularization is needed, we turn off SliceOut in the last 10% of training epochs to bridge a potential gap in test accuracy with the original architecture.

[2]Standard dropout is applied on the last fully connected layer of the network, but using SliceOut there would not result in meaningful speedups.

In CIFAR-10/100 experiments, we fine-tune EfficientNet models pre-trained on ImageNet without SliceOut (following the same experimental setup as in Tan & Le (2019); Kornblith et al. (2018)). We observe speedups of over 20% when using SliceOut, with comparable test accuracy to performing the fine tuning without SliceOut. This demonstrates that SliceOut can be used to achieve speedups when fine tuning networks that were pre-trained without. In ImageNet experiments, we train the networks from scratch and observe speedups of up to 20% with SliceOut with similar test accuracy (Table 5.3), resulting in a more desirable compute efficiency frontier (Fig. 5.1). We train a B2 architecture with SliceOut (0.4) as fast as a B1 architecture but with higher test accuracy (79.8% Vs 78.8%).

Table 5.3: EfficientNet results - ImageNet. Training time is the relative % of train time per epoch for a network trained with SliceOut Vs standard B3 architecture trained on the same dataset without SliceOut. ImageNet results are obtained by training from scratch.

SliceOut	Test accuracy				Training time savings (rel. to B3 baseline)			
rate	B0	B1	B2	B3	B0	B1	B2	B3
None	77.1%	78.7%	79.7%	80.6%	-68%	-50%	-39%	0%
0.3	**77.2%**	**78.8%**	**79.8%**	**81.0%**	-70%	-55%	-46%	-12%
0.4	76.8%	**78.8%**	**79.8%**	80.7%	-72%	-57%	-47%	-18%
0.5	76.4%	78.5%	79.4%	80.7%	-72%	-58%	-52%	-20%

5.3.4 Transformers

Table 5.4: Transformer results. We observe speedups and memory gains of $\sim$ 10% when using SliceOut, despite the fact in Transformers the performance is dominated by looking up embedding vectors. Although Transformers are typically under-parametrised for language modeling on LM1B, SliceOut is a more effective form of regularization compared to standard dropout or controlled dropout.

Width	Dropout rate	Dropout Perplexity	Controlled Perplexity	SliceOut Perplexity	Training time	Max cached memory
1024	0.0	**31.7**	-	-	-	-
	0.3	45.1	45.7	**33.7**	-8%	-9%
2048	0.0	**28.1**	-	-	-	-
	0.3	88.6	53.1	**28.1**	-11%	-10%

In our experiments, we evaluate a vanilla Transformer language model on the popular "One Billion Word Benchmark" (Chelba et al., 2013a). Similar to what we do our EfficientNets experiments, we also perform a final fine tuning without SliceOut for the last 10% epochs. Our results are shown in Table 5.4. Given the complexity of language modeling on the LM1B dataset, we observe that test set perplexity is reduced across the board as we increase model width. In the larger width setting, we obtain identical perplexity with SliceOut as for models trained without (standard dropout is always detrimental to performance), while reducing memory overhead by

$\sim 10\%$ and reducing steptime by $\sim 10\%$. Speedups are more modest in comparison to CNNs, and this is primarily due to the fact that, in Transformers, a significant portion of steptime is spent looking up embedding vectors and computing logits over a vocabulary of more than 32,000 elements. Similarly, much of the Transformer's memory is spent on storing the parameters, which SliceOut does not reduce.

Chapter 6

ARC Networks

The following is work done in collaboration with Joost van Amersfoort, Lewis Smith, and Yarin Gal. In this project I was the source of the method, led experiment design and oversaw execution, and supervised the project alongside Yarin Gal.

One of the most significant issues with depth-wise parallel training – that is, training a neural network by distributing contiguous groups of its layers across a set of accelerators – is the *gradient locking problem*, where the earlier components must sit idle after performing their forward pass while they wait for a gradient to arrive from the latter components of the network. There is a much more complete analysis of is effect in Chapter 7.

In order to prevent the gradient locking problem, we need to divide our model into sub-components that are each optimised locally – i.e components that can perform a parameter update given only an input x_l and a target y, without waiting for a global error signal to propagate backwards from the top layer.

In standard deep learning, we have data X, Y, and we attempt to fit a function that predicts the output from the input, $\hat{y} = f(x)$, by minimising some loss metric on the output of the model, $L(\hat{y}, y)$. Our model is a composition of simple functions $f = (f_1 \circ f_2 ... f_n)$, and we optimise the parameters θ_i of each component by moving them in the direction of the gradient of the loss as a function of the global output, $\partial_{\theta_i} L(\hat{y}, y)$. This leads to the gradient locking problem – as discussed above, in order to compute the gradient $\partial_{\theta_i} L(\hat{y}, y)$ using the chain rule, we need to wait until we have computed the gradient of the layer above. In order to avoid the locking problem, we need to have that each component f_i can update its parameters with only access to its input x_i, output $x_{i+1} = f_i(x_i)$, and the global target y. The most obvious way to do this is to use a simple, local auxiliary mapping to the target $\hat{y}_i = \mathrm{softmax}(\mathrm{h_i}(\mathrm{x_{i+1}}))$ at each layer, and update each layer greedily based on this local learning signal as though each layer were the final one; that is, to update the parameters according to $\partial_{\theta_i} L(\hat{y}_i, y)$ Belilovsky et al. (2019a). This is essentially the same as one would do in supervised greedy pre-training except we intend to learn all of these components simultaneously in parallel.

The drawbacks of this approach are obvious – while this scheme meets our computational requirements, the updates to the functions are no longer acting to minimise the global loss, but a local, greedy heuristic instead. There is a risk that each component will settle down into a sub-optimal solution. In the next section, we discuss the problems inherent in asynchronous greedy learning schemes, and our architectural decisions to mitigate these problems.

Table 6.1: Standard ResNet-inspired networks on CIFAR-10. Each network has only three components – one for each block of the ResNet architecture. *Global* is an end-to-end baseline with no asynchrony.

	Acc. (global)	Acc. (local)
AC (ResNet32)	93.60%	90.59%
ARC (RevNet38)	93.56%	92.21%

An ARC Network (Asynchronous Reversible Component Network) matches the description given above with the subtle variation that each component function f_i is constrained to be a reversible function:

$$x_{i+1}^{(1)}, x_{i+1}^{(2)} = f(x_i^{(1)}, x_i^{(2)}) \quad \text{where} \quad \begin{aligned} x_{i+1}^{(1)} &= x_i^{(1)} + \mathcal{F}(x_i^{(2)}) \\ x_{i+1}^{(2)} &= x_i^{(2)} + \mathcal{G}(x_{i+1}^{(1)}) \end{aligned} \tag{6.1}$$

We can then describe an ARC Network using a directed graph of ARCs, along which activations flow – but not gradients.

We can enforce reversibility of a single ARC f_i by using a reversible ResNet block as the function f_i, ensuring that f_i cannot discard information about its input. For each ARC, an in-feed queue continuously streams batches of data through the ARC. As soon as the outputs have been computed they are communicated to the in-feed queues of adjacent downstream ARCs. Each in-feed queue is a last-in first-out cyclic queue – the replay of previous batches unlocks the forward synchrony imposed by the dependency of each ARC on its predecessor (Belilovsky et al. (2019a) discuss this in detail).

In the following sections we justify the introduction of this reversible network structure and – more generally – explore the pathologies that emerge in the local learning setting and offer techniques for mitigating them.

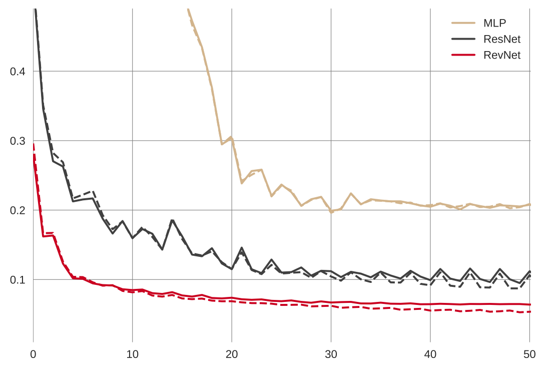


Figure 6.1: A comparison of simple toy networks to demonstrate the importance of information preservation. Three deep component networks trained on MNIST. Extremely thin layers and local losses demonstrate the importance of information propagation throughout the network. The reported error rate is of the final component of each network. The dashed lines measure training set error; solid lines, test set error. X axis is epoch, Y axis is error rate.

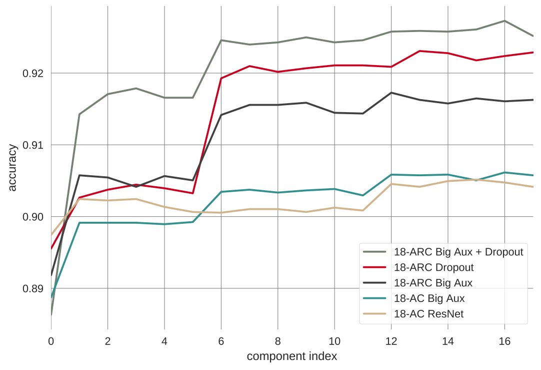


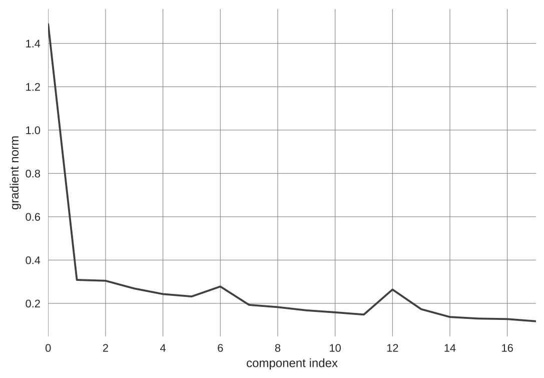
Figure 6.2: We compare five 18-component networks trained on CIFAR: One ARC Network trained with dropout at a rate of 50% applied to the first component's outputs; A second ARC Network trained with a larger auxiliary network composed of two stride two convolutions before the global average; A third ARC Network trained with both the larger auxiliary network and dropout; One non-reversible component network trained without any additions; And another non-reversible component network trained with the same larger auxiliary as above.

6.1 The Importance of Information Propagation

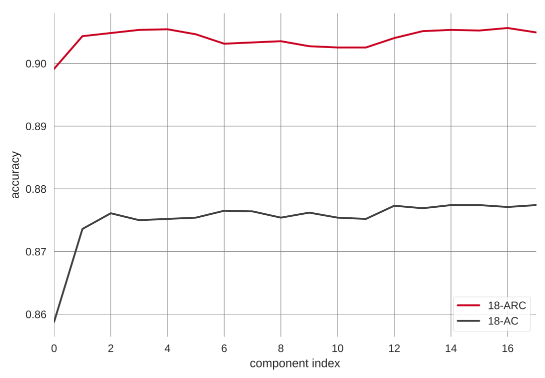
A core insight of our work is identifying and resolving a failure mode of component networks:

When gradient communication between components is cut, later components lose the ability to request specific features of the input data from their predecessors, leading to permanent and potentially catastrophic loss of information.

Here we demonstrate this phenomenon quantitatively. Our solution is to architecturally enforce the preservation of information by using reversible components which architecturally enforce the preservation of all information in the input, except when we *explicitly* choose to impose structured information loss by applying pooling operations. A simple demonstration of this effect is presented in Figure 6.1, we train three extremely simple component networks on MNIST: an MLP, a ResNet, and a RevNet. The main body (excluding the auxiliary layer leading to the logits) of each of the networks has ten weight matrices. To constrain the flow of information, each layer has only six neurons and every layer is treated as its own component. The networks



(a) The norm of the gradient taken through the cross-entropy loss – per component – after five epochs of training an 18-ARC network. The gradient signal drastically changes from the first to the second ARC, and decays gradually thereafter. This can have dramatic effects during training where other regularisation objective are present (e.g weight decay can begin to crush the later ARCs to identity functions).



(b) Two accuracy curves depicting negligible improvement between components of a network when the problem of identity collapse is not addressed. Contrast to consistent upwards trend in performance and converged accuracy given in Figure 6.2 using our methods tackling the problem of identity collapse.

Figure 6.3: Plots showing common indicators of the problem of identity collapse when models are under-regularised, trained on CIFAR10.

are all trained for fifty epochs using stochastic gradient descent with learning rate 10^{-3} and momentum 0.9.

The result shows the importance of information propagation; an MLP fails to exceed 80% accuracy, which grossly under-performs a simple linear classifier on the dataset, meaning crucial information about the input digit has been lost. The ResNet – whose residual function consists of a single ReLU-activated densely connected layer – is able to carry information much further (as was pointed out by Behrmann et al. (2018), a residual network is invertible if its residual function is a contraction) and, consequently, vastly outperforms the simple MLP. The RevNet – whose function preserves all information after the first densely connected layer – out-performs both of these architectures and demonstrates a much more stable learning curve.

6.2 The Problem of Identity Collapse

The second major failure mode of training component networks is the *problem of identity collapse*, where a component falls into the local optimum of a simple identity function. The component fails to improve on the previous component, resulting in wasted compute cycles. The problem is particularly difficult to overcome as the scenario described next is easily encountered by a sufficiently complex component with enough capacity to memorise the training set[1].

The problem of identity collapse:

- *Component one is an under-regularised network.*
- *Component one is rapidly optimised to a solution that generalises poorly (as a consequence of the component's regularisation). Component one has nearly zero training set error.*
- *Component two receives as input activations that – when near-identity is learned – result in zero error on the training set. This eliminates any gradient signal for the component, and settles the component into the same solution as its predecessor; that is, a solution that generalises as poorly as the first component.*

This is obviously a catastrophic failure for optimisation, resulting in large component networks that generalise as well as the small, poorly-generalising first component. Moreover, the very class of architectures that have come to dominate across data domains – residual networks – are at the most extreme risk of exhibiting this behaviour. A residual network can learn the identity simply by suppressing the parameters of the residual functions.

Overcoming this problem begins by studying the change in learning dynamics that arise in the component network setting. Each component effectively constructs a brand new dataset for the next component to be trained on, and as a consequence, optimisation dynamics on each of these new tasks may – and empirically, *do* – differ drastically from the original task.

In Figure 6.3 we plot the norm of the gradient and the accuracies of eighteen-component networks that *are not regularised to avoid the problem of identity collapse.* The plot shows that the gradient drastically changes scale after the first component, even after only a handful of epochs spent training. The right plot shows how both the ARC and AC network have their accuracy dominated by the first few components in this situation, and the remaining component learn a solution very close to the identity thereafter.

Another consequence of this drastic change in gradient scale is the amplified effects of weight decay. Components after the first suffer from having their parameters rapidly driven towards zero; resulting in near identity. To prevent this, we find that removing weight decay in all but the first component mitigates the issue – although we acknowledge that weight decay plays an often crucial element of optimisation and future work will need to address how best to reintroduce the method and decide its scaling relative to the cross-entropy.

[1]Note that Zhang et al. (2016) demonstrate even a modestly sized neural network is capable of this.

We find that regularisation explicitly intended to disrupt a component's ability to simply learn the identity can have a fairly dramatic effect. For instance, during train time we apply a single dropout layer to the activations passed forwards from the first component and observe a much more natural progression of task performance form one component to the next (see Figure 6.2). The hypothesis being that by disrupting the activations of the first component, subsequent components are forced to learn solution strategies beyond the identity. Table 6.1 (*'local'*) summarises the effects of regularisation on the identity collapse problem, reporting best model performance for AC nets vs ARC nets.

6.3 Reversibility and Downsampling

In practice, using fully reversible networks presents something of a problem. A reversible transformation clearly cannot reduce the dimensionality of its input, which precludes us from using spatial pooling operations like max pooling, average pooling, or strided convolutions – which are all common in practical network designs. These operations are valuable because they reduce the size of the activation map which needs to be stored, reducing computational cost, but also because they enforce a kind of prior knowledge – that at a certain level, fine grained spatial details are not relevant to the task at hand. We find in practice that adding pooling-blocks drastically improves the performance of networks that otherwise consist entirely of reversible components (pooling is applied at components 6 and 12 for all models in Figures 6.2 and 6.3 for a fair comparison). Why do we find that this information loss is beneficial, yet using reversible transformations improves performance?

We *hypothesise* that this is because spatial pooling operations are *structured* information loss – when designing the model, we know a priori that we want our final model to be insensitive to small translations of image components, which we achieve by using convolutions followed by pooling operations (Bruna & Mallat, 2013). However, by using reversible components, we still prevent the loss of any information other than the very specific kinds of information loss imposed by the design of our network, so the network is not free to throw away any other information about the input which might be encouraged by the local loss.

6.4 Conclusion

While the results of introducing reversibility and regularisation certainly do dramatically improve performance of the local learning setting, they do not *completely* erase the performance gap relative to global learning (Table 6.1). This lingering delta in performance between the two methods motivates the follow Chapter, where we introduce an alternative to local learning that allows for a continuous ablation between the local and global learning settings – enabling the user to choose the tradeoff between optimisation quality and efficiency.

Chapter 7

Interlocking Backpropagation

Local learning aims to optimise neural networks by computing parameter updates from multiple losses distributed throughout the network's computation graph. Generally, parameter updates will only incorporate the gradient received from their nearest loss, meaning they will not need to wait for all components of the neural network to be computed before being applied. This leads to theoretically faster step times as the method is amenable to taking advantage of pipeline parallelism across multiple accelerators. Below I explore the setting of local learning and discuss its limitations and introduce a method for overcoming some of them while preserving theoretical efficiency benefits.

The following is an excerpt from Gomez et al. (2022) and is published in The Journal of Machine Learning Research. This is work done in collaboration with Oscar Key, Kuba Perlin, Stephen Gou, Nick Frosst, Jeff Dean, and Yarin Gal. In this project I was the source of the method, led experiment design and oversaw execution, and supervised the project alongside Yarin Gal.

Modern state-of-the-art language models require billions of parameters. These models are often too large to fit in the memory of a single accelerator, and so the training computation must be distributed across multiple accelerator devices. Training such large models can be accomplished by partitioning the model across several accelerators and communicating the activations and gradients between them. Training such a model in the naive way incurs significant inefficiencies, as each accelerator must wait for all downstream accelerators to compute their forwards and backwards passes before it can begin computation of its own backwards pass. This optimisation setting is referred to as 'global learning', as there is a single global objective that must be evaluated in order to compute updates to the parameters.

An idealised model-parallel optimisation setting would be one where each accelerator need only push data to the next, never waiting for any returning gradient. In order to facilitate this, each accelerator's portion of the model must be able to compute weight updates with which to train itself, without access to any information from downstream accelerators. This idealised setting is referred to as 'local learning' and has seen an uptick in recent interest (Löwe et al., 2019; Belilovsky et al., 2019b). However, there remain core limitations to the proposed methods, principal among them: the degradation in modelling performance relative to global learning.

In this work we attempt to improve the efficiency of distributed model training by exploring strategies that strike a middle-ground between local and global learning via backpropagation. We train large scale neural networks with auxiliary classification layers throughout the network. We then explore various training regimes by restricting the gradient flow from each of these classification heads. We refer to these strategies as interlocking backpropagation. We find that interlocking backpropagation is significantly more compute efficient than the standard global backpropagation approach, yet it achieves similar test accuracy. In some cases it even outperforms the global baseline. Our work presents the following contributions:

- We explore modelling limitations of local optimisation.
- We propose a class of optimisation algorithms that aim to *preserve much of the compute efficiency* of local training, while *significantly improving modelling performance.*
- We provide a generic, open-source framework for the study of optimisation of locally trained networks. This is available at `https://github.com/oscarkey/interlocking-backprop`.

7.1 Methods

A neural network can be described as a composition of a series of smaller functions; for example $f = f_6 \circ \cdots \circ f_2 \circ f_1$. When the parameterization of the network exceeds the limit of a single hardware accelerator, contiguous groups of these functions can be placed on individual accelerators. Each of these contiguous groups is referred to as a *module.* The communication between these modules can be costly and so one could attempt to speed up the learning process by performing local learning on each module.

Consider a network composed of three modules of two layers each:

$$\begin{aligned} f &= f_{c_3} \circ f_{c_2} \circ f_{c_1} \\ \text{where, } f_{c_1} &= f_2 \circ f_1, \text{ parameterized by } \theta_{c_1} = (\theta_2, \theta_1) \\ f_{c_2} &= f_4 \circ f_3, \text{ parameterized by } \theta_{c_2} = (\theta_4, \theta_3) \\ f_{c_3} &= f_6 \circ f_5, \text{ parameterized by } \theta_{c_3} = (\theta_6, \theta_5) \end{aligned}$$

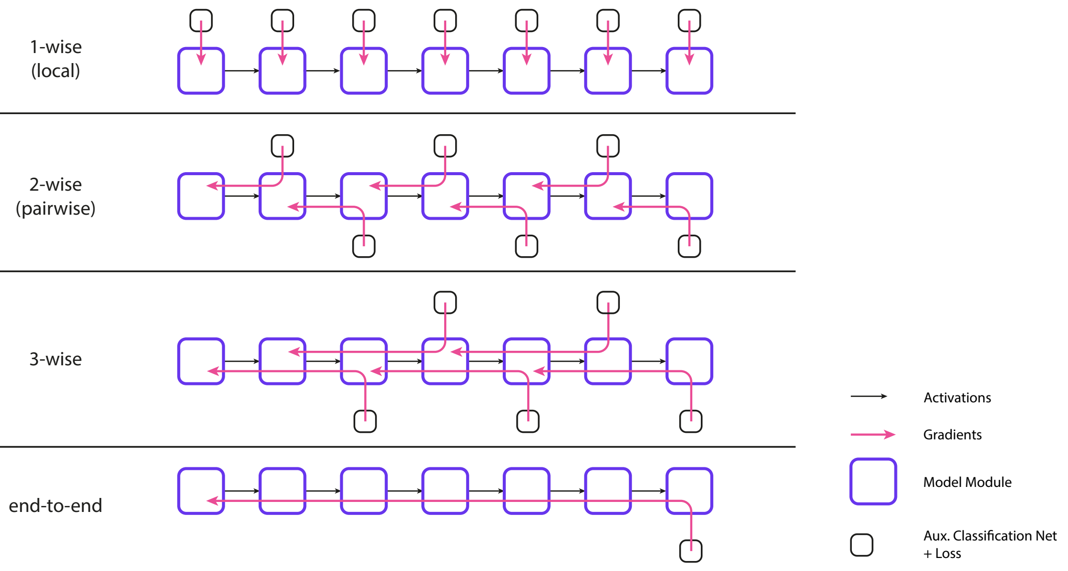


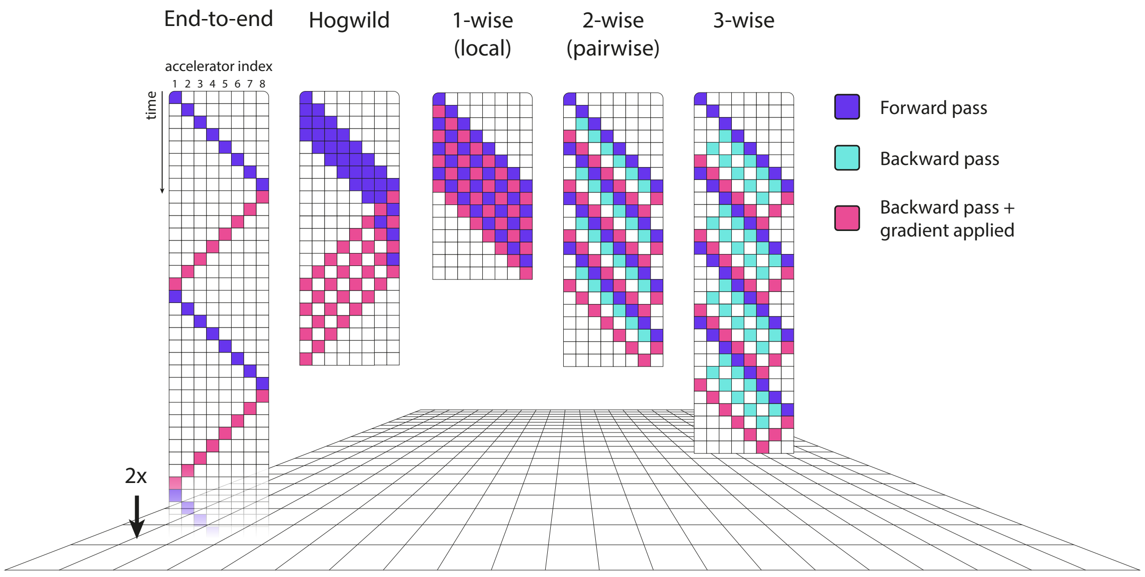
Figure 7.1: Depiction of the flow of activations and gradients through interlocking backpropagation for different optimization strategies. Activation flows are shown in black and gradients are shown in red. One extreme is 1-wise optimization, where there is no gradient communication between modules. The other extreme is end-to-end optimization, where gradients flow through all modules from a global loss function at the top of the network. The 2-wise and 3-wise strategies, as introduced in this paper, strike a middle ground. In 2-wise, gradients flow from an auxiliary network attached to each module, through the local module, and travel one module boundary before stopping. Similarly, 3-wise has gradients travel through two module boundaries before stopping.

We consider several possible approaches for training this model, which differ in the amount of communication between modules. One extreme, involving the most communication, is end-to-end training. Here we compute the loss based on the output of the final module, f_{c_3}, and propagate the loss backwards through each module to update their parameters. This approach is depicted in the bottom row of Figure 7.1. This approach achieves identical accuracy to if the model was in a single module on a single accelerator, however the communication between modules during the backwards pass leads to inefficiencies. If we consider the first module in the model, having completed its forward pass it must sit idle while it waits for the modules above it to complete their forward and backward passes, before it receives the gradient signal and can perform its own backwards pass.

The other extreme, requiring the least communication, is local training. This is shown in the first row of Figure 7.1. In this setting we augment each module with a local loss function, $\mathcal{L}_{c_k}$:

$$\mathcal{L}_{c_k}(x, y) = \mathcal{L}(\hat{y}_{c_k}(x), y)$$
$$\text{where, } \hat{y}_{c_k}(x) = h_{c_k}(f_{c_k} \circ \cdots \circ f_{c_1}(x)).$$

Here x is the training input to the model, y is the target, and $\mathcal{L}$ is a standard loss function, such as the cross-entropy loss. We call h_{c_k} the auxiliary network for



end-to-end	hogwild	1-wise (local)	2-wise	3-wise	n-wise
$2A$	2	2	4	6	$2N$

Figure 7.2: (top) Four training steps of different distributed optimisation strategies. 2-wise and 3-wise interlocking backpropagation, as introduced in this paper, are far cheaper than end-to-end in terms of total optimisation time, and offer a natural trade off between speed and optimization performance. (bottom) Table comparing the scaling of time per batch for different optimization strategies applied to A accelerators. Hogwild achieves its optimal time per batch when run long enough to fill its pipeline, which is not illustrated above.

module k. It produces predictions for the task directly from the outputs of the kth module. During training, we update the parameters of both the main and auxiliary networks of the kth module based *only* on gradients from $\mathcal{L}_{c_k}$. This means that during the backwards pass no communication is required between modules. Thus, the only communication necessary between the hardware accelerators holding each module is to propagate the activations during the forward pass. This strategy avoids accelerators idling during the backward pass, while they wait to receive gradients from subsequent accelerators.

While local training is time efficient, without backwards communication between modules it fails to match end-to-end in test accuracy. In this work we address this problem by introducing new intermediate strategies between end-to-end and local training, where we allow varying amounts of communication between modules . We refer to this family of strategies as n-wise interlocking backpropagation. Figure 7.1 illustrates 2-wise and 3-wise. Here the parameters in module k are updated using gradients from $\mathcal{L}_{c_{k+(N-1)}}$, which have been propagated backwards through the intermediate modules. When N is set to 1, this is equivalent to local optimisation (Belilovsky et al., 2019b) (i.e 1-wise); when N is set to the number of modules, this

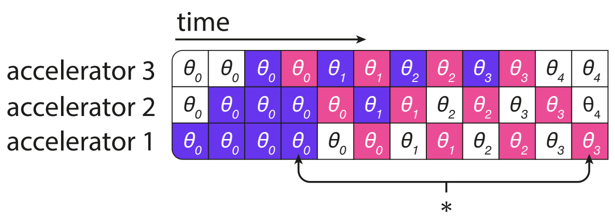


Figure 7.3: Depiction of the problem of stale gradients in Hogwild-style training. The θ_is denote the parameters for the corresponding accelerator's module at step i. (*) During the forward pass of the fourth batch of data, the first module computes its activations using the parameters θ_0 (left arrow); however, during the fourth batch's backwards pass the first module has already had it's parameters updated 3 times to θ_3 (right arrow). This mismatch between the weights used to compute the activations and those used to compute the gradient can disrupt optimisation dramatically (see Appendix Table 2).

is equivalent to global optimisation (i.e end-to-end) with inception-net style auxiliary losses. By selecting N, we can specify the trade-off between performance and accuracy to match our application.

During n-wise testing we only make predictions using the output of the final module, and this is what we use to compute test accuracy. An alternative approach would be to ensemble the predictions of the auxiliary network at each module. However, experimentally we find that this does not significantly increase performance, likely due to the modules being highly correlated, and in some cases earlier modules having much lower accuracy than later ones (see Appendix A).

The step times of these strategies are visualised in Figure 7.2. It shows that 2-wise and 3-wise are substantially faster than end-to-end training, yet in the next section we show that they can recover much of the performance on both image classification and Transformer language modelling tasks.

7.2 Training Speed of Interlocking Backpropagation

Interlocking backpropagation allows shorter training step times than end-to-end learning, as the gradients do not need to pass through the entire network. The step time can be controlled and reduced dramatically by lowering the N parameter of n-wise.

In interlocking backpropagation, multiple batches may be processed simultaneously by different modules of the network, making it another instantiation of pipeline parallelism, such as that achieved by GPipe (Huang et al., 2018b). In fact, interlocking backpropagation is orthogonal to the micro-batching done in GPipe, which splits a batch of data into smaller *micro-batches* that are fed through the model one-by-one similar to Hogwild (Recht et al., 2011), however, during the backward pass, instead of immediately applying the gradients from each micro-batch, the gradients are accumulated and only applied after all micro-batches have been included. This method can mitigate the gradient locking problem, but as the number of accelerators increase the micro-batch size will decrease and compute utilization will be negatively effected. Combining both GPipe-style parallelism with interlocking backpropagation yields additional benefits, as discussed below.

To understand the timing characteristics of n-wise in more detail, we propose a general algebraic model of *time per batch* for synchronous model-distributed learning. Our model abstracts local learning, end-to-end learning, GPipe, and interlocking back-propagation. Those training paradigms – and combinations thereof – are captured by the following three parameters:

- A – the number of sequential accelerators (modules).
- M – the number of micro-batches per mini-batch. $M = 1$ corresponds to not applying micro-batching.
- N – the n-wise parameter, in the range $[1, A]$. $N = 1$ and $N = A$ correspond to local learning and end-to-end learning, respectively.

As we are modelling a synchronous setting, we split the time domain into segments of equal duration. A grid with rows corresponding to accelerators, and columns corresponding to time slots, can be used to illustrate the processing done by the different learning strategies. An example timing diagram, with no micro-batching, is shown in Figure 7.2.

We denote the micro-batch processing cost $c(M)$, which is defined as the duration of a single time slot. This model makes the simplifying assumption that a forward and backward pass are the same duration.

The time per mini-batch (equivalently, per one gradient update) can then be derived as:

$$T(A, M, N) = \begin{cases} (2 + A - N) \cdot Mc(M) + 2(2N - A - 1) \cdot c(M) & \text{(for } M > 2,\, A < 2N - 1) \\ (N + 1) \cdot Mc(M) & \text{(for } M > 2,\, A \geq 2N - 1) \\ (N + 1) \cdot Mc(M) & \text{(for } M = 2) \\ 2N \cdot Mc(M) & \text{(for } M = 1) \end{cases}$$

For end-to-end learning ($N = A$), the formula degenerates to $T(A, M) = 2(M + A - 1) \cdot c(M)$. Note that for n-wise with $N < 1 + A/2$, the time per batch is completely independent of the total number of modules of the network. Thus the timing benefits of n-wise are most pronounced for big models, distributed across a large number of accelerators. Details of the model derivation, micro-batching timing diagrams, and fit of the c_0, c_1 parameters to experimental timing data, are presented in Appendix **??**.

The micro-batch cost model $c(M) := c_0 + c_1/M$ yields a good fit to our exper-

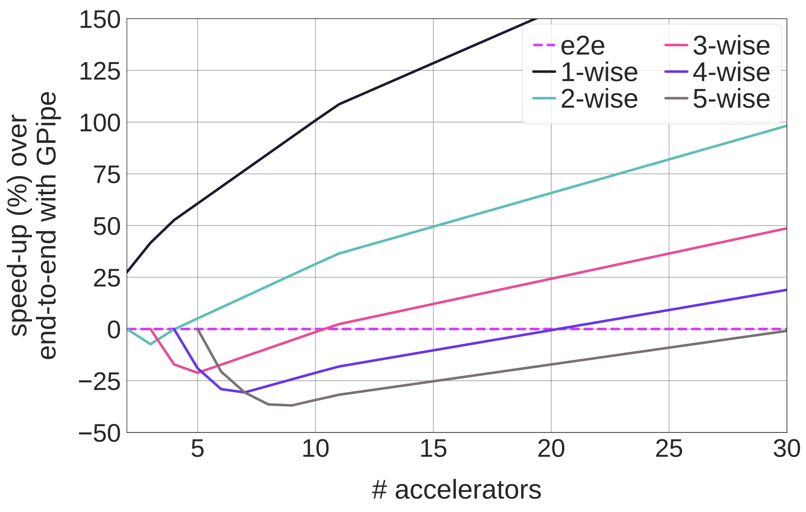


Figure 7.4: Modelled time per batch speed-up for n-wise, compared to end-to-end with GPipe, for varying numbers of accelerators. The optimal number of micro-batches is assumed at each point of the plot. Micro-batch cost $c(M) = 0.025\text{s} + 1.279\text{s}/M$, tuned to our Transformer experiments, is used.

imental data. This formula accounts for both a constant overhead present for each micro-batch (e.g. waiting times in inter-accelerator communication) and a processing time proportional to the number of examples in a micro-batch. For that cost function, the benefits of micro-batching are maximised at $M \propto \sqrt{A}$ for end-to-end learning, while $M = 2$ is optimal for n-wise (assuming $N < 1 + A/2$).

For optimal choices of M, and the cost function parameters ($c_0 = 0.025\text{s}$, $c_1 = 1.279\text{s}$) tuned to results of our Transformer experiments, the model predicts a 50% speed-up of time per batch (compared to end-to-end) for 2-wise at 15 or more accelerators (see Figure 7.4).

7.3 Information Flow in Interlocking Backpropagation

Unlike local training, in n-wise interlocking backpropagation the gradient from the loss at the final layer affects all the parameters of the network, it just does so indirectly. With 2-wise, module 2 optimises for both its local loss, and the loss of module 3. The module 3 optimises its loss and the loss of module 4, and so on until the final loss. As a result, there is indirect communication from modules at the head of the model to previous modules.

Consider a network with n modules trained using 2-wise:

$$f = f_{c_n} \circ \cdots \circ f_{c_1}$$

Each auxiliary network approximates the composition all of the modules above it:

$$h_{c_k} \approx f_{c_n} \circ \cdots \circ f_{c_{k+1}}$$

For training to work effectively this approximation should be as close as possible, so that the gradient computed from $\mathcal{L}_{c_k}$ encourages f_{c_k} to learn a representation which is useful for the modules above.

In 1-wise training, several factors discourage h_{c_k} from being a good approximation of the remainder of the modules. As h_{c_k} has much lower capacity than the rest of the network in the modules above, it may encourage f_{c_k} to greedily learn a simpler representation which is more amenable to immediately computing the logits, rather than feeding into the subsequent modules. This simpler representation may throw away information which the subsequent modules could use to achieve higher test accuracy.

In 2-wise training we hope that the communication between modules will allow lower modules to learn a representation that is useful for the modules above. We argue that this could happen by starting at the head and walking down the model. The penultimate module $f_{c_{n-1}}$ is updated using gradients which have propagated from the true loss at the head of the network and through f_{c_n}. Thus, $f_{c_{n-1}}$ will learn the most useful representation for $f_{c_{n-1}}$, rather than learning a representation which improves the performance of $h_{c_{n-1}}$. Now we examine the updates of the $f_{c_{n-2}}$. This module is updated with gradients that propagate from $\mathcal{L}_{c_{n-1}}$, and so depend on $h_{c_{n-1}}$. If $h_{c_{n-1}}$ is a close approximation to f_{c_n}, then these gradients will push $f_{c_{n-2}}$ towards a function which outputs a useful representation for both $f_{c_{n-1}}$ and f_{c_n}. As $h_{c_{n-1}}$ and f_{c_n} both have the same inputs and targets we hope that $h_{c_{n-1}}$ should become a close approximation to f_{c_n}, as close as possible given the limited capacity of $h_{c_{n-1}}$. Thus, $f_{c_{n-2}}$ should learn a useful representation for $f_{c_{n-1}}$ and f_{c_n}. We can continue to extend this reasoning down the model.

7.4 Experiments

Here we present results of experiments in both the image and language domains. We compare our n-wise strategies to both extremes of local learning and end-to-end training.

7.4.1 Validation of Approach Using a Small Convolutional Network

We investigate the behaviour of our method when training a small convolutional network on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). We consider models

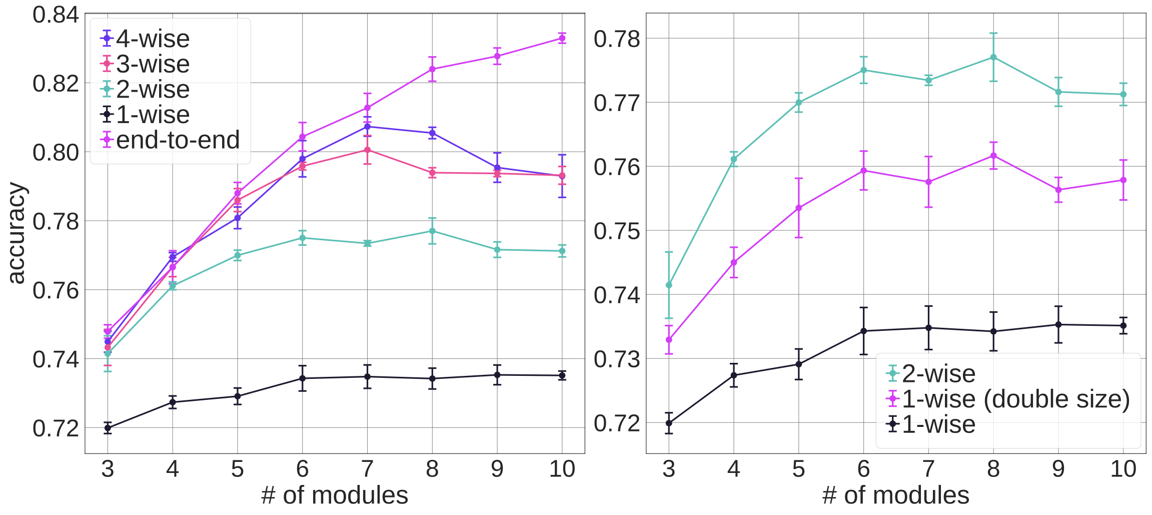


Figure 7.5: (left) Test accuracy of small convolutional models on CIFAR-10, comparing across depth and training method. (right) Comparison demonstrating that simply doubling the size of modules trained locally does not recover the full benefits of 2-wise training. In both cases the error bars show one standard deviation over four random seeds.

of 3 to 10 convolutional layers. Each module of the model contains a single convolutional layer and batch norm, with the final two modules also containing max pool layers. The auxiliary networks used for the local losses are comprised of a single linear layer. Appendix B gives full details of the experiment configuration. We train the model using several approaches: end-to-end, 1-wise, 2-wise, 3-wise and 4-wise. Figure 7.5 shows how the different approaches to training / as we increase the number of modules in each model. Unsurprisingly we see that end-to-end optimisation results in the best test accuracy, and local optimisation results in the worst. We note that pair-wise interlocking backpropagation training provides a clear improvement over training with only the local loss. For models with 6 or less modules, we find that n-wise training achieves comparable accuracy as end-to-end training within standard deviation. This shows that intermediate strategies between local and global optimization provide a good alternative to both, maintaining much of the test accuracy of global optimization, while drastically decreasing training time. This increased performance however degrades as we increase the number of modules. These results imply that this strategy will be most effective with networks with a small number of modules. While this is only a toy setup, these improvements are also seen when we examine ResNets and Transformer networks later in the paper. This confirms that n-wise interlocking backpropagation training allows the user to trade-off some test accuracy for a significant boost in efficiency when compared to end-to-end.

In order to understand the interplay between the number and size of the modules in the network and the optimisation strategy, we compare our 2-wise training scheme to an alternative approach with similar time complexity in Figure 7.5 (right). This approach, labelled "1-wise (double size)", is equivalent to performing 1-wise training, but with adjacent pairs of modules merged to give half the number of modules, each of which is twice the size. As merging modules is not possible in practice – each module would be large enough to fill an entire accelerator – we could implement this method

		end-to-end	1-wise	2-wise	3-wise
CIFAR-10	ResNet-32	95.20 (0.11)	94.20 (0.09)	95.05 (0.09)	**95.42** (0.06)
CIFAR-100	ResNet-32	76.71 (0.14)	75.02 (0.09)	**78.09** (0.13)	77.84 (0.04)
ImageNet	ResNet-50	75.60	72.05	74.45	**76.27**

Table 7.1: Accuracy of ResNet-32 and ResNet-50. For CIFAR we give the accuracy on the test set, for ImageNet we give the accuracy on the validation set. One standard error over three seeds is given in brackets for CIFAR. Bold indicates the best performing strategy.

by grouping modules into blocking pairs. We are interested in a comparison with this method because it is similar to 2-wise, except there is no possible communication between modules which are not directly adjacent. With 2-wise we hope that the fact that the pairs of modules are overlapping will allow indirect communication further down the model than the adjacent module that the gradients are passed to. Figure 7.5 shows that this approach performs better than 1-wise, but not as well as 2-wise, which may suggest that this additional communication is in fact taking place in 2-wise.

7.4.2 Image Domain Results Using ResNet Models

Having investigated our method in a toy setting, in this section we demonstrate that it continues to lead to improved performance with a more realistic model architecture. In particular, we consider ResNets (He et al., 2016b) on CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009). While a ResNet is usually sufficiently small to fit on a single accelerator, these results suggest that our method would work with significantly larger vision models which require multiple accelerators. In the next section we consider a language model, a Transformer, which is is too large to fit on a single accelerator.

Table 7.1 compares the performance of different training schemes for a ResNet-32 and ResNet-50. In both cases the model is split into four modules as follows. Referring to He et al. (2016a, Table 1), for the ResNet-50 the first module contains the layers labeled 'conv1' and 'conv2_x', the second module contains 'conv3_x', the third 'conv4_x', and the forth 'conv5_x' and the output layers. The ResNet-32 is split similarly. For the auxiliary classification network we use two convolutional layers with batch norm, global average pooling and a single linear layer. The full experiment configuration is given in Appendix B. These results show that n-wise training substantially closes the performance gap between local and end-to-end training. The results also show that 3-wise training does not consistently offer better performance than 2-wise training which, given the results in Section 7.4.1, is what we would expect for a model with only 4 modules.

In fact we see that for both CIFAR-10 and CIFAR-100, interlocking backpropagation outperforms end-to-end training. This is a surprising finding, as one would expect that a model in which all features were trained to optimize the single global loss would

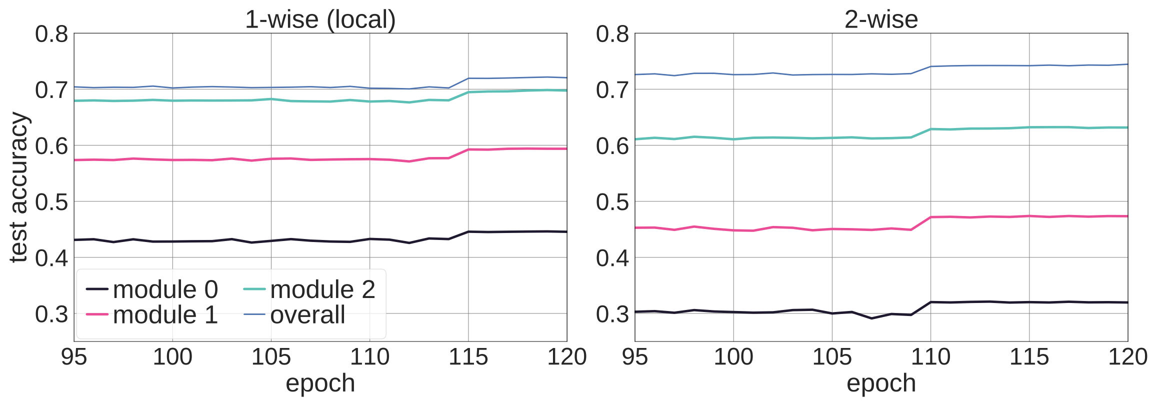


Figure 7.6: Test accuracy of each auxiliary classifiation head from a ResNet-50 model trained on ImageNet, trained with 1-wise (left) and 2-wise (right). 1-wise training leads to each module attempting to solve the entire problem on its own; this causes earlier modules to out-perform 2-wise, but ultimately, the final performance of 2-wise's more incremental solution strategy is better than 1-wise's.

outperform a model in which modules were optimised for local losses. Our results indicate that 2-wise training of large scale neural networks could outperform training equivalent models with end-to-end backpropagation. This surprising result may be explained by the success of Inception Nets (Szegedy et al., 2015), which found that adding auxiliary classification losses into the model improved training performance, leading to state of the art results at the time they were first published. We have argued that interlocking backpropagation may be able to propagate information from the top level loss to the initial layers. Inception Nets showed that auxiliary losses improved performance on CIFAR-10. The success of interlocking backpropagation in this setting may be understood as the consequence of these two findings.

We also use these ResNet experiments to further investigate the behaviour of each module in the network. Figure 7.6 shows the test accuracy of the outputs of the auxiliary network of each module in a network trained using 1-wise, compared against a network trained using 2-wise. Examining the accuracy of the model trained using 1-wise, we can see that the accuracy of module 2 is close to the accuracy of the overall model. This suggests that the model is encountering information loss, as later layers are unable to improve on the representations learned by earlier layers. The lower modules of the model learn a representation which is suited for the low capacity auxiliary network to map to the logits, throwing away useful information which the subsequent modules could otherwise use to improve accuracy further. In contrast, if we examine the 2-wise performance we notice that the training accuracy increases more gradually with each module, indicating that that this training regime is able to make use of the additional modules to improve performance.

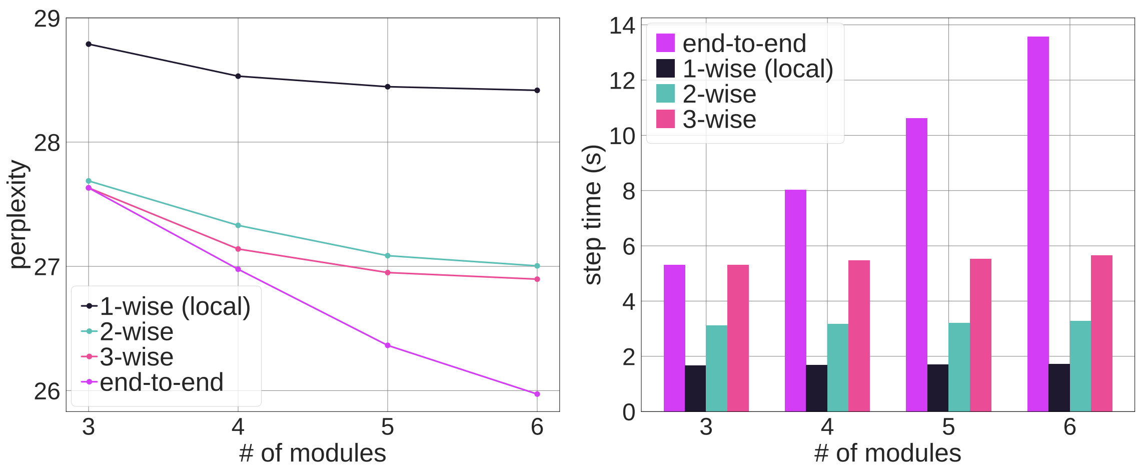


Figure 7.7: (left) Comparison of test perplexity of Transformer models across depth and training method. Models are run for a *fixed number of steps.* 2-wise recovers a large amount of lost model performance in local training. (right) Comparison of training step time (in seconds) of Transformer-based models across depth and training method.

7.4.3 Language Domain Results Using Transformer Models

Finally we investigated the performance of our method for training Transformer based models on language modelling tasks. The architecture we used largely follows the decoder only Transformer described in OpenAI's GPT-2 model (Radford et al., 2019b), with the addition of the auxiliary classification networks used for calculating the local losses. See Appendix B.3 for auxiliary network and training details. In these experiments each module is made out of 6 Transformer blocks. We run experiments with networks comprised of 3 to 6 modules. Each module was trained on a v3-8 TPU. We trained and evaluated the models with the One Billion Word Benchmark for Language Modelling (Chelba et al., 2013b). Each Transformer block module has a dimensionality of 1024. We train with a max sequence length of 128, and a batch size of 1024. For the experiments in Figure 7.7, we train for one epoch with the Adam optimiser; for the experiments in Figure 7.8 we train for 192 hours (eight days). We measure the performance of the models using perplexity, for which a lower value indicates better performance. Figure 7.7 shows the test set perplexity of models trained with 1-wise, 2-wise interlocking, 3-wise interlocking and, end-to-end backpropagation. We can see that, unsurprisingly, end-to-end greatly outperforms 1-wise training and the gap between them widens as we increase the size of the model. Interlocking backpropagation is able to make up much of the gap between 1-wise and end-to-end, but unlike our observations with CIFAR-10 and CIFAR-100, there is still a test perplexity gap between interlocking and end-to-end backpropagation for a fixed number of training steps.

In this real world setting we are able to substantially decrease the training time of these large models. The time per step for models of this size varies considerably based on the optimisation strategy used. Figure 7.7 visualises the test set perplexity and the

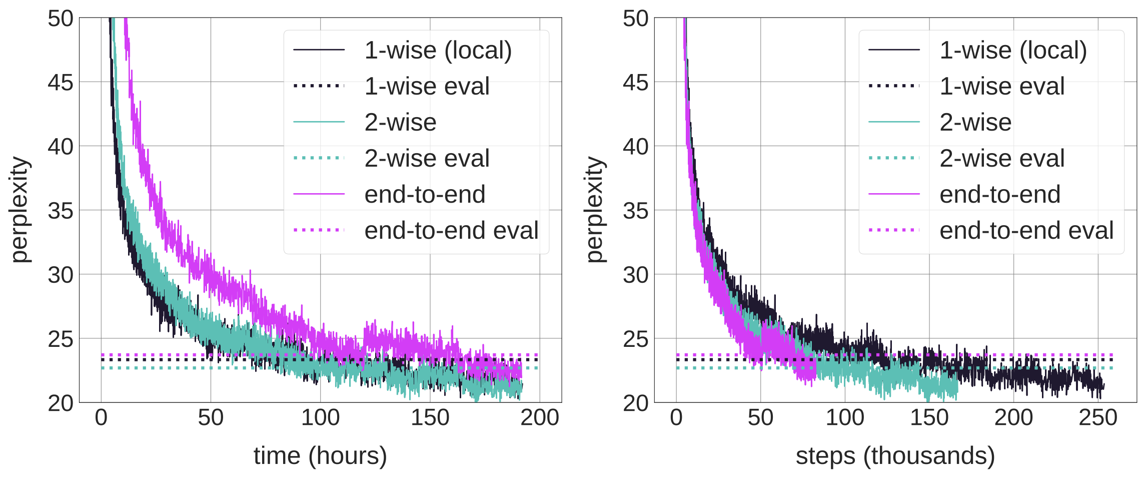


Figure 7.8: While Figure 7.7-(left) may suggest that end-to-end training always outperforms local training, it is important to keep in mind that this is still in the 'fixed steps' perspective; i.e we fix the number of steps each model is allowed to take and ignore the fact that running times may differ wildly. This figure exemplifies the importance of considering the 'fixed time' perspective: (left) depicts the training curves of 1-wise, 2-wise, and end-to-end in a 'per time' perspective; (right) depicts the same training curves in a 'per step' perspective. The difference between these two perspectives is quite extreme – from the 'per step' perspective, end-to-end training is best at any given point; however, when a 'per time' perspective is considered, the local learning strategies are best at any given point. Given a fixed time constraint, the logical decision to obtain the best possible model is to opt for a local learning strategy. The eval perplexities are measured at the end of each 8-day-long training.

train step time for models of various sizes trained with end-to-end, 1-wise, and 2-wise interlocking backpropagation. 2-wise interlocking backpropagation requires less than half the training step time of standard end-to-end training, to achieve similar test perplexity for models with 4 modules.

Figure 7.8 demonstrates the importance of considering a 'fixed time' perspective of training. Instead of fixing the number of optimiser steps, we fix the total elapsed training time to a set number of hours. The result is that methods which take gradient steps quicker will see many more weight updates relative to methods that are slower. When running for a fixed amount of time, the performance of interlocking backpropagation improves dramatically relative to end-to-end methods.

Chapter 8

Tabular Transformers

The primary means by which large language models' data is collected is via large-scale web scraping of text data. A limitation of the current sequence modelling regime is that there isn't an obvious formatting for modelling tabular data which is pervasive on the web. In this work we present an architectural change and learning rule that lets Transformers model tabular data.

The following work is an excerpt from Kossen et al. (2021) and is published in the 35th Conference on Neural Information Processing Systems. This is work done in collaboration with Jannik Kossen, Neil Band, Clare Lyle, Thomas Rainforth, and Yarin Gal. In this project I supervised the experimental design, in particular, the training strategies and data preparation, and I cosupervised the project with Yarin Gal and Thomas Rainforth.

From CNNs (LeCun et al., 1998b) to Transformers (Vaswani et al., 2017a), most of supervised deep learning relies on *parametric* modeling: models learn parameters $\boldsymbol{\theta}$ from a set of training data $\mathcal{D}_{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ to maximize training likelihoods $p(y \mid x; \boldsymbol{\theta})$ mapping from features $x \in X$ to target values $y \in Y$. At test time, they then make a prediction $p(y^* \mid x^*; \boldsymbol{\theta})$ that depends only on those parameters $\boldsymbol{\theta}$ and the test input x^*. That is, parametric models do not consider direct dependencies between datapoints.

This paper challenges parametric modeling as the dominant paradigm in deep learning. Based on the same end-to-end learning motivations that underpin deep learning itself, we consider giving models the *additional flexibility* of using training data *directly* when making predictions $p(y^* \mid x^*, \mathcal{D}_{\text{train}}; \boldsymbol{\theta})$.

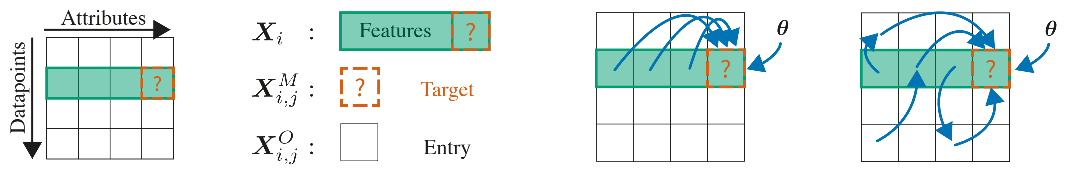
Concretely, we introduce **Non-Parametric Transformers (NPTs)**: a general deep learning architecture that takes the entire dataset as input and predicts by explicitly *learning* interactions between datapoints (Fig. 8.1). NPTs leverage both parametric and *non*-parametric predictive mechanisms, with the use of end-to-end training allowing the model to naturally learn from the data how to balance the two.

Namely, instead of just learning predictive functions from the features to the targets of independent datapoints, NPTs can also learn to reason about general relationships *between* inputs. We show that these models *learn* to look up information from other datapoints and capture the causal mechanism generating the data in semi-synthetic settings. However, unlike conventional non-parametric models, NPTs are not forced to *only* make predictions in this manner: they can also use the power of conventional parametric deep learning. We use multi-head self-attention Bahdanau et al. (2015); Vaswani et al. (2017a); Lee et al. (2019a) to model relationships between datapoints and construct a training objective for NPTs with a stochastic masking mechanism inspired by recent work in natural language processing Devlin et al. (2018).

A key contribution of this paper is opening the door to more general treatment of how deep learning models can make use of dependencies between datapoints for predictions. Our results demonstrate that NPTs make use of interactions between datapoints in practice, and we show highly competitive performance on several established tabular datasets as well as early image classification results. Additionally, we show that NPTs can solve complex reasoning tasks by combining representation learning and cross-datapoint lookup; something that is impossible for conventional deep learning or non-parametric models due to their inability to *learn* relations *between* datapoints.

Background. While questioning parametric modeling assumptions is unconventional in deep learning, in statistics so-called *non-parametric* models are a well-known and long-established field of study. Non-parametric models make predictions in explicit dependence of the training data $p(y^* \mid x^*, \mathcal{D}_{\text{train}})$. The most popular example of such models in the machine learning community are perhaps Gaussian Processes (Rasmussen, 2003). Non-parametric models typically do not require any training of parameters, and instead often directly interpolate between training points according to a fixed procedure, e.g., (Rasmussen, 2003, p.17). The interactions between inputs are fully defined by architectural choices and a small set of hyperparameters that must be carefully chosen. Conventional non-parametric models cannot *learn* – in the sense familiar to deep learning practitioners – interactions from the data, limiting the flexibility these models have in adapting to the data at hand. Approaches such as Deep Gaussian Processes (Damianou & Lawrence, 2013), Deep Kernel Learning (Wilson et al., 2016), and Neural Processes (Garnelo et al., 2018b,a; Kim et al., 2019) have all sought to apply ideas from deep neural networks to non-parametrics. Compared to NPTs, these approaches rely heavily on motivations from stochastic processes. This leads to them being either less flexible than NPTs or requiring strong assumptions on the data, making them *inapplicable* to the practical scenarios considered in this paper (cf. §8.2). Unlike previous work, NPTs explicitly learn interactions between datapoints and can be applied to general supervised machine learning tasks. We refer to §8.2 for an overview of these and other related approaches.

We next discuss the specifics of our model (§8), before moving on to related work (§8.2), empirical results (§9.4), and finally, limitations, future work, and conclusions (§8.4).



(a) Input Data (b) Notation (c) Parametric Model (d) **NPT**

Figure 8.1: NPTs learn direct interactions between datapoints. (a) Input data: predict masked target entry [?] for datapoint X_i. (b) Notation from §8. (c) Parametric models predict only from the features of the given input. (d) NPTs predict by modeling relationships between all points in the dataset.

8.1 Non-Parametric Transformers

Non-Parametric Transformers (NPTs) explicitly *learn* relationships between datapoints to improve predictions. To accomplish this, they rely on three main ingredients: **(1)** We provide the model with the **entire dataset – all datapoints – as input**. At test time, both training and test data are input to the model; during training, the model learns to predict targets from the training data only. We approximate this where necessary for large data (§8.1.6). **(2)** We use **self-attention between datapoints** to explicitly model relationships between them. For example, at test time, the attention mechanism models relationships amongst training points, amongst test points, and between the two. **(3)** NPT's training objective is to reconstruct a corrupted version of the input dataset. Similar to BERT (Devlin et al., 2018), we apply **stochastic masking** to both features and targets and minimize a loss on NPT's predictions at entries masked out in the input. Next, we introduce the three components in detail.

8.1.1 Datasets as Inputs

NPTs take as input the entire dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. The datapoints are stacked as the rows of this matrix $\{\boldsymbol{X}_{i,:} \in \mathbb{R}^d \mid i \in 1 \dots n\}$, and we refer to the columns as attributes $\{\boldsymbol{X}_{:,j} \in \mathbb{R}^n \mid j \in 1 \dots d\}$. Each attribute is assumed to share a semantic meaning among all datapoints. In single-target classification and regression, we assume that the targets (labels) are the final attribute $\boldsymbol{X}_{:,d}$, and the other attributes $\{\boldsymbol{X}_{:,j} \mid j \neq d\}$ are input features, e.g., the pixels of an image. Each $\boldsymbol{X}_{i,j}$ is an entry or value. In addition to tabular data, many modalities such as images, graphs, or timeseries can be reshaped to fit this format. Note that this is a departure from common notation for supervised learning as introduced in §9.1, as the input $\boldsymbol{X}$ now includes both features and targets (collectively, attributes).

In masked language modeling (Devlin et al., 2018), mask tokens denote which words in a sentence should be concealed and where model predictions will have a loss backpropagated at training time. Analogously, we use a binary matrix $\boldsymbol{M} \in \mathbb{R}^{n \times d}$ to specify which entries are *masked* in the input $\boldsymbol{X}$. This matrix is also passed to NPT

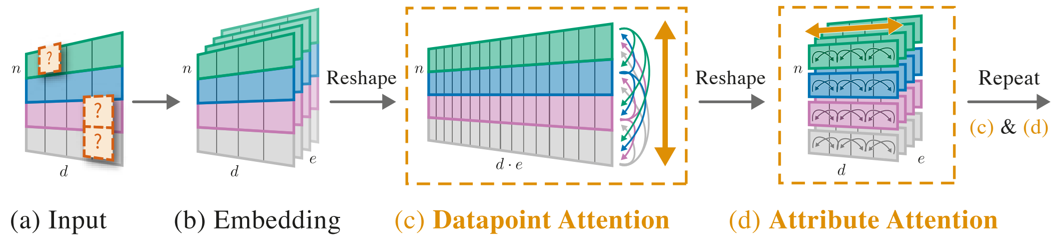


Figure 8.2: Overview of the Non-Parametric Transformer. (a) The input dataset and mask matrix are stacked and (b) linearly embedded for all datapoints independently. NPT then applies (c) Attention Between Datapoints (ABD, §8.1.4) across all n samples of hidden dimension $h = d \cdot e$. (d) Attention Between Attributes (ABA, §8.1.5) then attends between the attributes for each datapoint independently. We repeat steps (c) and (d) and obtain a final prediction from a separate linear projection (not shown).

as input. The task is to predict the masked values $X^M = \{X_{i,j} \mid M_{i,j} = 1\}$ from the observed values $X^O = \{X_{i,j} \mid M_{i,j} = 0\}$, i.e., to predict $p(X^M \mid X^O)$.

In summary, NPT takes as input the entire dataset and masking matrix (X, M), and makes predictions $\hat{X} \in \mathbb{R}^{n \times d}$ for values masked at input. This general setup accommodates many machine learning settings simply by adjusting the placement of the binary masks in M. We focus on single-target classification and regression – corresponding to a masking matrix M with 1s at all entries of the label column $X_{:,d}$. Next, we describe the NPT architecture.

8.1.2 NPT Architecture

An overview of the Non-Parametric Transformer (NPT) is depicted in Fig. 8.2. NPT receives the dataset and masking matrix (X, M) as input (Fig. 8.2a). We stack these and apply an identical linear embedding to each of n datapoints, obtaining an input representation $H^{(0)} \in \mathbb{R}^{n \times d \times e}$ (Fig. 8.2b). Next, we apply a sequence of multi-head self-attention layers (Vaswani et al., 2017a; Devlin et al., 2018; Bahdanau et al., 2015). Crucially, we alternatingly apply attention between *datapoints*, and attention between *attributes* of individual datapoints (Figs. 8.2c-d).

These operations allow our model to learn both relationships between datapoints as well as transformations of individual datapoints. Finally, an output embedding gives the prediction $\hat{X} \in \mathbb{R}^{n \times d}$, which now has predicted values at entries that were masked at input.

Property 1. *NPTs are equivariant to a permutation of the datapoints.*

In other words, if the set of input datapoints are shuffled, NPTs produce the same predictions but shuffled in an analogous manner. This explicitly encodes the assumption that the learned relations between datapoints should not depend on their ordering. At a high level, permutation-equivariance (PE) holds because all components of NPT are PE, and the composition of PE functions is PE. We now briefly recap multi-head self-attention, an important operation in the NPT architecture.

8.1.3 Multi-Head Self-Attention

Multi-head self-attention (MHSA) is a powerful mechanism for learning complex interactions between elements in an input sequence. Popularized in natural language processing (Vaswani et al., 2017a; Devlin et al., 2018; Bahdanau et al., 2015), MHSA-based models have since been successfully applied to many areas of machine learning (cf. §8.2).

Dot-product attention computes attention weights by comparing queries $\{Q_i \in \mathbb{R}^{1\times h_k} \mid i \in 1\dots n\}$ with keys $\{K_i \in \mathbb{R}^{1\times h_k} \mid i \in 1\dots m\}$, ultimately updating the representation of the queries by aggregating over values $\{V_i \in \mathbb{R}^{1\times h_v} \mid i \in 1\dots m\}$ via the attention weights. We stack the queries, keys, and values into matrices $Q \in \mathbb{R}^{n\times h_k}$, $K \in \mathbb{R}^{m\times h_k}$, and $V \in \mathbb{R}^{m\times h_v}$ and, as is commonly done for convenience, assume $h_k = h_v = h$. Then, we compute dot-product attention as

$$\text{Att}(Q, K, V) = \text{softmax}(QK^T/\sqrt{h})\, V. \tag{8.1}$$

Multi-head dot-product attention concatenates a series of k independent *attention heads*

$$\text{MHAtt}(Q, K, V) = \underset{\text{axis}=h}{\text{concat}}(O_1, \dots, O_k)\, W^O, \text{ where } O_j = \text{Att}(QW_j^Q, KW_j^K, VW_j^V). \tag{8.2}$$

We learn embedding matrices $W_j^Q, W_j^K, W_j^V \in \mathbb{R}^{h\times h/k}, j \in \{1, \dots, k\}$ for each head j, and $W^O \in \mathbb{R}^{h\times h}$ mixes outputs from different heads. Here, we focus on multi-head *self*-attention, $\text{MHSelfAtt}(H) = \text{MHAtt}(Q = H, K = H, V = H)$, which uses the *same* inputs for queries, keys, and values. Following Transformer best practices to improve performance Vaswani et al. (2017a); Devlin et al. (2018); Lee et al. (2019a); Chen et al. (2018); Narang et al. (2021), we first add a residual branch and apply Layer Normalization (LN) (Ba et al., 2016) followed by $\text{MHSelfAtt}(\cdot)$,

$$\text{Res}(H) = HW^{\text{res}} + \text{MHSelfAtt}(\text{LN}(H)), \tag{8.3}$$

with learnable weight matrix $W^{\text{res}} \in \mathbb{R}^{h\times h}$. Then, we add another residual branch with LN and a row-wise feed-forward network (rFF), finally giving the full multi-head self-attention layer as

$$\text{MHSA}(H) = \text{Res}(H) + \text{rFF}(\text{LN}(\text{Res}(H)) \in \mathbb{R}^{n\times h}. \tag{8.4}$$

8.1.4 Attention Between Datapoints (ABD)

The **Attention Between Datapoints (ABD)** layer is a key operation for NPT. It explicitly transforms data by reasoning about pairwise relationships between all datapoints, see Fig. 8.2c. As input to ABD, we flatten the output of the previous layer $\boldsymbol{H}^{(\ell)}$ from $\mathbb{R}^{n \times d \times e}$ to $\mathbb{R}^{n \times h}$ with $h = d \cdot e$. Then, we perform multi-head self-attention between the datapoints $\{\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{1 \times h} \mid i \in 1 \ldots n\}$ as

$$\mathrm{ABD}(\boldsymbol{H}^{(\ell)}) = \mathrm{MHSA}(\boldsymbol{H}^{(\ell)}) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n \times h}. \tag{8.5}$$

At the first ABD layer, we input $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$, the linearly embedded input data. After applying ABD, we reshape the output again, from $\mathbb{R}^{n \times h}$ to $\mathbb{R}^{n \times d \times e}$.

Note that this is distinct from how $\mathrm{MHSA}(\cdot)$ is usually applied in the literature, as we compute attention between *different datapoints* and not between the *features of a single datapoint* (Vaswani et al., 2017a; Devlin et al., 2018; Dosovitskiy et al., 2021a; Jaegle et al., 2021). For example, in natural language processing, attention is usually applied between the tokens (features) of a sentence (datapoint) but not between different sentences. For example, NPT could learn to attend between two datapoints with indices i and i' by embedding $\boldsymbol{Q}_i$ and $\boldsymbol{K}_{i'}$ in close proximity. Following (8.1), datapoint i will then attend more closely to i' because $\boldsymbol{Q}_i \boldsymbol{K}_{i'}^T$ will be large. By stacking many ABD layers, NPT can learn higher-order interactions between datapoints (Vaswani et al., 2017a; Devlin et al., 2018).

8.1.5 Attention Between Attributes (ABA)

We now introduce **Attention Between Attributes (ABA)**, which is always performed following ABD. ABA layers can help the model learn better per-datapoint representations for the between-datapoint interactions, see Fig. 8.2d. In ABA, we apply MHSA independently to each row (corresponding to a single datapoint) in the input $\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{d \times e}$, $i \in \{1, \ldots, n\}$, giving

$$\mathrm{ABA}(\boldsymbol{H}^{(\ell)}) = \underset{\text{axis}=n}{\text{stack}} \left(\mathrm{MHSA}(\boldsymbol{H}_1^{(\ell)}), \ldots, \mathrm{MHSA}(\boldsymbol{H}_n^{(\ell)})\right) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n \times d \times e}. \tag{8.6}$$

Just like in standard Transformers (Vaswani et al., 2017a; Devlin et al., 2018; Dosovitskiy et al., 2021a; Jaegle et al., 2021), ABA is used to transform attribute representations of single datapoints independently. We batch over the n dimension to compute ABA efficiently. By alternating between attention over datapoints (ABD) and attributes (ABA), NPTs can model both complex dependencies between points as well as learn suitable transformations of datapoints individually. This method is also significantly less computationally practical than doing attention between datapoints and attributes jointly. Next, we describe the use of masking mechanisms during NPT training and evaluation.

8.1.6 Masking and Optimization

Masking. Much like in masked language modeling (Devlin et al., 2018), we use masks to indicate which values NPT is expected to predict, and to prevent the model from accessing ground truth values. Recall that NPT needs to predict $p(X^M \mid X^O)$, with masked values $X^M = \{X_{i,j} \mid M_{i,j} = 1\}$ and observed values $X^O = \{X_{i,j} \mid M_{i,j} = 0\}$. Masked values can be either features or targets. Canonically, masked language modeling is used to perform self-supervised learning on a sequence of tokens in a sentence (Devlin et al., 2018). We use such *stochastic feature masking* to mask a feature value $X_{i,j}, j \neq d$ with probability p_{feature} during training. *Stochastic target masking* is done in the same manner on the targets of the training set $X_{:,d}$ with p_{target}. Note that we take great care to avoid test set leakage, and *never* reveal targets of the test set to NPT.

NPT Objective. During training, we compute the negative log-likelihood loss at training targets $\mathcal{L}^{\text{Targets}}$ as well as the auxiliary loss from masked-out features $\mathcal{L}^{\text{Features}}$. We write the NPT training objective as $\mathcal{L}^{\text{NPT}} = (1 - \lambda)\mathcal{L}^{\text{Targets}} + \lambda\mathcal{L}^{\text{Features}}$, where λ is a hyperparameter. At test time, we only mask and compute a loss over the targets of test points.

This objective has a few notable elements. Feature masking requires NPTs to make predictions over all attributes, encouraging the models to learn a representation of the entire dataset. This increases the difficulty of the task and adds more supervision, which we find tends to have a beneficial regularizing effect. Interestingly, stochastic target masking means that many training targets are *unmasked* to the model at training time. This allows NPTs to learn to predict, at each epoch, the masked targets of certain training datapoints using the *targets of other training datapoints* in addition to all training data features.[1] NPTs no longer have to memorize a mapping between training inputs and outputs in their parameters θ, and can instead use their representational capacity to learn functions using other *training features and targets as input.* For example, NPTs could learn to assign test datapoints to clusters of training datapoints, and predict on those points using interpolation of the training targets in their respective cluster. We explore the ability of NPTs to solve such complex reasoning tasks in §8.3.2.

Handling Large Datasets. Avoiding the poor $\mathcal{O}(n^2)$ time and space complexity of naïve self-attention, we resort to approximations once the data grows too large. For example, we reach 24 GB of GPU memory for standard NPT model sizes at about 8000 datapoints. We find that processing the data in random subsets for model training and prediction, i.e., *minibatching*, is a simple and effective solution. We construct minibatches such that, at test time, training and test data are both present in the same batch, to allow NPTs to attend to training datapoints. In §8.3.3, we show that

[1]A potential concern is that the model will memorize training targets and fail to generalize. In practice, we do not observe generalization issues, likely because (i) a loss is never backpropagated on an unmasked value, and (ii) BERT-style masking (Devlin et al., 2018) uses token randomization to prevent memorization.

NPTs make use of attention between datapoints with minibatching enabled. See §8.4 for further discussion and ideas for future work.

8.2 Related Work

Deep Non-Parametric Models. Deep Gaussian Processes (Damianou & Lawrence, 2013) and Deep Kernel Learning (DKL) (Wilson et al., 2016) extend ideas from Gaussian Processes (Rasmussen, 2003) to representation learning. Deep GPs stack standard GPs with the aim to learn more expressive relationships between input points, sharing motivation with NPTs. However, unlike NPTs, deep GPs are difficult to work with in practice, requiring complex approximate inference schemes (Dai et al., 2016; Bui et al., 2016; Salimbeni & Deisenroth, 2017). DKL applies a neural network to each datapoint *independently* before passing points on to a standard Gaussian Process, making predictions based directly on similarity in embedding space instead of *learning* the interactions themselves.

Neural Processes. Similar to GPs, Neural Processes (NPs) (Garnelo et al., 2018b,a) define a distribution over functions. They use a latent variable model parametrized by neural networks, fulfilling specific architectural constraints to approximately preserve consistency of finite-dimensional marginals. Attentive Neural Processes (ANPs) (Kim et al., 2019) extend Neural Processes to allow for direct attention between a context set and targets. However, as the authors themselves stress, "NPs and GPs have different training regimes" (Kim et al., 2019). While a GP can be trained on a single dataset, *(A)NPs require multiple realizations of the dataset.* The authors further note that *"a direct comparison between the two is usually not plausible"* (Kim et al., 2019), which is why we cannot compare (A)NPs to NPT on our standard tasks.

Attention. NPTs are part of a line of recent work that explores the use of Transformer-based architectures outside of natural language processing, e.g., Transformers in computer vision Parmar et al. (2018); Dosovitskiy et al. (2021a); Jaegle et al. (2021) or architectures exploiting desirable invariances or equivariances Lee et al. (2019a); Locatello et al. (2020); Fuchs et al. (2020); Hutchinson et al. (2020). Like NPTs, Set Transformer (Lee et al., 2019a) attends to a set of input points. However, unlike NPTs, Set Transformer relies on the existence of multiple independent sets for training and makes only a single prediction for each set. Like NPTs, Axial Transformers (Ho et al., 2019) and MSA Transformers (Rao et al., 2021) attend to multiple dimensions of matrix-shaped input. However, Axial Transformers process single images as input, i.e., no attention across datapoints is performed. MSA Transformers use attention within individual protein sequences and across an aligned protein family for contact prediction, but do not consider a more general setting. Recent works have improved neural network performance on tabular data using attention. AutoInt (Song et al., 2019) is a direct application of multi-head attention to tabular data, and TabNet (Arik & Pfister, 2019) sequentially attends to sparse subsets of the features inspired by tree-based models. Both approaches do not reason about interactions between datapoints, a key contribution that we introduce with NPT in this work.

Few-Shot Learning, Meta-Learning, and Prompting. In §8.3.2, we apply NPTs to tasks that require learning of relational structure between datapoints on training data to achieve good generalization performance on novel test inputs. This setup shares motivations with meta-learning (Biggs, 1985; Bengio et al., 1991; Lake et al., 2015; Finn et al., 2017), in which a model is pre-trained on a variety of tasks, such that it can then learn new tasks using only a small number of additional training points from the new task. However, we consider evaluation without any additional gradient updates, unlike recent meta-learning methods (Finn et al., 2017; Yoon et al., 2018) which are therefore inapplicable to this setting. Recent works on few-shot learning with text prompting (Radford et al., 2019b; Brown et al., 2020b) provide a trained Transformer-based language model with a few examples of a novel relationship in a prompt at prediction time, and observe strong generalization on the task. Similarly, we consider attention between a "context" of datapoints. While ground-truth input-output pairs are provided for prompting, we consider settings in which no ground-truth is given at prediction time, but the model can solve the task if it has learned the underlying relational structure. Another area of research that connects to NPTs is Retrieval Augmented Generation (Lewis et al., 2020) where a language model is given access to a knowledge base and is able to make reference to entries as it generates. NPTs could be treated in a similar fashion, where, at inference time, the system would retrieval examples from the dataset that are similar (according to some metric) to the inference-time input being considered. We don't explore this further in this work, we expect that such a technique could quite easily lead to improved model performance with no additional training.

Due to the unique properties of NPTs, we believe that there are many other exciting connections to be drawn. We discuss a selection of possible areas of application including semi-supervised learning, graph neural networks, and relational learning, and leave other areas such as prediction on missing data, semi-supervised learning, and continual learning to future research. In this initial study, we instead concentrate on questions at the core of NPTs.

8.3 Experiments

We seek to answer the following set of questions in our evaluation[2] of NPTs: (**Q1**) How do NPTs perform on standard benchmarks for supervised machine learning? (**Q2**) Can NPTs successfully model interactions between datapoints in idealized settings? (**Q3**) Do NPTs actually learn to rely on interactions between datapoints for prediction on real-world datasets? (**Q4**) If so, what is the nature of these interactions, e.g., which other datapoints are relevant for prediction?

[2]We release code for NPTs at github.com/OATML/Non-Parametric-Transformers.

Table 8.1: Average rank order of various methods ($\pm$ standard error) on UCI benchmarks, across binary classification, multi-class classification, and regression tasks. We determine rank using the test area under the receiver operating characteristic (AUROC) curve on binary classification (4 of 10 datasets), accuracy on multi-class classification (2 of 10), and root mean squared error (RMSE) on regression (4 of 10), and sort methods by ascending rank for each metric.

Method	AUROC
NPT	**2.50(87)**
CatBoost	2.75(85)
LightGBM	3.50(155)
XGBoost	4.75(125)
Gradient Boosting	5.00(71)
MLP	5.75(149)
Random Forest	6.00(71)
TabNet	6.50(132)
k-NN	8.25(48)

Method	Accuracy
NPT	**2.50(50)**
XGBoost	**2.50(150)**
MLP	3.00(200)
CatBoost	3.50(50)
Gradient Boosting	3.50(150)
Random Forest	6.50(50)
TabNet	7.50(50)
LightGBM	7.50(150)
k-NN	8.50(50)

Method	RMSE
CatBoost	**3.00(91)**
XGBoost	3.25(63)
NPT	3.25(131)
Gradient Boosting	4.00(108)
Random Forest	4.50(87)
MLP	5.00(122)
LightGBM	6.50(155)
TabNet	6.75(95)
k-NN	8.75(25)

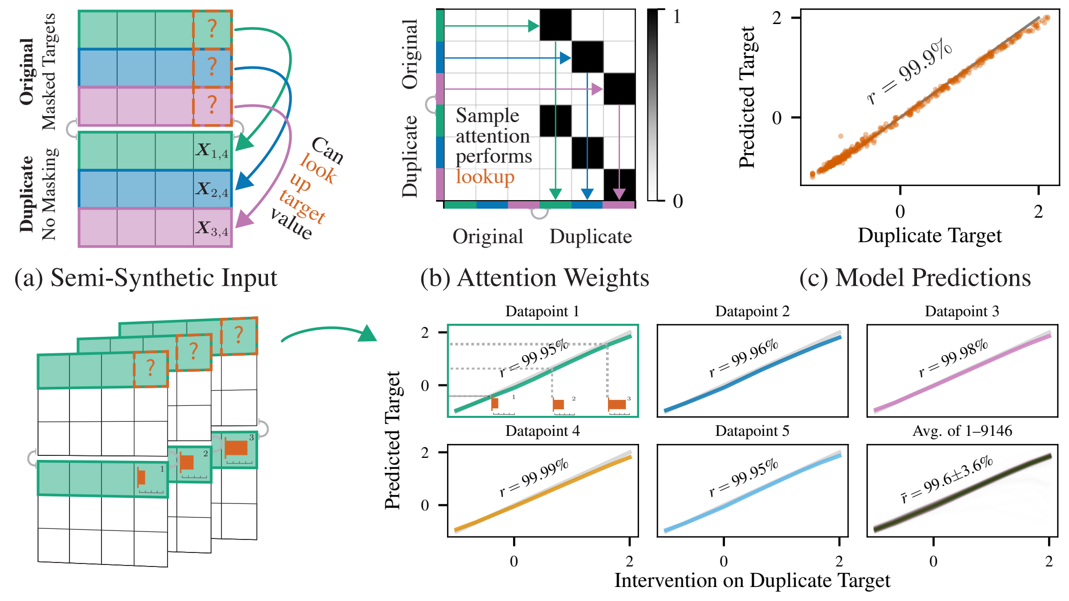
8.3.1 NPTs Perform Competitively on Established Benchmarks

To answer **(Q1)**, we evaluate NPTs on tabular data from the UCI Repository (Dua & Graff, 2017) as well as the CIFAR-10 (Krizhevsky et al., 2009) and MNIST (LeCun et al., 2010) image classification datasets. Tabular data is ubiquitous in real-world machine learning Chui et al. (2018) but notoriously challenging for general purpose deep neural networks, which consistently underperform boosting models (Schapire, 1990) and are rarely used in practice.[3]

Tabular Datasets, Setup, and Baselines. We evaluate NPTs over 10 datasets varying across the number of datapoints, number of features, composition (categorical or continuous) of features, and task. 4 of the 10 are binary classification, 2 are multi-class classification, and 4 are regression. We compare NPT against a wide set of standard or state-of-the-art baselines: Random Forests (Breiman, 2001), Gradient Boosting Trees (Friedman, 2001), XGBoost (Chen & Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), LightGBM (Ke et al., 2017), MLPs, k-NN (Fix, 1985; Altman, 1992), and TabNet (Arik & Pfister, 2019). We tune the parameters of all models on validation sets and use 10-fold cross-validation whenever computationally feasible. Note that while we perform an extensive grid search for the baselines, we only search over a small set of configurations for NPTs.

Tabular Data Results. We report the average rank order for NPT and various tree-based and deep learning baselines in Table 8.1. NPT achieves the highest average ranking on binary and multi-class classification tasks, outperforming CatBoost and XGBoost, two popular state-of-the-art boosting methods designed specifically

[3]We conduct an informal survey of all Kaggle (Inc., 2021) competitions using tabular data completed in 2020 with a public leaderboard. In 11 out of a total of 13 cases, the winning entries relied on some form of boosting.



(a) Semi-Synthetic Input (b) Attention Weights (c) Model Predictions

(d) Interventions on Duplicates (e) Model Responses to Interventions

Figure 8.3: Demonstrating NPT's ability to predict from Attention Between Datapoints (ABD). (a) We append to the original data with masked targets [?] a copy of the same data with all masked values revealed, such that perfect prediction via lookup is possible. (b) Attention weights indicate that the ideal lookup behavior is learned by NPT. Shown are actual values learned by NPT at head 0 and depth 4 for the first 3 datapoints. (c) NPT predictions closely match the ideal values. (d) Additionally, we intervene on the values of individual targets, (e) finding that NPT predictions adjust accordingly.

for tabular data. On regression tasks, NPT ties in average rank with XGBoost, and is outperformed only by CatBoost. In addition to its strong rank-wise performance, NPT achieves best performance on 4 of the 10 benchmark datasets – more than any other method. We find that these are remarkable results for a general purpose model that does not include tabular-specific design, supporting our hypothesis that attention between datapoints is a useful architectural inductive bias for prediction.

Image Data Results. NPT achieves 68.2% accuracy on CIFAR-10 and 98.3% accuracy on MNIST. Similar to previous work on Transformers for computer vision, we would expect (pre)-training on millions of images to significantly boost NPT's performance (Deng et al., 2009; Jaegle et al., 2021; Touvron et al., 2020; Sun et al., 2017; Ridnik et al., 2021). We perform no pre-training, and therefore a direct comparison of our results to this line of work is inappropriate. Crucially, we show in §8.3.3 that NPTs learn to make use of interactions between images, indicating that attention between datapoints is valuable for image classification.

8.3.2 NPTs Can Learn to Predict Using Attention Between Datapoints

To determine if NPTs can successfully learn to exploit interactions between datapoints **(Q2)**, we introduce a task with strong input correlations for which we know ground-truth interactions. Concretely, we take the UCI Protein regression dataset (cf. §8.3.1), to construct the following semi-synthetic task: for each batch, we input the original data with masked target values as well as a *copy* of the original data where all target values have been revealed, i.e., no masking is applied (Fig. 8.3a). NPTs can use attention between datapoints to achieve arbitrarily good performance by *learning* to look up the target values in the matching duplicate row. At test time, we input novel semi-synthetic test data to ensure that NPT has learned the correct relational mechanism and not just memorized target values.

NPTs successfully learn to perform this lookup between original and duplicate datapoints. The ABD attention weights, visualized for the first three datapoints in Fig. 8.3b, clearly show the model correctly attending to the duplicates. As a result, NPT predictions are Pearson-correlated with the duplicate targets at $r = 99.9\%$ (Fig. 8.3c). This equals an RMSE of only 0.44, about a magnitude lower than the error on the original Protein dataset. We conclude that NPTs learn to predict by looking up the target values from matching points.

Purely parametric models cannot exploit information from other datapoints, limiting their performance. For example, MLPs achieve an RMSE of 3.62 on this task. Non-parametric approaches also cannot solve this task in its original form, because unlike NPTs they must be told which datapoints are the originals (training data) and which the duplicates (test data) as well as which columns contain features and which target values. We demonstrate that even when we make these concessions, we can easily adapt the task such that both k-Nearest Neighbors and Deep Kernel Learning fail to solve it. In fact, we are not aware of any other model that can solve the adapted task.

Additionally, we perform an *interventional* experiment to investigate the extent to which NPTs have actually learned the causal mechanism underlying the lookup task. As illustrated in Fig. 8.3d, we now intervene on individual duplicate datapoints at test time by varying their target value across a wide range. We stress that we perform these experiments without retraining the model, using exactly the same NPT from Figs. 8.3a-c. The model is now confronted with target values associated with features that are highly unlikely under the training data. This label distribution shift (Garg et al., 2020) is a challenging setting for neural networks. However, NPT predictions follow the intervened target values with near-perfect correlation, Fig. 8.3e, continuing to predict by correctly looking up targets.

We now confidently conclude that NPTs robustly learn the causal data-generating mechanism underlying the semi-synthetic dataset. This requires NPTs to *learn* a non-trivial sequence of compuational steps. They must learn to match rows based on

Table 8.2: Drop in NPT performance after destroying information from other datapoints. Shown are changes in test set performance, where negative values indicate worse performance after corruption.

Δ *Accuracy*	CIFAR-10	Poker	Income	Higgs	MNIST	Forest	Kick	Breast Cancer
	−5.1	−1.1	−1.1	−0.5	−0.4	−0.1	−0.1	0.0
$^{\Delta RMSE}/_{RMSE}$ (%)	Yacht	Protein	Boston	Concrete				
	−52%	−21%	−20%	−7%				

similarity of relevant features; to look up the target value of the duplicated datapoint; and, to copy that value into the target of the masked datapoint.

8.3.3 NPTs Learn to Use Attention Between Datapoints on Real Data

We next consider **(Q3)**: do NPTs actually learn to use attention between datapoints for prediction on real data? We design a test that allows us to quantify the extent to which NPT predictions depend on relationships between datapoints at test time. Concretely, for each target value in the input we randomize the data for all *other* datapoints by independently shuffling each of their attributes across the rows. We then evaluate the loss on the prediction at the target entry and repeat this procedure for all test datapoints. This completely corrupts the information from all datapoints except the one for which we evaluate. Hence, a model that relies meaningfully on attention between datapoints will show deteriorating performance.

We report the resulting change in performance after corruption in Table 8.2 for all datasets from §8.3.1. We find that for most datasets, the corruption of other rows at test time significantly decreases the performance of the trained NPT models. This indicates that the NPTs have successfully learned to make predictions supported by attention between datapoints. For some datasets, the corruption experiment deteriorates performance completely. For example, for the Protein regression dataset NPT achieves state-of-the-art performance, but corrupting the input leads to NPT performing worse than all of the baselines considered in §8.3.1. We note that minor differences in performance are often still significant, as differences between competing models in §8.3.1 are often likewise small.

Interestingly, on certain datasets such as Forest Cover, Kick, and Breast Cancer, corrupted inputs do not significantly affect performance. It appears that when NPTs do not find it advantageous to rely on attention between datapoints during training, they can learn to completely ignore other inputs, essentially collapsing into a standard parametric model. This supports our earlier claims that NPTs can learn end-to-end from data the extent to which they rely on other datapoints for prediction. We think this is extremely interesting behavior and are unaware of prior work reporting similar results. However, we stress that these results reflect inductive biases of the NPT architecture and do not lend themselves to general statements about the performance of parametric versus non-parametric models.

8.3.4 NPTs Rely on Similar Datapoints for Predictions on Real Data

So far, we have presented convincing evidence that NPTs (sometimes strongly) depend on attention between datapoints. However, we do not know what kind of interactions are learned in practice on real data (**Q4**). As an initial step towards understanding this, we now present two experiments investigating *to which* other datapoints NPT attends.

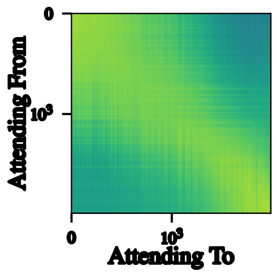


Fig. 4: Attention weights.

Qualitative Evidence. Figure 8.4 shows an attention map for attention between datapoints (ABD) of NPT on a batch of the Protein regression dataset. We sort the input data with respect to their feature space distance such that similar datapoints are now close to each other. The diagonal pattern in Fig. 8.4 indicates that NPT attends more strongly to datapoints that are similar in feature space. ?? discusses this further and gives additional attention maps.

Quantitative Evidence. Seeking a quantitative measure for this hypothesis, the *data deletion* experiment repeats the following procedure for all test set points: iteratively delete other datapoints from the input if they do not significantly affect the prediction. We stop if less than 2% of the original datapoints remain, or if the total change in prediction for the target (relative to the original prediction with all data) exceeds 10%. We investigate the average feature space distances between the test point and the *kept* datapoints, as well as the distances between the test point and the *deleted* datapoints. We find that kept datapoints have a significantly lower average feature space distance to the test point than those deleted. This indicates that two datapoints i, i' that are similar in feature space, such that $\sum_{j<d}(X_{i,j} - X_{i',j})^2$ is low, have a larger effect on the predictions of one another. A Wilcoxon signed-rank test is significant at $p \approx 8.77 \cdot 10^{-130}$. We give full details on this in ??.

Both experiments support the hypothesis that NPTs rely on similar datapoints for prediction in real data settings. One possible explanation is that similar datapoints might have different realizations of observation noise which NPTs could learn to average out. Altogether, we conclude that NPTs can and do learn representations which rely on interactions between datapoints for prediction.

8.4 Limitations, Future Work, and Conclusions

Limitations. NPTs share scaling limitations with all naïvely non-parametric approaches (Rasmussen, 2003) and GNNs (Kipf & Welling, 2017). While we have seen success with random minibatching (§8.1.6), future work might consider applying principled attention approximations, such as learning representative input points (Lee et al., 2019a), kernelization (Katharopoulos et al., 2020; Choromanski et al., 2021), or other sparsity-inducing methods (Tay et al., 2020; Child et al., 2019; Beltagy et al., 2020), to improve the scalability of NPTs.

Future Work. We believe that the unique predictive mechanism of NPTs makes them an interesting object of study for other tasks including continual learning, multi-task learning, few-shot generalization, and domain adaptation. For example, when predicting under distribution shift, general relations between datapoints and attributes may remain valid and allow NPTs to accommodate such scenarios better. Additionally, future work could explore the connections to stochastic processes, e.g., by extending NPTs to be approximately consistent, similar to Neural Processes (Garnelo et al., 2018b,a; Kim et al., 2019).

Conclusions. We have introduced Non-Parametric Transformers (NPTs), a novel deep learning architecture that takes the entire dataset as input and uses self-attention to model complex relationships *between* datapoints. NPTs challenge and naturally extend parametric modeling as the dominant paradigm of deep learning. They have the additional flexibility to learn to predict by directly attending to other datapoints. Notably, NPTs learn this end-to-end from the data at hand. Empirically, NPTs achieve highly competitive performance on a variety of benchmarks, and additional experiments demonstrate their ability to solve complex reasoning tasks over datapoints. Further, we show that on real data, NPTs learn to rely on attention between datapoints for prediction. We believe that the characteristics of NPTs will make them an exciting object of further study.

Chapter 9

Prioritized training on points that are learnable, worth learning, and not yet learned

As dataset scale has increased, practitioners have had to turn towards ”less clean” sources of data such as scraping the web. This results in datasets that contain significant quantities of noise and repetition, which pose a hindrance to model quality. In this work, we propose a model-based data selection method that can substantially improve model quality and reduce the amount of training necessary to reach good performance.

The following is an excerpt from Mindermann et al. (2022) and was published at the International Conference on Machine Learning. This is work done in collaboration with Sören Mindermann, Jan Brauner, Muhammed Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Holtgen, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. In this project I spurred the initial project idea in conversation with Sören and Adrien about improving data efficiency, helped with experiment design, and supervised the project alongside Yarin Gal.

9.1 Introduction

State-of-the-art models such as GPT-3 (Brown et al., 2020b), CLIP (Radford et al., 2021), and ViT (Dosovitskiy et al., 2021b) achieve remarkable results by training on vast amounts of web-scraped data. But despite intense parallelization, training such a model takes weeks or months (Radford et al., 2021; Chowdhery et al., 2022). Even practitioners who work with smaller models face slow development cycles, due to numerous iterations of algorithm design and hyperparameter selection. As a result,

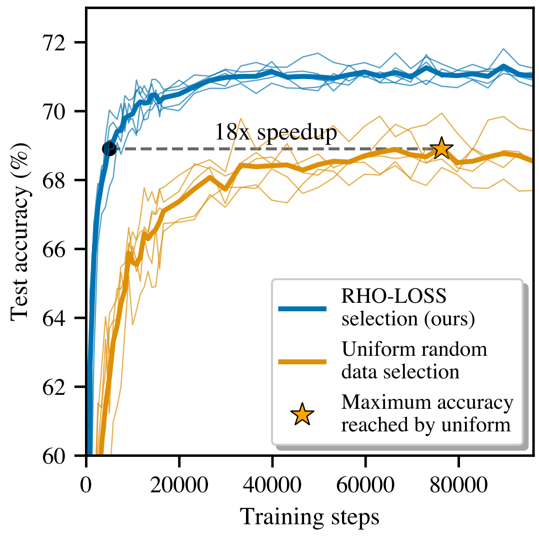


Figure 9.1: Speedup on large-scale classification of web-scraped data (Clothing-1M). RHO-LOSS trains all architectures with fewer gradient steps than standard uniform data selection (i.e. shuffling), helping reduce training time. Thin lines: ResNet-50, MobileNet v2, DenseNet121, Inception v3, GoogleNet, mean across seeds. Bold lines: mean across all architectures.

the total time required for training is a core constraint in the development of such deep learning models.

If it further sped up training, practitioners with sufficient resources would use much larger batches and distribute them across many more machines (Anil et al., 2018). However, this has rapidly diminishing returns (LeCun et al., 2012), to a point where adding machines does not reduce training time (McCandlish et al., 2018; Anil et al., 2018)—see e.g. GPT-3 and PaLM (Chowdhery et al., 2022).

Additional machines can, however, still speed up training by filtering out less useful samples (Alain et al., 2015). Many web-scraped samples are *noisy*, i.e. their label is incorrect or inherently ambiguous. For example, the text associated with a web-scraped image is rarely an accurate description of the image. Other samples are learned quickly and are then *redundant*. Redundant samples are commonly part of object classes that are over-represented in web-scraped data (Tian et al., 2021) and they can often be left out without losing performance. Given that web-scraped data is plentiful—often enough to finish training in a single epoch (Komatsuzaki, 2019; Brown et al., 2020b)—one can afford to skip less useful points.

However, there is no consensus on which datapoints are the most useful. Some works, including curriculum learning, suggest prioritizing *easy* points with low label noise before training on all points equally (Bengio et al., 2009). While this approach may improve convergence and generalization, it lacks a mechanism to skip points that are already learned (redundant). Other works instead suggest training on points that are *hard* for the model, thereby avoiding redundant points, whose loss cannot be further reduced. Online batch selection methods (Loshchilov & Hutter, 2015; Katharopoulos & Fleuret, 2018; Jiang et al., 2019) do so by selecting points with high loss or high gradient norm.

We show two failure modes of prioritising hard examples. Firstly, in real-world noisy datasets, high loss examples may be mislabelled or ambiguous. Indeed, in controlled experiments, points selected by high loss or gradient norm are overwhelmingly those with noise-corrupted labels. Our results show that this failure mode degrades performance severely. More subtly, we show that some samples are hard because they are outliers—points with unusual features that are less likely to appear at test time. For the aim of reducing test loss, such points are less *worth learning.*

To overcome these limitations, we introduce *reducible holdout loss selection* (RHO-LOSS). We propose a selection function grounded in probabilistic modelling that quantifies by how much each point would reduce the loss on unseen data if we were to train on it, *without actually training on it.* We show that optimal points for reducing holdout loss are non-noisy, non-redundant, and task-relevant. To approximate optimal selection, we derive an efficient and easy-to-implement selection function: the reducible holdout loss.

We explore RHO-LOSS in extensive experiments on 7 datasets. We evaluate the reduction in required training steps compared to uniform sampling and state-of-the-art batch selection methods. Our evaluation includes Clothing-1M, the main large benchmark with noisy, web-scraped labels, matching our main application. RHO-LOSS reaches target accuracy in 18x fewer steps than uniform selection and achieves 2% higher final accuracy (Fig. 9.1). Further, RHO-LOSS consistently outperforms prior art and speeds up training across datasets, modalities, architectures, and hyper-parameter choices. Explaining this, we show that methods selecting "hard" points prioritize noisy and less relevant examples. In contrast, RHO-LOSS chooses low-noise, task-relevant, non-redundant points—points that are learnable, worth learning, and not yet learned.

9.2 Background: Online Batch Selection

Consider a model $\mathrm{p}(y \mid x; \theta)$ with parameters θ training on data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ using stochastic gradient descent (SGD). At each training step t, we load a batch b_t of size n_b from $\mathcal{D}$. In online batch selection (Loshchilov & Hutter, 2015), we uniformly pre-sample a larger batch B_t of size $n_B > n_b$. Then, we construct a smaller batch b_t that consists of the top-ranking n_b points in B_t ranked by a label-aware selection function $S(x_i, y_i)$. We perform a gradient step to minimize a mini-batch loss $L(y_i, \mathrm{p}(y_i \mid x_i; \theta))$ summed over $i \in b_t$. The next large batch B_{t+1} is then pre-sampled from $\mathcal{D}$ without replacement of previously sampled points (points are replaced at the start of the next epoch).

9.3 Reducible Holdout Loss Selection

Previous online batch selection methods, such as loss or gradient norm selection, aim to select points that, if we were to train on them, would minimize the *training set* loss. (Loshchilov & Hutter, 2015; Katharopoulos & Fleuret, 2018; Kawaguchi & Lu, 2020; Alain et al., 2015). Instead, we aim to select points that minimize the loss on a *holdout set*. It would be too expensive to naively train on every candidate point and evaluate the holdout loss each time. In this section, we show how to (approximately) find the points that would most reduce the holdout loss if we were to train the current model on them, without actually training on them.

For simplicity, we first assume only one point $(x, y) \in B_t$ is selected for training at each time step t (we discuss selection of multiple points below). $\mathrm{p}(y' \mid x'; \mathcal{D}_\mathrm{t})$ is the predictive distribution of the current model, where $\mathcal{D}_\mathrm{t}$ is the sequence of data the model was trained on before training step t. $\mathcal{D}_\mathrm{ho} = \{(x_i^\mathrm{ho}, y_i^\mathrm{ho})\}_{i=1}^{n^\mathrm{ho}}$, written as $\mathbf{x}^\mathrm{ho}$ and $\mathbf{y}^\mathrm{ho}$ for brevity, is a holdout set drawn from the same data-generating distribution $\mathrm{p}_\mathrm{true}(x', y')$ as the training set $\mathcal{D}$. We aim to acquire the point $(x, y) \in B_t$ that, if we were to train on it, would minimize the negative log-likelihood/cross-entropy loss on the holdout set:

$$\arg\min_{(x,y)\in B_t} - \log \mathrm{p}(\mathbf{y}^\mathrm{ho} \mid \mathbf{x}^\mathrm{ho}; \mathcal{D}_\mathrm{t} \cup (x, y)). \tag{9.1}$$

For a model using a point estimate of θ (such as an MLE or MAP), rather than a distribution over θ, the holdout loss factorises and (up to a constant factor) forms a Monte Carlo approximation of the expected loss under p_true: $\mathbb{E}_{\mathrm{p}_\mathrm{true}(x',y')}[L[y' \mid x'; \mathcal{D}_\mathrm{t} \cup (x, y)]] \approx \frac{1}{|\mathcal{D}_\mathrm{ho}|} \sum_{(x_i^\mathrm{ho}, y_i^\mathrm{ho}) \in \mathcal{D}_\mathrm{ho}} L[y_i^\mathrm{ho} \mid x_i^\mathrm{ho}; \mathcal{D}_\mathrm{t} \cup (x, y)]$, where $L[\cdot]$ denotes the cross-entropy loss: $L[y \mid x] := -\log \mathrm{p}(y \mid x)$.

Deriving a tractable selection function. We now derive a tractable expression for the term in Eq. (9.1) that does not require us to train on each candidate point $(x, y) \in B_t$ and then evaluate the loss on $\mathcal{D}_\mathrm{ho}$. To make our claims precise and our assumptions transparent, we use the language of Bayesian probability theory. We treat model parameters as a random variable with prior $\mathrm{p}(\theta)$ and infer a posterior $\mathrm{p}(\theta|\mathcal{D}_\mathrm{t})$ using the already-seen training data $\mathcal{D}_\mathrm{t}$. The model has a predictive distribution $\mathrm{p}(y|x, \mathcal{D}_\mathrm{t}) = \int_\theta \mathrm{p}(y|x, \theta)\, \mathrm{p}(\theta|\mathcal{D}_\mathrm{t})\, d\theta$. When using a point estimate of θ, the predictive distribution can be written as an integral with respect to a Dirac delta.

Using Bayes rule and the conditional independence $\mathrm{p}(y_i \mid x_i, x_j; \mathcal{D}_\mathrm{t}) = \mathrm{p}(y_i \mid x_i; \mathcal{D}_\mathrm{t})$, we can derive a tractable selection function from Eq. (9.1). For readability, we switch the sign of the selection function, later changing the minimization to a maximization.

$$
\begin{aligned}
&\log \mathrm{p}(\mathbf{y}^{\mathrm{ho}} \mid \mathbf{x}^{\mathrm{ho}}; \mathcal{D}_{\mathrm{t}} \cup (x, y)) \\
&= \log \frac{\mathrm{p}(y \mid x; \mathbf{x}^{\mathrm{ho}}, \mathbf{y}^{\mathrm{ho}}, \mathcal{D}_{\mathrm{t}})\, \mathrm{p}(\mathbf{y}^{\mathrm{ho}} \mid \mathbf{x}^{\mathrm{ho}}, x; \mathcal{D}_{\mathrm{t}})}{\mathrm{p}(y \mid x, \mathbf{x}^{\mathrm{ho}}; \mathcal{D}_{\mathrm{t}})} && \text{Bayes rule} \\
&= \log \frac{\mathrm{p}(y \mid x; \mathbf{y}^{\mathrm{ho}}, \mathbf{x}^{\mathrm{ho}}, \mathcal{D}_{\mathrm{t}})\, \mathrm{p}(\mathbf{y}^{\mathrm{ho}} \mid \mathbf{x}^{\mathrm{ho}}; \mathcal{D}_{\mathrm{t}})}{\mathrm{p}(y \mid x; \mathcal{D}_{\mathrm{t}})} && \begin{array}{c}\text{conditional}\\ \text{independence}\end{array} \\
&\propto L[y \mid x; \mathcal{D}_{\mathrm{t}}] - L[y \mid x; \mathcal{D}_{\mathrm{ho}}, \mathcal{D}_{\mathrm{t}}],
\end{aligned}
\tag{9.2}
$$

where in the final line, we dropped terms independent of (x, y), rearranged, and applied the definition of $L[\cdot]$.

As exact Bayesian inference (conditioning on $\mathcal{D}_{\mathrm{t}}$ or $\mathcal{D}_{\mathrm{ho}}$) is intractable in neural networks (Blundell et al., 2015), we fit the models with SGD instead (**Approximation 1**). We study the impact of this approximation in Section 9.4.1. The first term, $L[y \mid x; \mathcal{D}_{\mathrm{t}}]$, is then the *training loss* on the point (x, y) using the current model trained on $\mathcal{D}_{\mathrm{t}}$. The second term, $L[y \mid x; \mathcal{D}_{\mathrm{ho}}, \mathcal{D}_{\mathrm{t}}]$, is the loss of a model trained on $\mathcal{D}_{\mathrm{t}}$ and $\mathcal{D}_{\mathrm{ho}}$.

Although the selection function in Eq. (9.2) is tractable, it is still somewhat expensive to compute, as both terms must be updated after each acquisition of a new point. However, we can approximate the second term with a model trained only on the holdout dataset, $L[y \mid x; \mathcal{D}_{\mathrm{ho}}, \mathcal{D}_{\mathrm{t}}] \approx L[y \mid x; \mathcal{D}_{\mathrm{ho}}]$ (**Approximation 2**). This approximation saves a lot of compute: it is now sufficient to compute the term once before the first epoch of training. Later on, we show that this approximation empirically does not hurt performance on any tested dataset and even has some desired properties (Section 9.4.1). We term $L[y \mid x; \mathcal{D}_{\mathrm{ho}}]$ the *irreducible holdout loss* (IL) since it is the remaining loss on point $(x, y) \in \mathcal{D}$ after training on the holdout set $\mathcal{D}_{\mathrm{ho}}$; in the limit of $\mathcal{D}_{\mathrm{ho}}$ being large, it would be the lowest loss that the model can achieve without training on (x, y). Accordingly, we name our approximation of Eq. (9.2) the *reducible holdout loss*—the difference between the training loss and the irreducible holdout loss (IL).

Our method still requires us to train a model on a holdout set, but a final approximation greatly reduces that cost. We can efficiently compute the IL with an "irreducible loss model" (IL model) that is smaller than the target model and has low accuracy (**Approximation 3**). We show this and explain it in Sections 9.4.1, 9.4.2, and 9.4.3. Counterintuitively, the reducible holdout loss can therefore be negative. Additionally, one IL model can be reused for many target model runs, amortizing its cost (Section 9.4.2). For example, we trained all 40 seeds of 5 target architectures in Fig. 9.1 using a single ResNet18 IL model. Further, this model trained for 37x fewer steps than each target model (reaching only 62% accuracy). Section 9.5 details further efficiency improvements. o

In summary, selecting a point that minimizes the holdout loss in Eq. (9.1), for

Algorithm 2 Reducible holdout loss selection (RHO-LOSS)

```
1: **Input:** Model $\mathrm{p}(y \mid x; \mathcal{D}_{\mathrm{ho}})$ trained on a holdout set $\mathcal{D}_{\mathrm{ho}}$, batch size $n_b$, large batch size $n_B > n_b$, learning rate $\eta$.
2: **for** $(x_i, y_i)$ in training set **do**
3:     IrreducibleLoss[i] $\leftarrow L[y_i \mid x_i; \mathcal{D}_{\mathrm{ho}}]$
4: **end for**
   Initialize parameters $\theta^0$ and $t = 0$
5: **for** $t = 0, 1, \ldots$ **do**
6:     Randomly select a large batch $B_t$ of size $n_B$.
7:     $\forall i \in B_t$, compute Loss[i], the train loss of point $i$ given parameters $\theta^t$
8:     $\forall i \in B_t$, compute RHOLOSS[i]$\leftarrow$ Loss[i] $-$ IrreducibleLoss[i]
9:     $b_t \leftarrow$ top-$n_b$ samples in $B_t$ in terms of RHOLOSS.
10:    $g_t \leftarrow$ mini-batch gradient on $b_t$ using parameters $\theta^t$
11:    $\theta^{t+1} \leftarrow \theta^t - \eta g_t$
12: **end for**
```

a model trained on $\mathcal{D}_{\mathrm{t}}$, can be approximated with the following easy-to-compute objective:

Reducible holdout loss selection (RHO-LOSS)

$$\underset{(x,y)\in B_t}{\arg\max} \; \overbrace{\underbrace{L[y \mid x; \mathcal{D}_{\mathrm{t}}]}_{\text{training loss}} \quad - \quad \underbrace{L[y \mid x; \mathcal{D}_{\mathrm{ho}}]}_{\text{irreducible holdout loss (IL)}}}^{\text{reducible holdout loss}} \tag{9.3}$$

Although we required additional data $\mathcal{D}_{\mathrm{ho}}$, this is not essential for large (Section 9.4.0) nor small (Section 9.4.2) datasets.

Understanding reducible loss. We now provide intuition on why reducible holdout loss selection (RHO-LOSS) avoids redundant, noisy, and less relevant points. **i) Redundant points.** We call a training point redundant when the model has already learned it, i.e. its training loss cannot be further reduced. Since redundant points have low training loss, and the reducible loss is always less than the training loss (Eq. (9.3)), such points have low reducible loss and are not selected. **ii) Noisy points.** While prior methods select based on high training loss (or gradient norm), not all points with high loss are informative—some may have an ambiguous or incorrect (i.e. noisy) label. The labels of such points cannot be predicted using the holdout set (Chen et al., 2019). Such points have high IL and, consequently, low reducible loss. These noisy points are less likely to be selected compared to equivalent points with less noise. **iii) Less relevant points.** Loss-based selection has an additional pitfall. The training loss is likely higher for outliers in input space—values of x far from most of the training data, in regions with low input density under $\mathrm{p}_{\mathrm{true}}(x)$. Points with low $\mathrm{p}_{\mathrm{true}}(x)$ should not be prioritized, all else equal. Consider an 'outlier' (x, y) and a non-outlier (x', y'), with $\mathrm{p}_{\mathrm{true}}(x) < \mathrm{p}_{\mathrm{true}}(x')$ but *equal* training loss $L[y|x; \mathcal{D}_{\mathrm{t}}] = L[y'|x'; \mathcal{D}_{\mathrm{t}}]$. As the holdout set $\mathcal{D}_{\mathrm{ho}}$ is also drawn from $\mathrm{p}_{\mathrm{true}}$, $\mathcal{D}_{\mathrm{ho}}$ will contain fewer points from the region around x in input space compared to the region around x'. Thus, training on (x, y) is likely to reduce the holdout loss (Eq. (9.1)) less, and so we prefer to train on the non-outlier (x', y'). In the specific sense described, (x, y) is thus less *relevant* to the holdout set. As desired, RHO-LOSS deprioritizes (x, y): since $\mathcal{D}_{\mathrm{ho}}$ contains few points from the region around x, the IL of (x, y) will be large.

In short, RHO-LOSS *deprioritizes* points that are redundant (low training loss), noisy (high IL), or less relevant to the holdout set (high IL). That is, RHO-LOSS *prioritizes* points that are not yet learned, learnable, and worth learning. We provide empirical evidence for these claims in Section 9.4.3. See Algorithm 2 for the implementation of RHO-LOSS.

Selecting multiple points concurrently. We showed which point is optimal when selecting a single point (x, y). When selecting an entire batch b_t, we select the points with the top-n_b scores from the randomly pre-sampled set B_t. This is nearly optimal when assuming that each point has little effect on the score of other points, which is often used as a simplifying assumption in active learning (Kirsch et al., 2019). This assumption is much more reasonable in our case than in active learning because model predictions are not changed much by a single gradient step on one mini-batch.

Simple parallelized selection. For large-scale neural network training, practitioners with sufficient resources would use many more machines if it further sped up training (Anil et al., 2018). However, as more workers are added in synchronous or asynchronous gradient descent, the returns diminish to a point where adding more workers does not further improve wall clock time (Anil et al., 2018; McCandlish et al., 2018). For example, there are rapidly diminishing returns for using larger batch sizes or distributing a given batch across more workers, for multiple reasons (McCandlish et al., 2018; Keskar et al., 2016). The same holds for distributing the model across more workers along its width or depth dimension (Rasley et al., 2020; Shoeybi et al., 2019; Huang et al., 2019). However, we can circumvent these diminishing returns by adding a new dimension of parallelization, namely, for data selection.

Since parallel *forward* passes do not suffer from such diminishing returns, one can use extra workers to evaluate training losses in parallel (Alain et al., 2015). The theoretical runtime speedup can be understood as follows. The cost per training step of computing the selection function on B_t is $\frac{n_B}{3n_b}$ times as much as the cost of the forward-backward pass needed to train on b_t since a forward pass requires at least

3x less computation than a forward-backward pass (Jouppi et al., 2017). One can reduce the time for the selection phase almost arbitrarily by adding more workers that compute training losses using a copy of the model being trained. The limit is reached when the time for selection is dominated by the communication of parameter updates to workers. More sophisticated parallelization strategies allow reducing the time overhead even further (Section 9.5). To avoid assumptions about the particular strategy used, we report experiment results in terms of the required number of training epochs.

9.4 Experiments

We evaluate our selection method on several datasets (both in controlled environments and real-world conditions) and show significant speedups compared to prior art, in the process shedding light on the properties of different selection functions.

Recall that our setting assumes training time is a bottleneck but data is abundant—more than we can train on (see Bottou & LeCun (2004)). This is common e.g. for web-scraped data where state-of-the-art performance is often reached in less than half of one epoch (Komatsuzaki, 2019; Brown et al., 2020b). As data is abundant, we can set aside a holdout set for training the IL model with little to no downside. For the large Clothing-1M dataset, we implement RHO-LOSS by training the IL model on 10% of the training data, while all baselines are trained on the full 100% of the training data. For the smaller datasets, we simulate abundance of data by reserving a holdout set and training *all* methods only on the remaining data. However, RHO-LOSS also works on small datasets without additional data by double-using the training set (Section 9.4.2).

Datasets. We evaluate on 7 datasets: 1) QMNIST (Yadav & Bottou, 2019) extends MNIST (LeCun et al., 1998a) with 50k extra images which we use as the holdout set. 2) On CIFAR-10 (Krizhevsky & Hinton, 2009) we train on half of the training set and use the other half as a holdout to train the irreducible loss (IL) model. 3) CIFAR-100: same as CIFAR-10. 4) CINIC-10 (Darlow et al., 2018) has 4.5x more images than CIFAR-100 and includes a holdout set and a test set with 90k images each. 5) Clothing-1M (Xiao et al., 2015), which contains over 1 million 256x256-resolution clothing images from 14 classes. The dataset is fully web-scraped—a key application area of our work—and is the most widely accepted benchmark for image recognition with noisy labels (Algan & Ulusoy, 2021). We use the whole training set for training and reuse 10% of it to train the IL model. We further evaluate on two NLP datasets from GLUE (Wang et al., 2018): 6) CoLA (grammatical acceptability) and 7) SST-2 (sentiment). We split their training sets as for CIFAR.

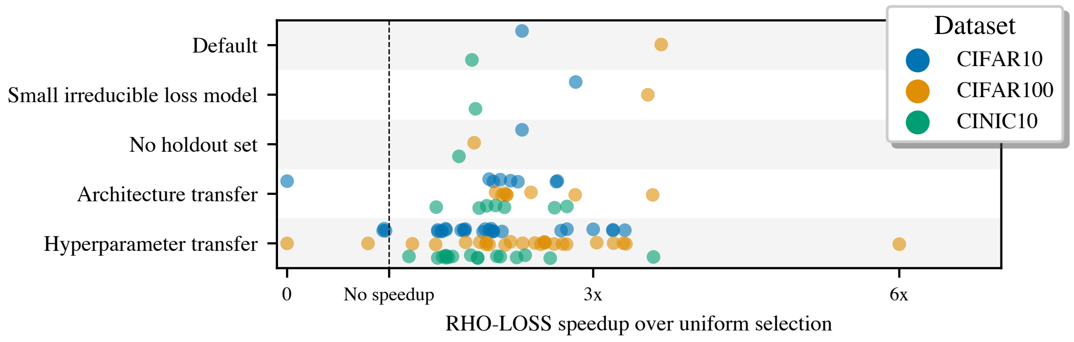


Figure 9.2: The irreducible loss model can be small, trained with no holdout data, and reused across target architectures and hyperparameters. Here, we use clean datasets, where speedups are smallest. The x-axis shows speedup, i.e. after how many fewer epochs RHO-LOSS exceeds the highest accuracy uniform selection achieves within 100 epochs. Row 1 uses a ResNet18 as irreducible loss model. All other rows instead use a small, cheap CNN. Each dot shows an experiment with a given combination of irreducible loss model and target model (mean across 2-3 seeds for all but the last row).

Baselines. Aside from uniform sampling (without replacement, i.e. random shuffling), we also compare to selection functions that have achieved competitive performance in online batch selection recently: the (training) loss, as implemented by Kawaguchi & Lu (2020), gradient norm, and gradient norm with importance sampling (called *gradient norm IS* in our figures), as implemented by Katharopoulos & Fleuret (2018). We also compare to the core-set method Selection-via-Proxy (SVP) that selects data offline before training (Coleman et al., 2020). We report results using maximum entropy SVP and select with the best-performing model, ResNet18. Finally, we include selection using the negative IL (see Eq. 9.3) to test if it is sufficient to only skip noisy and less relevant but not redundant points.

Models and hyperparameters. To show our method needs no tuning, we use the PyTorch default hyperparameters (with the AdamW optimizer (Loshchilov & Hutter, 2017)) and $\frac{n_b}{n_B} = 0.1$. We test many additional hyperparameter settings in Figs. 9.2 (row 5) and 9.6. We test various architectures in Figs. 9.1 and 9.2 (row 4). In all other figures, we use a 3 layer MLP for experiments on QMNIST, a ResNet-18 adapted for small images for CIFAR-10/CIFAR-100/CINIC-10, and a ResNet-50 for Clothing-1M. All models for Clothing-1M are pre-trained on ImageNet (standard for this dataset Algan & Ulusoy (2021)) and the IL model is *always* a ResNet-18. For the NLP datasets, we use a pretrained ALBERT v2 (Lan et al., 2019). We always use the IL model checkpoint with lowest validation loss (not highest accuracy); this performs best.

Evaluation. We measure speedup in terms of the number of epochs needed to reach a given test accuracy. We measure epochs needed, rather than wall clock time, as our

Table 9.1: Spearman's rank correlation of rankings of data points by selection functions that are increasingly less faithful approximations of Eq. (9.2), compared to the most faithful approximation. Approximations added from left to right. Mean across 3 seeds.

	Non-Bayesian	Not converged	Not updating IL model	Small IL model
Rank correlation	0.75	0.76	0.63	0.51

focus is on evaluating a new selection function, not an entire training pipeline. Wall clock time depends primarily on the hardware used and implementation details that are beyond our scope. Most importantly, data selection is amenable to parallelization beyond standard data parallelism as discussed in Section 9.3.

9.4.1 Impact of Approximations

In Section 9.3, we introduced a function for selecting exactly the points that most reduce the model's loss on a holdout set. To make this selection function efficient for deep neural networks, we made several approximations. Here, we study how these approximations affect the points selected, by successively introducing one approximation after the other.

Because the exact selection function (Eq. (9.2)) is intractable, we start with a close (and expensive) approximation as the gold standard (Approximation 0). To make Approximation 0 feasible, the experiments are conducted on an easy dataset—QMNIST (with 10% uniform label noise and data duplication to mimic the properties of web-scraped data). We then successively introduce the Approximations 1, 2, and 3 described in Section 9.3. To assess the impact of each approximation, we train a model without and with the approximations, and then compute the rank correlation (Spearman's correlation coefficient) of the selection function evaluated on each batch B_t. Across the first epoch, we present the mean of the rank correlations. Since each approximation selects different data, the corresponding models become more different over time; this divergence causes some of the observed difference in the points they select.

Approximation 0. To get as close as possible to the Bayesian inference/conditioning used in Eq. (9.2), we use a deep ensemble of 5 neural networks and train them *to convergence* after every time step t on the acquired dataset $b_t \cup \mathcal{D}_t$ Wilson & Izmailov (2020).

Approximation 1: SGD instead of Bayesian inference/conditioning. Approximation 0 is a close approximation of Eq. (9.2), but training an ensemble to convergence at every step t is far too expensive in practice. Starting from this gold-standard, we introduce two stronger approximations (1a and 1b) to move to standard neural network

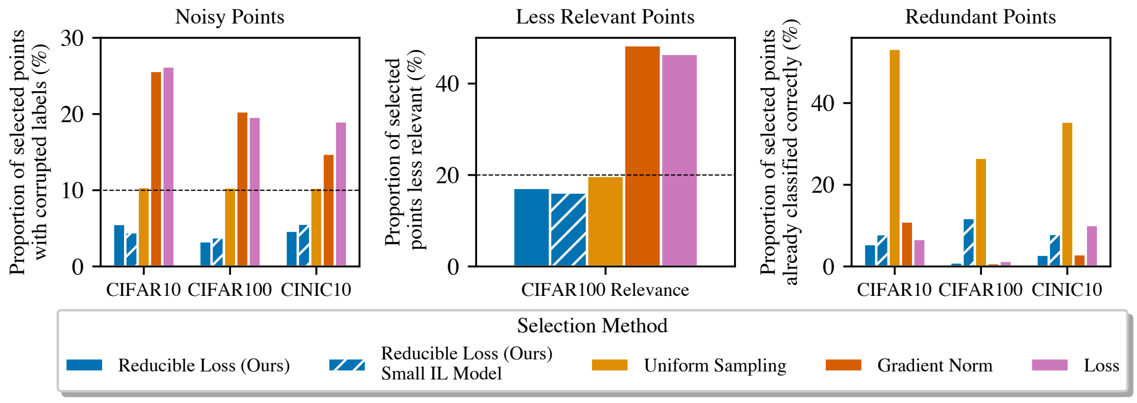


Figure 9.3: Properties of RHO-LOSS and other methods. RHO-LOSS prioritizes points that are non-noisy, task-relevant, and non-redundant—even when the irreducible loss (IL) model is a small CNN. In contrast, loss and gradient norm prioritize noisy and less relevant points (while also avoiding redundant points). **Left.** Proportion of selected points with corrupted labels. We added 10% uniform label noise, i.e., we randomly switched each point's label with 10% probability. **Middle.** Proportion of selected points from low relevance classes on CIFAR100 Relevance dataset. **Right.** Proportion of selected points that are already classified correctly, which is a proxy for redundancy. Mean over 150 epochs of training and 2-3 seeds.

fitting with AdamW. 1a) First, we replace the ensemble with a single model, while still training to convergence at each time step. The Spearman's coefficient between this approximation and Approximation 0 is 0.75, suggesting similar points are selected ("Non-Bayesian" in Table 9.1). 1b) Next, we only take one gradient step on each new batch b_t. The Spearman's coefficient, when comparing this to Approximation 0, is 0.76 ("Not Converged" in Table 9.1).

Approximation 2. Not updating the IL model on the acquired data $\mathcal{D}_t$. Second, we save compute by approximating $L[y \mid x; \mathcal{D}_{\mathrm{t}}, \mathcal{D}_{\mathrm{ho}}]$ with $L[y \mid x; \mathcal{D}_{\mathrm{ho}}]$. The points selected are still similar to Approximation 0 (Spearman's coefficient 0.63, "Not updating IL model" in Table 9.1). This approximation also performs well on other datasets.

Approximation 3: Small IL model. Lastly, we use a model with 256 hidden units instead of 512 (4x fewer parameters) as the IL model and see again that similar points are selected (Spearman's coefficient 0.51). We study cheaper IL models in other forms and datasets in the next section.

9.4.2 Cheap Irreducible Loss Models & Robustness

RHO-LOSS requires training an IL model on a holdout set, which poses additional costs. Here, we show how to minimize these costs and amortize them across many training runs of target models. The same experiments also show the robustness of

RHO-LOSS across architectures and hyperparameter settings. To fit our computational budget, we perform these experiments on moderate-sized clean benchmark datasets although RHO-LOSS speeds up training more on noisy or redundant web-scraped data (see Section 9.4.4).

Irreducible loss models can be small and cheap. In our default setting (Fig. 9.2, row 1), both the target model and IL model have the same architecture (ResNet-18). In rows 2 and below, we instead used a small CNN similar to LeNet as the IL model (LeCun et al., 1989). It has 21x fewer parameters and requires 29x fewer FLOP per forward pass than the ResNet-18. **The smaller IL model accelerates training as much or more than the larger model**, even though its final accuracy is far lower than the target ResNet-18 (11.5% lower on CIFAR-10, 7% on CIFAR-100, and 8.1% on CINIC-10). We examine in Section 9.4.3 why this useful result holds.

Irreducible loss models without holdout data. Web-scraped datasets are often so large that even a small fraction of the overall data can be sufficient to train the IL model. E.g., in our experiments on Clothing-1M (Fig. 9.1), the holdout set is only 10% as large as the main train set. Additionally, we can train the IL model without any holdout data (Fig. 9.2, row 3). We split the training set $\mathcal{D}$ into two halves and train an IL model on each half (still using small IL models). Each model computes the IL for the half of $\mathcal{D}$ that it was not trained on. Training two IL models costs no additional compute since each model is trained on half as much data compared to the default settings.

Irreducible loss models can be reused to train different target architectures. We find that a single small CNN IL model accelerates the training of 7 target architectures (Fig. 9.2, row 4): VGG11 (with batchnorm), GoogleNet, Resnet34, Resnet50, Densenet121, MobileNet-v2, Inception-v3. RHO-LOSS does not accelerate training on CIFAR-10 for VGG11, which is also the architecture on which uniform training performs the worst; i.e. RHO-LOSS empirically does not "miss" a good architecture. Not only is RHO-LOSS robust to architectures choice, a single IL model can also be *reused by many practitioners* who use different architectures (as we did in Fig. 9.1).

Irreducible loss models can be reused to train many targets in a hyperparameter sweep. We find that a single small CNN accelerates the training of ResNet-18 target models across a hyperparameter grid search (Fig. 9.2, last row). We vary the batch size (160, 320, 960), learning rate (0.0001, 0.001, 0.01), and weight decay coefficient (0.001, 0.01, 0.1). RHO-LOSS speeds up training compared to uniform on nearly all target hyperparameters. The few settings in which it doesn't speed up training are also settings in which uniform training performs very poorly ($< 30\%$ accuracy on CIFAR-100, $< 80\%$ on CIFAR-10).

Table 9.2: Epochs required to reach a given target test accuracy (final accuracy in parentheses). On CIFAR10/100, CoLA, and SST-2, only half of the data is used for training (Section 9.4), lowering accuracy. Figs. 9.4 and 9.5 (Appendix) show all training curves. Some datasets have 10% uniform label noise added. Results averaged across 2-4 seeds. Best performance in **bold**. RHO-LOSS performs best in both epochs required and final accuracy. *NR* indicates that the target accuracy was not reached.

Dataset	Target Acc	Number of epochs method needs to reach target accuracy ↓ (Final accuracy in parentheses)						
		Train Loss	Grad Norm	Grad Norm IS	SVP	Irred Loss	Uniform	RHO-LOSS
Clothing-1M	60.0%	8	13	2	*NR*	*NR*	2	**1**
	69.0%	*NR* (65%)	*NR* (64%)	9 (70%)	*NR* (55%)	*NR* (48%)	30 (70%)	**2 (72%)**
CIFAR10	80.0%	81	*NR*	57	*NR*	*NR*	79	**39**
	87.5%	129 (90%)	*NR* (61%)	139 (89%)	*NR* (55%)	*NR* (60%)	*NR* (87%)	**65 (91%)**
CIFAR10 (Label Noise)	75.0%	*NR*	*NR*	57	*NR*	*NR*	62	**27**
	85.0%	*NR* (28%)	*NR* (23%)	*NR* (84%)	*NR* (48%)	*NR* (62%)	*NR* (85%)	**49 (91%)**
CIFAR100	40.0%	138	139	71	*NR*	93	65	**48**
	52.5%	*NR* (42%)	*NR* (42%)	132 (55%)	*NR* (18%)	*NR* (43%)	133 (54%)	**77 (61%)**
CIFAR100 (Label Noise)	40.0%	*NR*	*NR*	94	*NR*	89	79	**49**
	47.5%	*NR* (4%)	*NR* (4%)	142 (48%)	*NR* (14%)	*NR* (43%)	116 (50%)	**65 (60%)**
CINIC10	70.0%	*NR*	*NR*	34	*NR*	*NR*	38	**27**
	77.5%	*NR* (36%)	*NR* (50%)	64 (82%)	*NR* (39%)	*NR* (60%)	97 (80%)	**38 (83%)**
CINIC10 (Label Noise)	60.0%	*NR*	*NR*	22	*NR*	30	24	**13**
	67.5%	*NR* (16%)	*NR* (16%)	35 (79%)	*NR* (39%)	*NR* (64%)	38 (78%)	**17 (82%)**
SST2	82.5%	8	2	3	*NR*	7	**1**	**1**
	90.0%	*NR* (87%)	4 (91%)	*NR* (89.7%)	*NR* (66%)	*NR* (83%)	6 (90%)	**3 (92%)**
CoLA	75.0%	8	6	16	*NR*	*NR*	34	**3**
	80.0%	*NR* (78%)	*NR* (79%)	*NR* (78%)	*NR* (62%)	*NR* (69%)	*NR* (76%)	**39 (80%)**

9.4.3 Properties of RHO-LOSS & Other Selection Functions

We established that RHO-LOSS can accelerate the training of various target architectures with a single IL model, even if the IL model is smaller and has considerably lower accuracy than the target models (Section 9.4.2). This suggests robustness to target-IL architecture mismatches.

To understand this robustness, we investigate the properties of points selected by RHO-LOSS, when the target and IL model architectures are identical, and when they differ. In both cases, we find that RHO-LOSS prioritizes points that are non-noisy, task-relevant, and not redundant. We also investigate the properties of points selected by prior art.

Noisy points. We investigate how often different methods select noisy points by uniformly corrupting the labels for 10% of points and tracking what proportion of selected points are corrupted. RHO-LOSS deprioritizes noisy points for both IL models (Fig. 9.3). We observe a failure mode of the widely-used loss and gradient norm selection functions: they select far more noisy points than uniform. These methods also severely drop in accuracy when the noise follows the class confusion matrix (Rolnick et al., 2017) and when we add ambiguous images (Mukhoti et al., 2021).

Together, this suggests that noisy points have high loss (and gradient norm), but also high IL and thus low reducible loss. Their IL is high even when the IL model is

Figure 9.4: Training time for parallelized data selection methods. Wall clock time estimated with Amdahl's law **b)** Training times with 10% label noise.

small as noisy labels cannot be predicted well using the holdout set.

Relevant points. We study how often less relevant points are selected by creating the CIFAR100 Relevance dataset, in which 80% of the data comes from 20% of the classes. This mimics natural distributions of NLP and vision data where most data comes from few object classes, topics, or words (Baayen, 2001; Tian et al., 2021). Concretely, we retain all examples from 20 randomly chosen "high relevance" classes but only 6% of the examples from other, "low relevance" classes. Intuitively, since the high relevance classes have higher $p_{\text{true}}(x)$ and are 17x more likely to appear at test time, improving their accuracy improves the test accuracy much more than improving the accuracy of less relevant classes.

The loss and gradient norm methods select more points than uniform selection from the low relevance classes (Fig. 9.3). In contrast, RHO-LOSS selects somewhat fewer low relevance points, suggesting these classes have high IL. Since the less relevant classes are less abundant in the holdout set, both the small and large IL models have higher loss on them.

Redundant points. To investigate whether methods select redundant points, we track the percentage of selected points that are already classified correctly. This is only a proxy for redundancy; points that are classified correctly but with low confidence are not fully redundant, since their loss can be further reduced. We control for the different accuracy reached by each method by averaging only over epochs in which test accuracy is lower than the final accuracy reached by the weakest performing method. Fig. 9.3 shows that all methods select fewer redundant points than uniform sampling.

9.4.4 Speedup

Finally, we evaluate how much different selection methods speed up training. Recall that the main application area for our work is large web-scraped datasets, known for high levels of noise and redundancy. Clothing-1M is such a dataset (Section 9.4.0). We also include smaller, clean benchmarks from vision (CIFAR-10, CIFAR-100, CINIC-10) and NLP (CoLA, SST-2). Finally, we study if selection functions are robust to the controlled addition of label noise.

Speedup on clean data. RHO-LOSS reaches target accuracies in fewer epochs than uniform selection on all datasets (Table 9.2). It also outperforms state-of-the-art methods by a clear margin in terms of speed and final accuracy. On the challenging CoLA language understanding dataset, the speedup over uniform selection exceeds 10x. In Table 9.3, we find similar speedups when using no holdout data.

Speedup on noisy data. When adding 10% label noise, batch selection with RHO-LOSS achieves greater speedups while, as hypothesized, prior art degrades (Table 9.2). Notably, on noisier data, the speedup over uniform selection grows.

Speedup on large web-scraped data. On Clothing-1M, loss-based and gradient norm-based selection fail to match uniform selection, suggesting they are not robust to noise. In contrast, RHO-LOSS reaches the highest accuracy that uniform selection achieves during 50 epochs in just 2 epochs and improves final accuracy (72% vs 70%). Notably, this was possible even though the IL model we used has low accuracy (62.2%) and was trained on ca. 10x less data. RHO-LOSS also used 2.7x fewer FLOPs to reach the peak accuracy of uniform selection, including the cost of training the IL model (which could be amortized) and despite our implementation being designed to save time, not compute. Table 9.2 differs from Fig. 9.1 as Table 9.2 shows only ResNet-50 and more epochs.

9.5 Related Work

Time-efficient data selection. Forward passes for selection can be accelerated using low-precision hardware or parallelization. While backward passes typically require high precision, forward passes can tolerate lower precision (Jouppi et al., 2017; Jiang et al., 2019). A forward pass by default requires roughly 3x less time than a forward-backward pass but this speedup can be increased to a factor around 10x when using the low-precision cores available in modern GPUs and TPUs (Jouppi et al., 2017; Jiang et al., 2019). Further, prior work uses a set of workers that perform forward passes on B_t or on the entire dataset asynchronously while the master process trains on recently selected data (Alain et al., 2015).

Compute-efficient data selection. While we limit our scope to comparing selection functions and we compute them naively, this choice is inefficient in practice. Selection can be made cheaper by reusing losses computed in previous epochs (Loshchilov & Hutter, 2015; Jiang et al., 2019) or training a small model to predict them (Katharopoulos & Fleuret, 2017; Zhang et al., 2019; Coleman et al., 2020). Alternatively, core set methods perform selection once before training (Mirzasoleiman et al., 2020; Borsos et al., 2020), although typically with more expensive selection functions.

Data selection functions. RHO-LOSS is best understood as an alternative to existing selection functions, which can be categorized by the properties of points they select and whether they use information about labels. “Hard” points are selected both by high loss (Loshchilov & Hutter, 2015; Kawaguchi & Lu, 2020; Jiang et al., 2019) and high prediction uncertainty (Settles, 2009; Li & Sethi, 2006; Coleman et al., 2020). However, prediction uncertainty does not require labels and can thus be used for active learning. Despite this, they both suffer from the same problem: high loss and high uncertainty can be caused by noisy (in particular, ambiguous) labels. This also applies to selection of points whose labels are easily forgotten during training (Toneva et al., 2018). Noisy points are avoided by our negative IL baseline and similar methods (Pleiss et al., 2020; Chen et al., 2019). Points that reduce (expected) holdout loss are also selected for other applications (Kirsch et al., 2021; Killamsetty et al., 2020; Ren et al., 2018), although using much more computation.

Variance reduction methods. Online batch selection is also used to reduce the variance of the gradient estimator computed by SGD (Katharopoulos & Fleuret, 2018, 2017; Johnson & Guestrin, 2018; Alain et al., 2015). Such methods typically use importance sampling—points with high (approximate) gradient norm are sampled with high probability but then down-weighted in the gradient calculation to de-bias the gradient estimate. Without de-biasing, methods like RHO-LOSS also create selection bias. However, bias can improve test performance, both in theory and practice (Farquhar et al., 2021; Kawaguchi & Lu, 2020).

9.6 Conclusion

To reduce excessive training times, we introduce a theoretically grounded selection function that enables substantial speedups on clean data and even larger speedups on noisy and web-scraped data. By illuminating three properties of optimal selection, we hope to motivate new directions in batch selection. However, our selection function should be combined with methods in Section 9.5 for cheap and fast selection with maximal speedups.

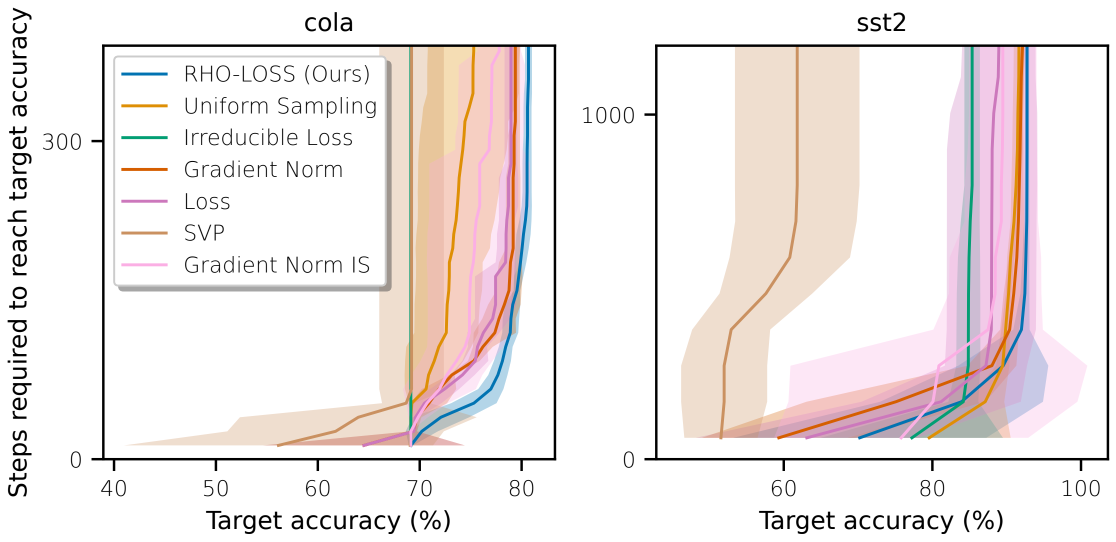


Figure 9.5: NLP datasets—gradient steps required to achieve a given test accuracy **(lower is better)**. **Left:** CoLA grammatical acceptibility classification. **Right:** SST2 sentiment classification. A step corresponds to lines $5-10$ in Algorithm 1. Lines correspond to means and shaded areas to standard deviations across 4 or more random seeds. Only half of the data is used for training (see text).

Table 9.3: Epochs required to reach a given target test accuracy when using no holdout data (lower is better). Final accuracy in parentheses. Results averaged across 2-3 seeds. Best performance in bold. RHO-LOSS performs best in both epochs required and final accuracy.

Dataset	Target Acc	Uniform	RHO-LOSS
CIFAR10	80%	39	**17**
	90%	177 (90.8%)	**47 (92.2%)**
CIFAR100	50%	47	**22**
	65%	142 (67.8%)	**87 (68.1%)**
CINIC10	70%	37	**26**
	80%	146 (80.1%)	**70 (82.1%)**

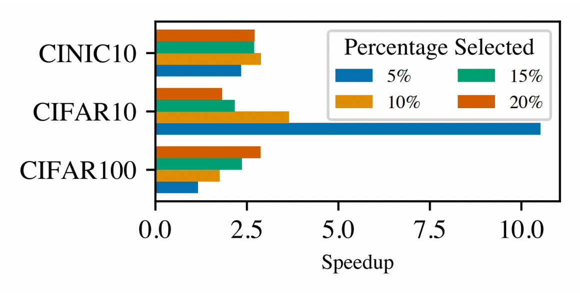


Figure 9.6: Varying the percent of data points selected in each training batch. Average over 3 random seeds.

Chapter 10

Disease Variant Prediction with Deep Generative Models of Evolutionary Data

As dataset scale has increased, practitioners have had to turn towards ”less clean” sources of data such as scraping the web. This results in datasets that contain significant quantities of noise and repetition, which pose a hindrance to model quality. In this work, we propose a model-based data selection method that can substantially improve model quality and reduce the amount of training necessary to reach good performance.

The following is an excerpt from Frazer et al. (2021) and is published in the journal Nature. This is work done in collaboration with Jonathan Frazer, Pascal Notin, Mafalda Dias, Joseph Min, Kelly Brock, Yarin Gal, and Debora Marks. In this project I contributed to the machine learning methodologies and experiment design.

The exponential growth in human genome sequencing has underlined the substantial genetic variation in the human population. Understanding the disease relevance of this genetic variation has the potential to transform healthcare and motivates the massive investment in the collection of human population genomic information together with demographics and clinical data such as the UK BioBank (Van Hout et al., 2020), ChinaMAP (Cao et al., 2020), deCODE (Gudbjartsson et al., 2015). The access to sequencing has enabled both genetic studies that associate variants with diseases and more mechanism-based approaches that associate variants with biochemical and cellular phenotypes. However, relating specific changes in the genome to disease phenotypes remains an open challenge as the number of variants in the human population dwarfs the number we are able to investigate. Protein coding regions alone contain large variation between people; to date, 6.5 million missense variants

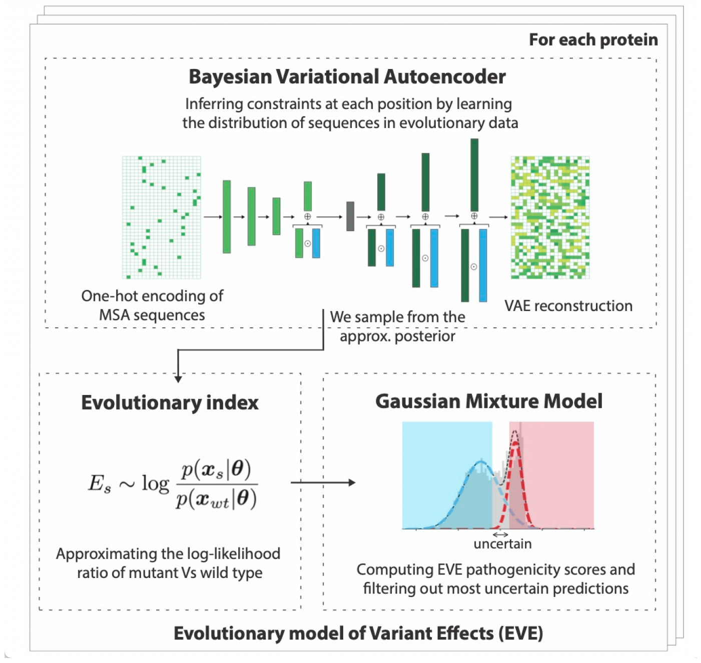

For each protein
Bayesian Variational Autoencoder
Inferring constraints at each position by learning the distribution of sequences in evolutionary data
One-hot encoding of MSA sequences
We sample from the approx. posterior
VAE reconstruction
Evolutionary index
$E_s \sim \log \frac{p(\boldsymbol{x}_s|\boldsymbol{\theta})}{p(\boldsymbol{x}_{wt}|\boldsymbol{\theta})}$
Approximating the log-likelihood ratio of mutant Vs wild type
Gaussian Mixture Model
uncertain
Computing EVE pathogenicity scores and filtering out most uncertain predictions
Evolutionary model of Variant Effects (EVE)

Figure 10.1: For each protein, a Bayesian VAE learns a distribution over amino acid sequences from evolutionary data. This enables the computation of the evolutionary index for each single-variant sequence, which approximates the log-likelihood ratio of variant vs wild type sequences. A global-local mixture of GMM separates variants into benign and pathogenic clusters based on that index. The outcome of the model is both a continuous score that reflects pathogenicity propensity, and probabilistic assignment to benign and pathogenic classes below a user-defined uncertainty threshold (Extended Data Figs 1, 3).

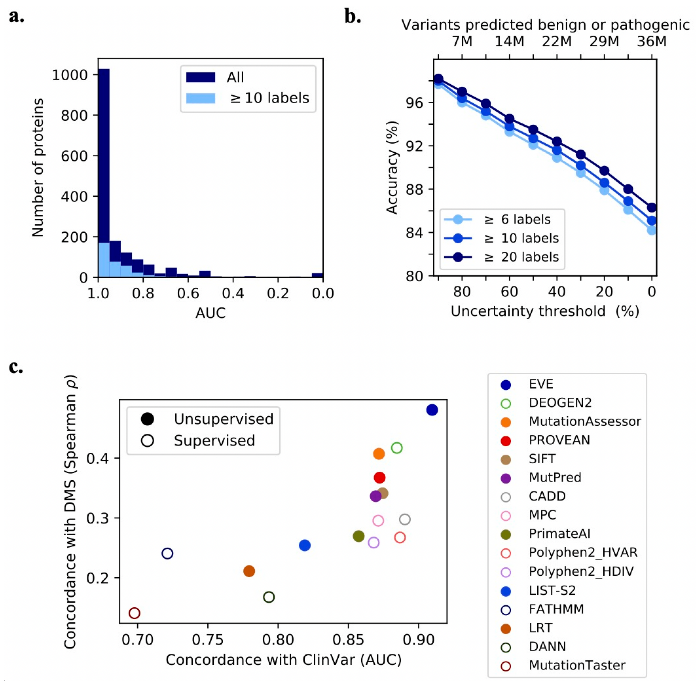


Figure 10.2: a. Distribution of AUC for EVE scores computed over known clinical labels from ClinVar, on all 3,219 proteins covered by our study (dark blue) and for the subset of proteins with at least 5 Benign and 5 Pathogenic known labels (pale blue). b. Tradeoff between accuracy of EVE and the uncertainty threshold (percentage of variants set as Uncertain) or the total number of variants given a class assignment. Accuracy computed over all labels for proteins with at least 3 (5 or 10) Benign labels and 3 (5 or 10) Pathogenic labels. c. Performance comparison of EVE to state-of-the-art computational variant effect predictors: 7 unsupervised and 8 supervised. Performance estimated against known clinical labels (average AUC over disease genes in ClinVar – x-axis), and against high-throughput functional assays developed to assess clinical impact of variants (average spearman correlation – y-axis). (Supplementary Note 2, Extended Data Fig. 4, Supplementary Table 2, 3, 4)

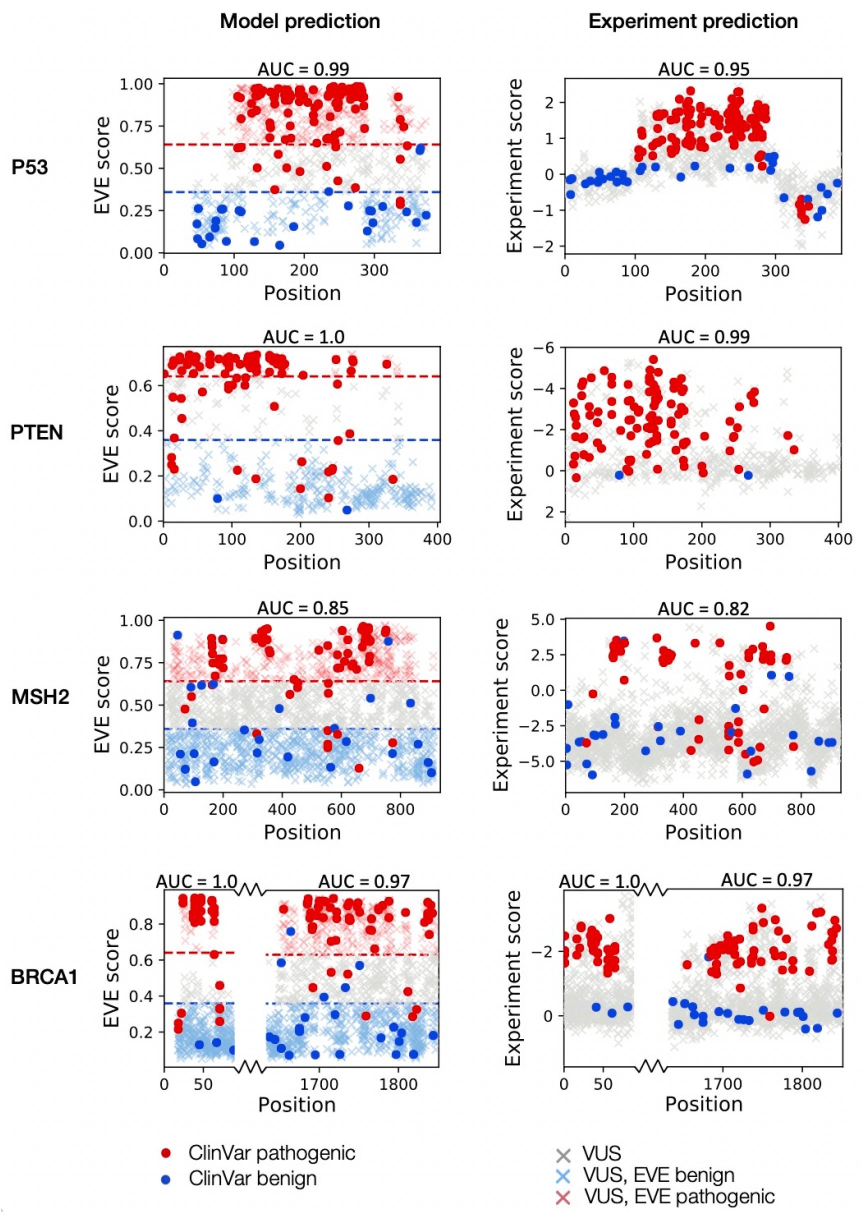


Figure 10.3: Comparison of computational model predictions (left panels, y-axis EVE score) and experimental predictions (right panels, y-axis experimental score) to ClinVar labels (dots) and variants of unknown significance VUS (crosses), where pale red and pale blue crosses indicate EVE predictions; the x-axis corresponds to position in protein. Dashed red and blue lines correspond to EVE predictions setting the 25% most uncertain assignments as Uncertain (see Methods). Experimental data from deep mutational scans of P5313, PTEN19, MSH243, BRCA112 and patch clamp assay of SCN5A14. (Extended Data Fig. 5, Supplementary Table 6).

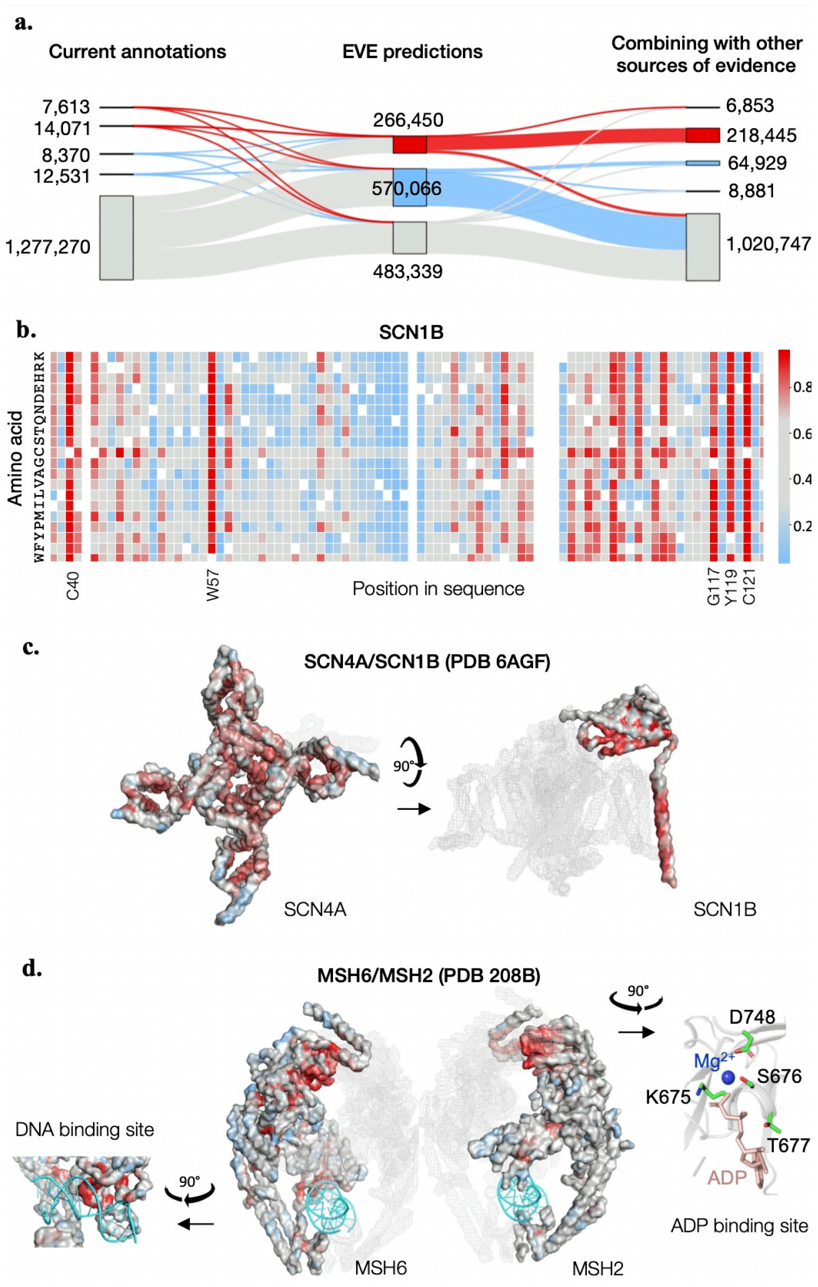


Figure 10.4: a) Combining EVE classifications with other sources of evidence. Left hand-side: ClinVar labels and variants of unknown significance based on gnomAD and UKBiobank; middle: EVE predictions setting 25% of all possible variants as Uncertain; Right hand-side: predictions after combining EVE with other sources of evidence (Methods, Supplementary Table 2). b. Heatmap of EVE pathogenicity scores in SCN1B. c. and d. Representations of 3D structures of SCN4A/SCN1B (PDB 6agf38) and MSH6/MSH2 bound to ADP and a G T mispair (PDB 2o8b41), colored by mean score per position (SCN4A, MSH6, MSH2) and maximum score per position (SCN1B). Clusters of high pathogenicity in 3D include the pore region of SCN4A, hydrophobic core of SCN1B (positions 40, 57, 117, 119, 121), c-terminus alpha-helix of SCN1B and interface with SCN4A, ADP binding site of MSH2 (such as D748N/V/H, K675E, S676L, T677R) and DNA binding site of MSH6.

have been observed (gnomAD (Karczewski et al., 2020)) and the consequences of the vast majority (98%) of these, even in disease related genes, are unknown (Landrum & Kattman, 2018). It is estimated that there will be a variant for every protein position (bar lethal) somewhere in the 9 billion human population.

Given this challenge, new experimental technologies have emerged that can assess the effects of thousands of mutations in parallel (sometimes called Deep Mutational Scans DMSs or Multiplexed Assays of Variant Effects MAVEs) (Esposito et al., 2019; Trenkmann, 2018). The results of these high-throughput experiments are then scrutinized by expert panels such as ClinGen (Richards et al., 2015) for assigning clinical interpretation to human variants. However, these technologies do not easily scale to thousands of proteins, especially not to combinations of variants, and depend critically on the availability of assays that are relevant to or at least associated with human disease phenotypes.

Ideally, computation could also accelerate clinical variant interpretation. However, state-of-art computational methods are supervised on clinical labels in a way that causes inflated accuracy in real-world prediction scenarios. This inflated performance results from variant aggregation across genes (label bias), label sparsity, label noise (Findlay et al., 2018) and data leakage (Gudbjartsson et al., 2015). It is hardly surprising, therefore, that there is some hesitation to use computational methods for anything but "weak evidence" for variant classification, as in the guidelines from The American College of Medical Genetics and Association for Molecular Pathology (ACMG-AMP) (Richards et al., 2015). By contrast, unsupervised probabilistic models of evolutionary sequences alone have been remarkably successful at predicting the effects of variants on protein function and stability (Hopf et al., 2017; Marks et al., 2011; Hopf et al., 2014; Lapedes et al., 2012) and are fundamentally generalizable as they avoid learning from labels. However, there has been little progress in developing these models to address disease relevance since early pioneering efforts (Vaser et al., 2016; Reva et al., 2011).

In this work we revisit the clinical value of evolutionary information in light of recent developments in unsupervised generative modelling. We introduce EVE (Evolutionary model of Variant Effect), a new computational method for the classification of human genetic variants trained solely on evolutionary sequences. We show that EVE outperforms current state-of-the-art computational methods at predicting variant pathogenicity (without the risk of overfitting clinical labels), and is surprisingly as accurate as predictions from high-throughput experiments.

10.1 Predicting pathogenicity from evolution

Our method – EVE – learns the propensity of human missense variants to be pathogenic from the distribution of sequence variation across species (Fig. 10.1). In a first step, we capture constraints from natural sequences across evolution, including complex dependencies between positions, by learning the distribution of amino acid sequences for each protein using an expressive deep generative model, a variational autoencoder (VAE) (Rezende et al., 2014; Kingma & Welling, 2013). VAEs have been successful in learning complex high-dimensional distributions across multiple domains including protein function prediction (Methods). For each human protein of interest, a Bayesian VAE is trained on a multiple sequence alignment retrieved by searching 250 million protein sequences in UniRef (Suzek et al., 2015) (Methods, Supplementary Table 1). After training on evolutionary sequences, we estimate the relative likelihood of each single amino acid variant with respect to the wild type – what we call the "evolutionary index" – by sampling from the approximate posterior distribution learned by the VAE. We performed a thorough architecture and hyperparameter search to ensure stability and performance across proteins, and demonstrate its superiority over prior methods (Extended Data Fig. 2). When comparing this evolutionary index against clinical labels, the value that separates pathogenic from benign labels is remarkably consistent across proteins (Extended Data Fig. 3a), suggesting we may use unsupervised methods to infer pathogenicity. Therefore, in a second step, rather than using (semi-)supervised learning to map scores to label categories, we fit a two-component global-local mixture of Gaussian Mixture Models (GMM) on the distributions of evolutionary indices for all single amino acid variants across proteins (Methods, Extended Data Fig. 3b). The output of this process is both the EVE score – a continuous pathogenicity score defined over the interval [0,1], with zero being most benign and one being most pathogenic – and class assignments. For these assignments, we use the predictive entropy of the GMM as a measure of classification uncertainty, and bin variants into one of three categories: Benign, Uncertain or Pathogenic (Methods).

We apply EVE to a set of 3,219 human genes that have been associated with disease in ClinVar (Landrum & Kattman, 2018) (Methods). Our model is predictive of clinical significance for all labeled variants across all genes (average AUC 0.91, Fig. 10.2 b, Supplementary Table 2) including 60 "clinically actionable" genes (Kalia et al., 2017) (average AUC 0.92 Extended Data Fig.4a). Furthermore, the performance of EVE is robust to the number of labels per protein (Fig. 10.2 b) suggesting generalizability to genes with less (or no) annotation, as we would expect from an unsupervised approach.

EVE outperforms all supervised and unsupervised methods at predicting known clinical labels (Fig. 10.2 c, x-axis, Supplementary Table 3). This is despite a large fraction of these labels being used in training the top performing methods, as well as, in some cases, being used extensively in defining labels. As a second benchmark which avoids some of these circularities, we compare the model predictions against 40k experimentally measured variants across 10 proteins (Methods). Since these experiments are, in principle, independent of the ClinVar labeling process, we expect this benchmark to provide a less biased estimate of performance, albeit for a comparatively smaller number of proteins. On this benchmark EVE outperforms all methods (Fig. 10.2 c, y-axis, Supplementary Table 4) including meta-predictors (Extended Data Fig. 4b, Supplementary Note 2, Methods).

Since the consequence of variant classification significantly varies from one gene to another, an important feature of our method is the ability to assign a degree of uncertainty to the prediction, allowing a trade-off between predicted accuracy and coverage of variants. Setting aside an increasing number of variants as “Uncertain” enables to reach higher accuracy over the variants that we do classify as Pathogenic or Benign. For instance, excluding the 25% of most uncertain variants results in an accuracy of 90% for Pathogenic and Benign classifications (Fig. 10.2 b, Supplementary Table 2). In practice, we envision researchers deciding on specific trade-offs on a gene-by-gene and use case basis.

10.2 EVE as accurate as experimental prediction

We now ask whether our computational predictions are as accurate as experimental predictions. For the five genes with a large number of high-quality labels in ClinVar (BRCA1, P53, PTEN, MSH2, SCN5A) the overall performance of EVE at predicting clinical significance is as good as, or better than, that of the DMS experiments that were specifically designed to predict pathogenicity (Findlay et al., 2018; Glazer et al., 2020; Giacomelli et al., 2018; Mighell et al., 2018; Jia et al., 2021) (Fig. 10.3, Extended Data Fig. 5, Methods). For instance, for P53, EVE predicts near perfect separation of benign and pathogenic variants for the whole protein in contrast to the experimental predictions (Giacomelli et al., 2018) that are weaker in the tetramer domain (from position 300 to end). For SCN5A (associated with Brugada syndrome34 and long QT syndrome35) R814Q is predicted by EVE as pathogenic but is seen as near wild-type in the experiments from Glazer et al. Glazer et al. (2020) suggesting that evolutionary data contains information about gain of function and supports the known genetics35. EVE also has marginally better performance than experiments on a larger set of genes that have fewer high-quality labels (Methods, Extended Data Fig. 6, Supplementary Table 5, 6).

Since EVE and MAVEs are independent sources of evidence, comparison of their results may help evaluate the clinical labels themselves. Across MSH2, PTEN and P53, 23 of the 27 variants (85%) where the EVE score disagrees with ClinVar, MAVE experimental data supports the EVE classification. Both EVE and experiments support a benign score for variants R337H and R337C in P53, S554N/T, D660G and I774V in MSH2, and 15 variants in PTEN score where ClinVar has Pathogenic labels. Similarly, both EVE and experimental assays support a pathogenic clinical effect where ClinVar has Benign labels for G759E and E198G in MSH2 (the pathogenic assignment of the latter is further supported by new experimental data (Bouvet et al., 2019)). An obvious caveat where concordance between functional assay prediction and EVE may be misleading is the case of functional RNA, e.g. splice variation (Jia et al., 2021).

Taken together, our analysis shows that EVE prediction performs as well as predictions from high-throughput experiments, suggesting it may be beneficial to focus experimental efforts on genes where EVE does not perform well (Supplementary Table 7).

10.3 Predictions for 36 million variants

We provide both continuous EVE scores and class assignments for the 36 million single amino acid variants across the 3,219 disease associated genes. Of these variants, 1.3M have been observed in at least one human to date (UK Biobank (Van Hout et al., 2020) and gnomAD (Karczewski et al., 2020); Methods), but only about 3% have any clinical interpretation in ClinVar (Fig. 10.4 a, left). The EVE class assignments, after dropping the 25% most uncertain variants to keep accuracy at 90%, provide an interpretation for 27 million variants in total and over 800k (64%) of the variants seen to date in humans (Fig. 10.4 a, middle, Methods).

The continuous scores for all single amino acid variants provide a complementary picture to that of class assignments. The distribution of EVE scores within proteins highlights clusters of high pathogenicity, following trends that might be expected by functional importance, such as hydrophobic cores, and ligand-binding and active sites. For instance, many variants with high EVE scores in the SCN4A/SCN1B ion channel complex (PDB 6agf38) lie at the complex interface, line the SCN4A pore and the hydrophobic core of SCN1B (Fig. 10.4 b,c). For the mismatch DNA repair complex MSH2/MSH6 (associated with Lynch syndrome39 and 20% of sporadic cancers (Peltomaki, 2003)), EVE pathogenic signals are strong for variants proximal to the bound ADP and DNA (2o8b41) where clinical labels are sparse (yet observed in the population) (Fig. 10.4 d).

10.4 Combining EVE with other evidence

EVE provides a single source of evidence, making it ideal for combining with other, orthogonal sources of evidence (as is typically performed by expert panels, e.g. ClinGen (Richards et al., 2015)). To illustrate this, we combine our model class assignments with population data from gnomAD (Karczewski et al., 2020) and other forms of existing evidence. This results in 256k variants with no previous clinical interpretation for potential reclassification, and another 539 variants that contradict current ClinVar status for which we find independent supporting evidence. Examples of the latter include MSH2 variants described above, and P53 variant R337Q (Methods, Fig. 10.4 a, Supplementary Table 8).

Being unsupervised also opens the door to a more refined approach where the strength of evidence provided by the model may be allowed to vary on a gene-by-gene basis, in close analogy with recommendations for functional assays42. This offers a clear advantage over supervised methods. For example, if we consider the 1,000 genes with at least 10 labels for validation, a supervised method (using a 90% train, 10% test, random split) leaves only 50 proteins on which to test (Extended Data Fig. 7).

10.5 Discussion

It has long been appreciated that looking at the patterns of sequence conservation across species can yield insights into the consequences of variation within a species (Lewontin; Kreitman, 1983) including insights into human variants and disease association45. By bringing together recent developments in machine learning with the rapidly increasing amount of sequencing data from diverse organisms, we can extract more precise statements than previously realized and on a sufficiently large scale to be able to impact our sum knowledge of the clinical significance of variants. All data, results and code are available at or linked from evemodel.org (Extended Data Fig. 8) which will be regularly updated with new genes.

We have demonstrated that deep generative models trained on sequence alignments alone achieve state-of-the-art performance on variant classification and do so while avoiding the issues that typically impact supervised methods. This not only leads to better generalization guarantees but also provides a source of evidence which is independent and complementary to other large-scale efforts (e.g. population data from biobanks) and yields an order magnitude gain of scope when validating on a gene-by-gene basis. Although we do not know precisely how the constraints learnt from the sequences relate to disease, we observe performance on a par with functional assay in predicting pathogenicity. This suggests that expert panels could subject our method to similar scrutiny for classification as experiments like MAVEs.

The primary advantage of our approach over experimental approaches is significant gain in scope at a negligible fraction of the cost. An appealing prospect is that our method may be useful in guiding future experimental efforts, essentially acting as a means of identifying which variants and which genes would be most informative to probe (Supplementary Table 7).

There are important challenges in assessing missense variants that are not covered in this report. First, is the heterogeneity of disease presentation; we know that different variants of the same gene, and even the same variant, can lead to different disease severity or even different diseases, aspects that will be masked by the use of simple discrete pathology categories. Although we expect that the continuous EVE score, as opposed to the discrete classifications, may be useful to predict disease severity, this remains speculative and does not account for entirely different disease presentations from variants in the same gene. Secondly, this current work does not explicitly address the effect of combinations of variants. Since humans have on average 12% of their genes with two or more variants compared to a reference genome, albeit not necessarily in the same chromosome, this will be an important consideration. For the “ACMG actionable genes” there are 21k distinct pairs of variants in the same gene, occurring 1.5 million times (UK BioBank (Van Hout et al., 2020); Methods, Extended Data Fig. 9, Supplementary Table 9).

We conclude with a remark on biodiversity. Our analysis is one small but unusually direct demonstration of how the diversity of life on Earth benefits human health. Our models make use of data from over 140k organisms. Of these we identified 17k organisms which are on the International Union for Conservation of Nature’s (IUCN) Red List of Threatened Species (Iucn) including 1,301 classified as vulnerable, 1,148 endangered, 548 critically endangered, 10 extinct in the wild and 21 extinct organisms. The progressive disappearance of species is a threat to the diversity upon which this work is built.

Chapter 11

Conclusion

Artificial intelligence stands to be among the most impactful human technologies. Efficient automation of intellectual work will unlock an immense acceleration in scientific discovery, medicine, and economic output. Today, the most promising effort towards compelling artificially intelligent systems is the scaling of large neural networks and the data that trains them.

As part of a meta-trend in machine learning research that has seen methods that scale flourish (Sutton, 2019), the work in this thesis strives to contribute insights and methods that aide practitioners as they continue to pursue ever larger models and ever more data.

I hope that these methods – or work derived from them – increases the speed at which we are able to acquire compelling artificial intelligence, and I hope that this technology contributes to a world where all illnesses are treatable to great effect, where all humans have access to fulfilment and an abundance of resources, and where all beings are able to meaningfully contribute to the flourishing of intelligent life in our universe.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. URL `http://download.tensorflow.org/paper/whitepaper2015.pdf`.

Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *CoRR*, abs/1711.08856, 2017. URL `http://arxiv.org/abs/1711.08856`.

Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

Görkem Algan and Ilkay Ulusoy. Image classification with deep learning in the presence of noisy labels: A survey. *Knowledge-Based Systems*, 215:106771, 2021.

Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46, 1992.

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

Sercan O Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv:1908.07442*, 2019.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization, 2019.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 3084–3092, 2013.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

R Harald Baayen. *Word frequency distributions*, volume 18. Springer Science & Business Media, 2001.

Mohammad Babaeizadeh, Paris Smaragdis, and Roy H Campbell. Noiseout: A simple way to prune neural networks. *arXiv preprint arXiv:1611.06211*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019a.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019b.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *International Joint Conference on Neural Networks*, volume 2, 1991.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

John B Biggs. The role of metalearning in study processes. *British journal of educational psychology*, 55, 1985.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/blundell15.html`.

Zalán Borsos, Mojmír Mutný, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *arXiv preprint arXiv:2006.03875*, 2020.

Léon Bottou and Yann LeCun. Large scale online learning. *Advances in neural information processing systems*, 16:217–224, 2004.

D. Bouvet et al. Methylation tolerance-based functional assay to assess variants of unknown significance in the mlh1 and msh2 genes and identify patients with lynch syndrome. *Gastroenterology*, 157:421–431, 2019. doi: 10.1053/j.gastro.2019.03.071.

Leo Breiman. Random forests. *Machine learning*, 45, 2001.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020a.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020b.

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

Thang Bui, Daniel Hernández-Lobato, Jose Hernandez-Lobato, Yingzhen Li, and Richard Turner. Deep gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, 2016.

ByungSoo Ko, H. Kim, Kyo-Joong Oh, and H. Choi. Controlled dropout: A different approach to using dropout on deep neural network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 358–362, 2017.

Y. Cao et al. The chinamap analytics of deep whole genome sequences in 10,588 individuals. *Cell Res*, 30:717–731, 2020. doi: 10.1038/s41422-020-0322-9.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013a.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013b.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. The best of both worlds: Combining recent advances in neural machine translation. In *Annual Meeting of the Association for Computational Linguistics*, volume 56, 2018.

Pengfei Chen, Ben Ben Liao, Guangyong Chen, and Shengyu Zhang. Understanding and utilizing deep neural networks trained with noisy labels. In *International Conference on Machine Learning*, pp. 1062–1070. PMLR, 2019.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Knowledge Discovery and Data Mining*, volume 22, 2016.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv:1904.10509*, 2019.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. Notes from the AI frontier: Insights from hundreds of use cases, 2018.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *International Conference on Learning Representations*, 2020.

Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. Pruning neural networks: is it time to nip it in the bud? *arXiv preprint arXiv:1810.04622*, 2018.

Zhenwen Dai, Andreas Damianou, Javier González, and Neil Lawrence. Variational auto-encoded deep gaussian processes. In *International Conference on Learning Representations*, 2016.

Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, volume 16, 2013.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pp. 4860–4874, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL `https://arxiv.org/abs/2010.11929`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021a.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021b.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

D. Esposito et al. Mavedb: an open-source platform to distribute and interpret data from multiplexed assays of variant effect. *Genome Biol*, 20:223, 2019. doi: 10.1186/s13059-019-1845-6.

Utku Evci, Fabian Pedregosa, Aidan N. Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *CoRR*, abs/1906.10732, 2019. URL `http://arxiv.org/abs/1906.10732`.

Sebastian Farquhar, Yarin Gal, and Tom Rainforth. On statistical bias in active learning: How and when to fix it. *arXiv preprint arXiv:2101.11665*, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021. URL `https://arxiv.org/abs/2101.03961`.

G. M. Findlay et al. Accurate classification of brca1 variants with saturation genome editing. *Nature*, 562:217, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, volume 34, 2017.

Evelyn Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale, 2019a.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *CoRR*, abs/1903.01611, 2019b. URL `http://arxiv.org/abs/1903.01611`.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *CoRR*, abs/1912.05671, 2019c. URL `http://arxiv.org/abs/1912.05671`.

Jonathan Frazer, Pascal Notin, Mafalda Dias, Aidan Gomez, Joseph K Min, Kelly Brock, Yarin Gal, and Debora S Marks. Disease variant prediction with deep generative models of evolutionary data. *Nature*, 599(7883):91–95, 2021.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pp. 3584–3593, 2017.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019.

Saurabh Garg, Yifan Wu, Sivaraman Balakrishnan, and Zachary Lipton. A unified view of label shift estimation. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, volume 35, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv:1807.01622*, 2018b.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Dropblock: A regularization method for convolutional networks, 2018.

A. O. Giacomelli et al. Mutational processes shape the landscape of tp53 mutations in human cancer. *Nat Genet*, 50:1381–1387, 2018. doi: 10.1038/s41588-018-0204-y.

A. M. Glazer et al. High-throughput reclassification of scn5a variants. *Am J Hum Genet*, 107:111–123, 2020. doi: 10.1016/j.ajhg.2020.05.015.

Aidan N Gomez, Ivan Zhang, Siddhartha Rao Kamalakara, Divyam Madaan, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Learning sparse networks using targeted dropout. *arXiv preprint arXiv:1905.13678*, 2019.

Aidan N Gomez, Oscar Key, Kuba Perlin, Stephen Gou, Nick Frosst, Jeff Dean, and Yarin Gal. Interlocking backpropagation: Improving depthwise model-parallelism. *Journal of Machine Learning Research*, 23(171):1–28, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017.

D. F. Gudbjartsson et al. Large-scale whole-genome sequencing of the icelandic population. *Nat Genet*, 47:435–444, 2015. doi: 10.1038/ng.3247.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585, 2020.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016b.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv:1912.12180*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

T. A. Hopf et al. Sequence co-evolution gives 3d contacts and structures of protein complexes. *Elife*, 3, 2014. doi: 10.7554/eLife.03430.

T. A. Hopf et al. Mutation effects predicted from sequence co-variation. *Nat Biotechnol*, 35:128–135, 2017. doi: 10.1038/nbt.3769.

Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018a.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018b. URL `http://arxiv.org/abs/1811.06965`.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

Michael Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. Lietransformer: Equivariant self-attention for lie groups. *arXiv:2012.10885*, 2020.

Google Inc. Kaggle. *https://www.kaggle.com/*, 2021.

IUCN. The Iucn. *Red List of Threatened Species.* <. URL `https://www.iucnredlist.org`.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions, 2014.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv:2103.03206*, 2021.

X. Jia et al. Massively parallel functional testing of msh2 missense variants conferring lynch syndrome risk. *Am J Hum Genet*, 108:163–175, 2021. doi: 10.1016/j.ajhg.2020.12.003.

Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018.

Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. *Advances in Neural Information Processing Systems*, 31:7265–7275, 2018.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016a.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016b.

S. S. Kalia et al. Recommendations for reporting of secondary findings in clinical exome and genome sequencing, 2016 update (acmg sf v2.0): a policy statement of the American college of medical genetics and genomics. *Genet Med*, 19:249–255, 2017. doi: 10.1038/gim.2016.190.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N Gomez. Exploring low rank training of deep neural networks. *arXiv preprint arXiv:2209.13569*, 2022.

K. J. Karczewski et al. The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature*, 581:434–443, 2020. doi: 10.1038/s41586-020-2308-7.

Angelos Katharopoulos and François Fleuret. Biased importance sampling for deep neural network training. *arXiv preprint arXiv:1706.00043*, 2017.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, volume 37, 2020.

Kenji Kawaguchi and Haihao Lu. Ordered sgd: A new stochastic optimization framework for empirical risk minimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 669–679. PMLR, 2020.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, volume 30, 2017.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Mikhail Khodak, Neil A. Tenenholtz, Lester Mackey, and Nicolo Fusi. Initialization and regularization of factorized neural layers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=KTlJT1nof6d`.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. *arXiv preprint arXiv:2012.10630*, 2020.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *arXiv preprint arXiv:1906.08158*, 2019.

Andreas Kirsch, Tom Rainforth, and Yarin Gal. Active learning under pool set distribution shift and noisy data. *CoRR*, abs/2106.11719, 2021. URL `https://arxiv.org/abs/2106.11719`.

B. Ko, H. Kim, and H. Choi. Controlled dropout: A different dropout for improving training speed on deep neural network. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 972–977, 2017.

Aran Komatsuzaki. One epoch is all you need. *arXiv preprint arXiv:1906.06669*, 2019.

Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better?, 2018.

Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems*, 34: 28742–28756, 2021.

M. Kreitman. Nucleotide polymorphism at the alcohol dehydrogenase locus of drosophila melanogaster. *Nature*, 304:412–417, 1983. doi: 10.1038/304412a0.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks, 2019.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350, 2015.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

M. J. Landrum and B. L. Kattman. Clinvar at five years: Delivering on the promise. *Hum Mutat*, 39:1623–1630, 2018. doi: 10.1002/humu.23641.

A. Lapedes, B. Giraud, and C. Jarzynski. Using sequence alignments to predict protein structure and stability with high accuracy. arxiv. preprint, 2012.

Guillaume Leclerc, Manasi Vartak, Raul Castro Fernandez, Tim Kraska, and Samuel Madden. Smallify: Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 1998b.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*, 2, 2010.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, volume 36, 2019a.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. *CoRR*, abs/1906.06307, 2019b. URL `http://arxiv.org/abs/1906.06307`.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

R. C. Lewontin. The genetic basis of evolutionary change. (columbia university press. 1974.

Mingkun Li and Ishwar K Sethi. Confidence-based active learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1251–1261, 2006.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018a.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018b.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through *l_0* regularization. *arXiv preprint arXiv:1712.01312*, 2017.

Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pp. 3033–3045, 2019.

D. S. Marks et al. Protein 3d structure computed from evolutionary sequence variation. *PLOS ONE*, 6, 2011. doi: 10.1371/journal.pone.0028766.

Charles H. Martin and Michael W. Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning, 2018.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

T. L. Mighell, S. Evans-Dutson, and B. J. A O'Roak. Saturation mutagenesis approach to understanding pten lipid phosphatase activity and genotype-phenotype relationships. *The American Journal of Human Genetics*, 102:943–955, 2018.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pp. 15630–15649. PMLR, 2022.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. 2016.

Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip HS Torr, and Yarin Gal. Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty. *arXiv preprint arXiv:2102.11582*, 2021.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. Do transformer modifications transfer across implementations and applications? *arXiv:2102.11972*, 2021.

Pascal Notin, Aidan N Gomez, Joanna Yoo, and Yarin Gal. Improving compute efficacy frontiers with sliceout. *arXiv preprint arXiv:2007.10909*, 2020.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, volume 35, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

P. Peltomaki. Role of dna mismatch repair defects in the pathogenesis of human cancer. *J Clin Oncol*, 21:1174–1179, 2003. doi: 10.1200/JCO.2003.04.060.

Geoff Pleiss, Tianyi Zhang, Ethan Elenberg, and Kilian Q Weinberger. Identifying mislabeled data using the area under the margin ranking. *Advances in Neural Information Processing Systems*, 33:17044–17056, 2020.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019b.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. *bioRxiv*, 2021.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp; Data Mining*, pp. 3505–3506, 2020.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, 2003.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pp. 4334–4343. PMLR, 2018.

B. Reva, Y. Antipin, and C. Sander. Predicting the functional impact of protein mutations: application to cancer genomics. *Nucleic Acids Res*, 39, 2011. doi: 10.1093/nar/gkr407.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

S. Richards et al. Standards and guidelines for the interpretation of sequence variants: a joint consensus recommendation of the American college of medical genetics and genomics and the association for molecular pathology. *Genet Med*, 17:405–424, 2015. doi: 10.1038/gim.2015.30.

Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv:2104.10972*, 2021.

Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pp. 1746–1754, 2014.

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2014.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, 2017.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5, 1990.

Burr Settles. Active learning literature survey. 2009.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost, 2018.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pp. 10414–10423, 2018.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In Peter Auer and Ron Meir (eds.), *Learning Theory*, pp. 545–560, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31892-7.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014a. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014b.

C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

X. Sun, X. Ren, S. Ma, and H. Wang. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. *ArXiv e-prints*, June 2017.

Richard Sutton. The bitter lesson. In *http://incompleteideas.net/IncIdeas/BitterLesson.html*, 2019. Accessed: 2019-05-23.

B. E. Suzek, Y. Wang, H. Huang, P. B. McGarvey, and C. H. UniRef clusters Wu. a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31:926–932, 2015. doi: 10.1093/bioinformatics/btu739.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization, 2016.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv:2009.06732*, 2020.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Yonglong Tian, Olivier J Henaff, and Aaron van den Oord. Divide and contrast: Self-supervised learning from uncurated data. *arXiv preprint arXiv:2105.08054*, 2021.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks, 2014.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2020.

M. Trenkmann. Putting genetic variants to a fitness test. *Nat Rev Genet*, 19:667, 2018. doi: 10.1038/s41576-018-0056-4.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016a.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016b.

C. V. Van Hout et al. Exome sequencing and characterization of 49,960 individuals in the uk biobank. *Nature*, 586:749–756, 2020. doi: 10.1038/s41586-020-2853-0.

R. Vaser, S. Adusumalli, S. N. Leng, M. Sikic, and P. C. Sift Ng. missense predictions for genomes. *Nat Protoc*, 11:1–9, 2016. doi: 10.1038/nprot.2015.123.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017c.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. *Vol. 1: MT Researchers' Track*, pp. 193, 2018.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost, 2021.

Huan Wang, Qiming Zhang, Yuehai Wang, and Haoji Hu. Structured probabilistic pruning for convolutional neural network acceleration. *arXiv preprint arXiv:1709.06994*, 2017.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *International Conference on Artificial Intelligence and Statistics*, volume 19, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. In *CVPR*, 2015.

Chhavi Yadav and Léon Bottou. Cold case: The lost mnist digits. In *Advances in Neural Information Processing Systems 32*, 2019.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31, 2018.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. pp. 67–76, 2017. doi: 10.1109/CVPR.2017.15.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Jiong Zhang, Hsiang-Fu Yu, and Inderjit S Dhillon. Autoassist: A framework to accelerate training of deep neural networks. *arXiv preprint arXiv:1905.03381*, 2019.

A. Zunino, S. Adel Bargal, P. Morerio, J. Zhang, S. Sclaroff, and V. Murino. Excitation Dropout: Encouraging Plasticity in Deep Neural Networks. *ArXiv e-prints*, May 2018.