Robust Networks: Neural Networks Robust to Quantization Noise and Analog Computation Noise Based on Natural Gradient

by

Pradyumna Kadambi

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2019 by the
Graduate Supervisory Committee:

Visar Berisha, Chair
Gautam Dasarathy
Jae-Sun Seo
Yu Cao

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

Deep neural networks (DNNs) have had tremendous success in a variety of statistical learning applications due to their vast expressive power. Most applications run DNNs on the cloud on parallelized architectures. There is a need for for efficient DNN inference on edge with low precision hardware and analog accelerators. To make trained models more robust for this setting, quantization and analog compute noise are modeled as weight space perturbations to DNNs and an information theoretic regularization scheme is used to penalize the KL-divergence between perturbed and unperturbed models. This regularizer has similarities to both natural gradient descent and knowledge distillation, but has the advantage of explicitly promoting the network to find a broader minimum that is robust to weight space perturbations. In addition to the proposed regularization, KL-divergence is directly minimized using knowledge distillation. Initial validation on FashionMNIST and CIFAR10 shows that the information theoretic regularizer and knowledge distillation outperform existing quantization schemes based on the straight through estimator or L_2 constrained quantization.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Visar Berisha for his guidance, knowledge, and patience. His support, feedback, and encouragement have been indispensable. I am lucky to have had a supervisor who was so invested in helping me become a better researcher. I am grateful for the opportunity to work on this research.

I would like to thank the rest of my thesis committee: Dr. Gautam Dasarathy, Dr. Jae-Sun Seo, and Dr. Yu Cao, for their insightful comments and questions.

I would also like to thank Paul Whatmough and Chu Zhou of ARM Inc for providing me a summer internship opportunity and Dr. Jae-Sun Seo for recommending me for the position.

To my labmates, thank you for your friendship, your help and inputs, and the and for the fun times outside the lab.

I would like to thank my friends and family for their love through the ups and downs. Finally, I must express my very profound gratitude to my parents and for providing me with support and continuous encouragement throughout my years of study. This would not have been possible without them. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Deep learning has emerged as a useful tool for solving problems in a variety of machine learning domains such as computer vision, speech recognition, natural language processing, recommender systems and anomaly detection. The excellent performance offered by deep neural networks (DNNs) comes at a high computational cost since DNNs typically have many layers and can have millions of parameters. In fact, the increases in the state of the art performance have come jointly with an increase in the model size. GoogLeNet (Szegedy *et al.*, 2015) achieved 74.8% Top-1 accuracy on ImageNet with 6 million parameters, while ResNeXt-101 managed 79.6% Top-1 accuracy with nearly 45 million parameters.

A significant amount of deep learning inference is expected to be performed at the edge, which is challenging since edge devices are restricted by small power budgets and have limited computational resources. Applications like health monitoring, autonomous driving, and smart home devices must also meet real-time constraints and ensure data privacy. This makes it challenging to always offload inference to the cloud. Therefore, it is crucial to develop efficient hardware for DNN inference on mobile devices. Interest in efficient DNN architectures also extends to the datacenter where power density is anticipated to rise if an increasing share of workload comes from DNN inference.

Several methods have been proposed for efficient deep learning hardware. Standard methods improve the efficiency of DNNs by applying quantization, pruning, and compression techniques to the model with an effort to retain the original model's high performance. By reducing the number of bits used to represent each weight, reducing the number of parameters, and even reducing the number of layers, the memory and power consumption can be reduced, and the latency can be improved. This efficient model can be further optimized if implemented on custom hardware such as an ASIC or FPGA. Specifically, analog hardware using non-volatile memory (NVM) cells for performing DNN inference *in-memory* are an active area of research, since they can drastically reduce the energy cost of multiply-accumulate operations (at the cost of inherently noisy analog computations) which drives deep learning inference cost.

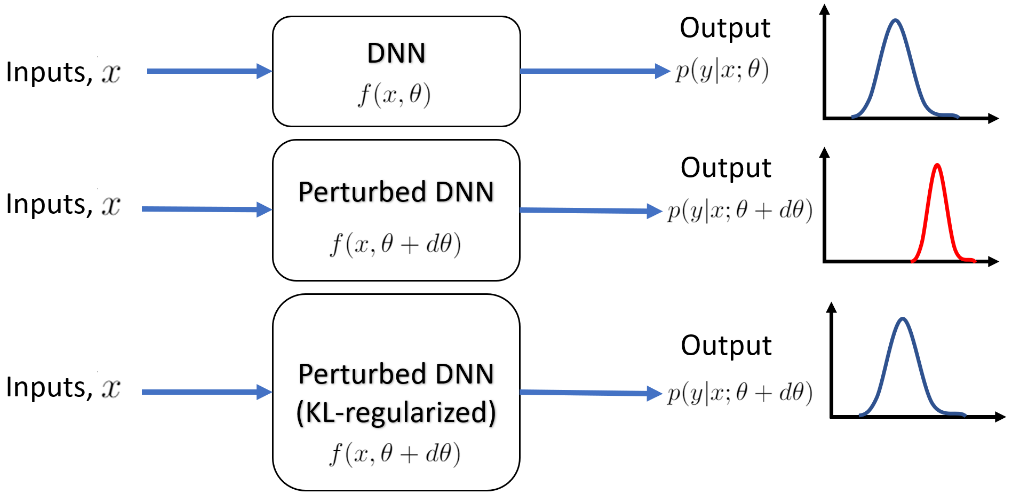


Figure 1.1: Upon a perturbation to the parameter space (like quantization or analog hardware noise), the output distribution of a DNN model can change significantly. Regularizing the KL-divergence between the output of the perturbed model and original model leaves the output distribution unchanged after perturbation.

Both quantization and analog hardware noise can be considered as a more general problem: ensuring that a DNN is robust to perturbation of its weights. By regularizing the KL-divergence of the output of the network against a perturbed version of the network, the model's output distribution will not change under perturbation (Figure 1.1). We develop two methods of regularizing the KL-divergence. One method approximates the KL-divergence using Fisher information, a quantity that ranks parameter importance. The connection between this method, second order optimization, and natural gradient descent is explored. Knowledge distillation is shown as another method that is equivalent to regularizing the KL-divergence. This regularization approach is applied in the context of efficient DNN hardware. Quantization of weights and mixed-signal noise from analog hardware are both treated as weight perturbations, and the network is regularized for robustness against these perturbations. On a Lenet-5 network trained on FashionMNIST, and ResNet-18 trained on CIFAR-10, distillation and Fisher regularization are demonstrated to outperform standard quantization methods. The efficacy of these methods for providing robustness to mixed-signal in-memory compute noise is then evaluated.

The document is structured as follows. In Chapter 2, an overview is provided of related work in quantization of DNNs, analog hardware for DNNs, and second order optimization methods. In Chapter 3, the Fisher regularizer is derived, knowledge distillation is introduced, and modifications to SGD and ADAM are outlined that include the regularization method. Chapter 3 also describes the noise model used for an NVM accelerator. Chapter 4 covers the experimental validation. Chapter 5 contains concluding remarks and suggestions for future work.

Chapter 2

LITERATURE REVIEW

This section explores several avenues of related work. First, we shall develop background in DNN quantization and compression methods. To complete the discussion on hardware for neural networks, we cover non-volatile memory based accelerators for DNNs. To motivate our algorithmic contribution, we will examine natural gradient descent and second order methods for DNNs.

2.1 Neural Network Quantization and Compression

Prior to the advent of modern deep learning methods, several works studied quantization of neural networks. Dundar and Rose (1995) build on the work of (Xie and Jabri, 1992) and evaluate the performance of neural networks used as analog-to-digital converters (ADCs), sine wave generators, and classifiers. The networks were directly trained with quantized weights such that after each weight update, the parameters were quantized. They found that training could often freeze, since the gradients would not be large enough to change the value of the weight since the weight would return to its previous value after quantization. They suggest training based on randomly perturbing the network, or by flipping the bit of the weights with the largest gradients in the direction of the gradient. Balzer *et al.* (1991) apply quantization to Boltzman Machines and study effective architectures for quantization.

A myriad of quantization methods have been proposed for modern deep learning systems (Sze *et al.*, 2017). An overview of related work is provided in Figure 2.1. Methods to quantize neural networks typically employ either *quantization-aware*

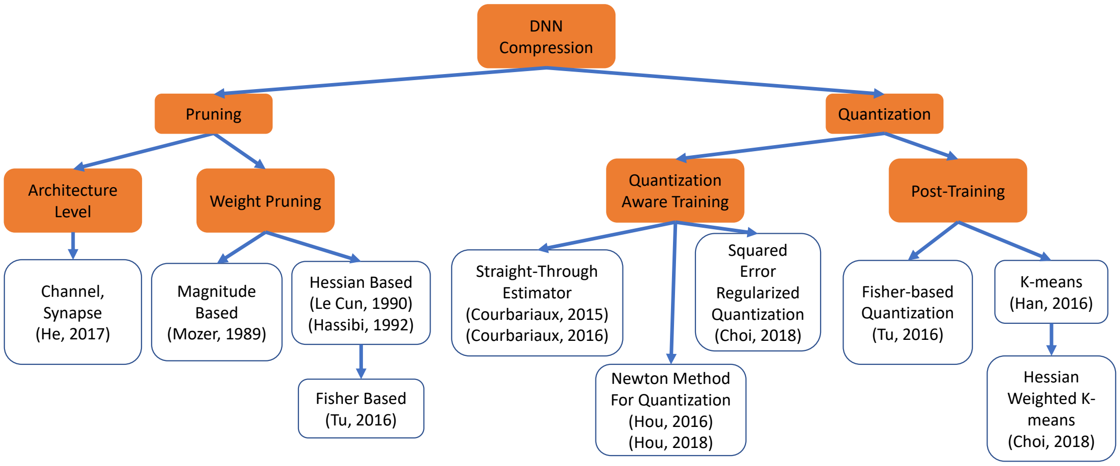

DNN Compression
Pruning
Quantization
Architecture Level
Weight Pruning
Quantization Aware Training
Post-Training
Channel, Synapse (He, 2017)
Magnitude Based (Mozer, 1989)
Hessian Based (Le Cun, 1990) (Hassibi, 1992)
Fisher Based (Tu, 2016)
Straight-Through Estimator (Courbariaux, 2015) (Courbariaux, 2016)
Newton Method For Quantization (Hou, 2016) (Hou, 2018)
Squared Error Regularized Quantization (Choi, 2018)
Fisher-based Quantization (Tu, 2016)
K-means (Han, 2016)
Hessian Weighted K-means (Choi, 2018)

Figure 2.1: An outline of work in model compression related to our method.

training or *post-training quantization*. As the name implies, post-training quantization begins with an already trained 32 bit floating point (FP32) model; weights and activations of the trained network are then quantized (Fig. 2.3). The benefit of this approach is that minimal training is required since pre-trained models are widely available. A small set of calibration data can be used to find the parameters for the quantizer. At INT8 precision, the accuracy of post-training quantization can nearly match FP32. However, reducing weights and activations to 4-bits and below with post-training quantization can significantly impact accuracy (Banner *et al.*, 2018).

In Han *et al.* (2015), a model compression pipeline known as Deep Compression is proposed. Initially, weights are pruned based on their magnitude based on a determined threshold. The network is then retrained to recover the loss in accuracy. Then k-means is employed for vector quantization of the weights, and each quantization bin is fine tuned by summing the gradient of all the weights that lie in that bin. This clustering is performed on a layer-by-layer basis. As a final step, Huffman Coding is performed to further reduce the memory footprint of the weights.

Quantization-Aware Training

In *quantization aware* training, the network is trained with quantized weights. Weights are quantized in the forward pass, but the backward pass is problematic, since the gradient of the quantization function is zero almost everywhere. In Bengio *et al.* (2013) the straight-through estimator(STE) is introduced for backpropogation through non-differentiable functions (such as the quantization function), and in Courbariaux *et al.* (2015) the STE is applied for training quantized networks. As the name implies, the STE replaces the gradient of the quantizer with identity so that gradients can flow unimpeded to the weights (Fig. 2.2). The STE is used in a multitude of training techniques for deep neural networks (Courbariaux *et al.*, 2015; Mishra *et al.*, 2017; Zhou *et al.*, 2016; Rastegari *et al.*, 2016; Polino *et al.*, 2018). Some works Courbariaux *et al.* (2015); Choi *et al.* (2016) use variants of the STE that enforce:

$$\frac{\partial Q(\theta)}{\partial \theta} = \mathbb{1}\{|\theta| \leq 1\} \tag{2.1}$$

(i.e. gradients for a parameter are cancelled when the FP32 value of that parameter has magnitude larger than 1). However, in practice, libraries like Tensorflow implement quantization-aware training by eschewing the condition (2.1) and equation simply apply pass through gradients for quantization such that $\frac{\partial Q(\theta)}{\partial \theta} = 1 \forall \theta$. Quantization-aware training directly minimizes the loss $L(Q(\theta))$, the loss with respect to the quantized parameters.

Aggressively quantizing networks to 4-bits and lower is of considerable interest as it can provide several benefits. Using a binary representation for weights and

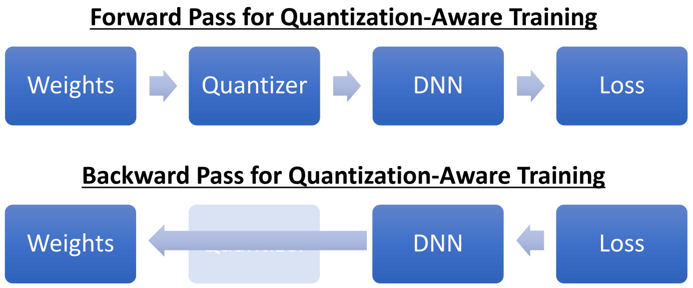

Forward Pass for Quantization-Aware Training
Weights
Quantizer
DNN
Loss
Backward Pass for Quantization-Aware Training
Weights
DNN
Loss

Figure 2.2: Quantization aware training using straight-through estimator. The quantization function is replaced with identity in the backward pass.

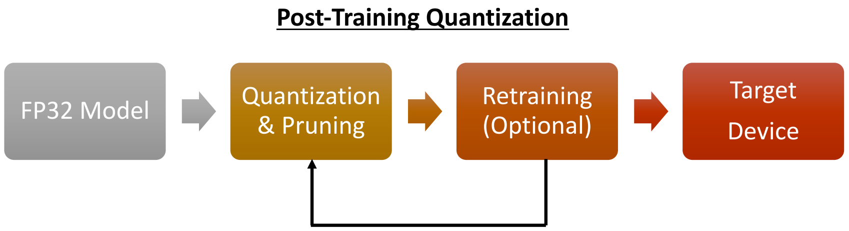


Figure 2.3: Post training quantization pipeline.

activations not only reduces memory footprint significantly, but also has the added benefit of transforming multiplications into simple XNOR operations. Weight access energy can also potentially decrease, since memory access cost is proportional to the size of the memory (Hubara *et al.*, 2017). BNN (Hubara *et al.*, 2018) introduces binary neural networks and demonstrates a binarized VGG network with minimal loss of accuracy on CIFAR-10. XNOR net (Rastegari *et al.*, 2016) introduces scaling factors for the binary weights and extended the study of binary networks to the ILSVRC2012 ImageNet dataset, achieving 56.8% using AlexNet. Wide reduced precision networks (Mishra *et al.*, 2017) shows that accuracy can be recovered for low precision networks (4b A 2b W, 2b A 2b W, 1b A 1b W) by simply increasing the number of feature maps in each layer. ResNet-34 Top-1 accuracy on ImageNet is improved from 60.54% to 72.38% at the cost of tripling the number of feature maps. Recently, Bi-real (Liu *et al.*, 2018) has demonstrated 62.2% top-1 accuracy on ImageNet for a binarized ResNet-34.

Loss Aware Quantization

The quantization error criterion, $\theta - Q(\theta))$, can also be incorporated into the loss function or included as an optimization constraint. This method has some similarities to our approach. In Hou *et al.* (2016) and Hou and Kwok (2018), binarization and ternarization of weights are introduced as optimization constraints. This is then solved via a proximal Newton method where the curvature matrix is estimated using an adaptive learning rate algorithm. They demonstrate their method on feedforward networks on CIFAR-10 and CIFAR-100, as well as on LSTMS on the PennTreebankd dataset. In Choi *et al.* (2018), quantized networks are learned via regularizing the mean squared quantization error: $MSQE(\theta) = \sum_i ||\theta_i - Q(\theta_i)||_2^2$. This results in a Top-1/Top-5 accuracy on ImageNet for 1b weights and activations of 41.1%/66.6% on AlexNet, and 38.9%/65.4% on ResNet-18.

Neural Architecture Search

A notable challenge that complex model compression methods face is that it is often difficult to use approaches like variable bit width for channels and weights, Huffman encoding of weights, insufficiently sparse weight representations, etc. in mobile hardware. Models are often pruned and deployed with INT8 or even FP32 precision to accommodate existing hardware. However, accelerators have been proposed to solve this issue (Lee *et al.*, 2018). An emerging area of research that can directly target existing hardware platforms is neural architecture search (NAS) (Zoph and Le, 2016). In NAS, the optimization algorithm finds not only a set of optimal weights, but also an optimal architecture. This search can be conducted by a reinforcement learning (RL) agent modeled as a markov decision process, or by Bayesian optimization.

To limit the immense computational burden (since the search space can be large, each candidate network may have to be trained, and the RL agent itself must be learned), NAS methods often start with a backbone architecture and allow learning of architecture-level parameters such as bit width, number of channels for a specific layer, number of layers, filter size, etc. The benefit of this approach is that the RL agent can find network architectures that are constrained by objectives that incorporate information about the power, latency, and memory available on the actual target device (Fedorov *et al.*, 2019). Therefore, device specific architectures can be learned. MobileNetV3 applies NAS to improve on Top-1 accuracy by 3.2% and reduce latency on mobile CPU by 20% (Howard *et al.*, 2019).

2.2 Analog Accelerators for Neural Networks

Von Neumann computing architectures separate data memory from the processing elements that perform the computation 2.5 This paradigm is at the heart of most modern CPUs and GPUs. In the context of neural networks, this requires that weights and activations for each layer must be read from a memory and fed to the processing element to produce the output. The most frequent operation in DNN inference is the multiply-accumulate operation that is executed for computing a matrix-vector product of a weight matrix by an input vector. For convolutional layers, an image to column (im2col) operation is applied prior to performing a matrix multiply. Each each $n \times n$ patch of the image - where n is the filter size - is extracted with appropriate stride, and transformed into a matrix of row vectors. The filter kernels are transformed into a matrix of column vectors. Thus a convolution is performed with a general matrix multiply (GEMM) operation which is itself comprised of many MACs.

MACs comprise an overwhelming majority of DNN energy use (Sze *et al.*, 2017). In fact, the energy required for each MAC operation is dominated by the cost of reading weights from memory, while the energy consumed by the processing element for performing the computation itself is relatively small (Hubara *et al.*, 2018). Additionally, the energy cost of accessing weight memory increases significantly as a function of the size of the memory. If the accuracy degradation is tolerable, this problem can be solved with aggressive quantization. In lieu of traditional digital hardware, this problem can be solved using a non-von Neumannn, *in-memory.* Analog in-memory compute for DNNs use the memory itself to perform computation (Binas *et al.*, 2016).

2.2.1 In-Memory Analog Computation

Consider Fig 2.4. Analog computation for performing MAC operations begins by first programming each cell in a non-volatile memory (NVM) array with a resistance value corresponding to weight from a layer. The memory can be resistive random access memory (ReRAM) (Song *et al.*, 2017), phase change memory (PCM) (Joshi *et al.*, 2019), or even magnetic random access memory (MRAM) (Patil *et al.*, 2019).

To feed inputs to the accelerator, the inputs are converted from digital values to analog voltages by a digital to analog converter. Then the currents that are excited across the resistances constitute multiplication, and the accumulation operation sums the currents by simply connecting wires. The output can easily be read by a analog to digital converter that converts this accumulated current to a voltage. Memory access cost is nullified. However, since computation is done in analog it is is prone to several sources of noise. The resistances of the NVM array (weights) cannot be exactly programmed. When deployed, the weights will be subject to thermal noise, and drift of weight value. The accuracy drop from this noise can be significant (10-30%) (Jain *et al.*, 2018). Shafiee *et al.* (2016) explore the design space for eDRAM based accelerators. Critically, it is shown that 60%-80% of the energy cost can come from the ADC/DAC. Also, they specify the control flow hardware that is needed in addition to the NVM compute cells to be able to realize a neural network using this type of accelerator.

Noise Model

Many works (Jain *et al.*, 2018; Agarwal *et al.*, 2016) have focused on modeling device level noise sources in detail such as wire resistance, non-ideal ADC/DAC,

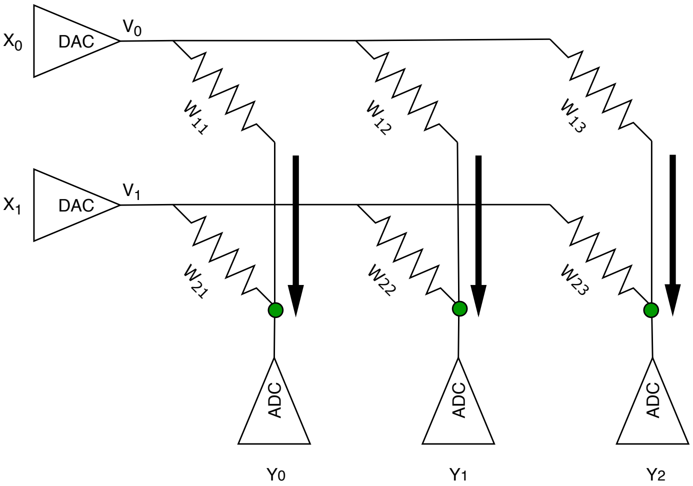


Figure 2.4: This figure above shows how a memroy array can compute the product of a vector $X \in \mathbb{R}^{2\times 1}$ and a weight matrix $W \in \mathbb{R}^{3x2}$ in the analog domain to produce the result $Y = WX$. Each memory cell has a resistance $R_{i,j}$ that can be programmed with the inverse corresponding weight value $\frac{1}{W_{i,j}}$. The MAC operation is performed by first converting the input vector is into an analog voltage on the wires. Multiplication is achieved via Ohm's law when the voltage is applied across the resistor to excite a current along the wire. This current is summed by simply connecting wires at the end of each column of the memory cell; an ADC computes the result by sensing the current.

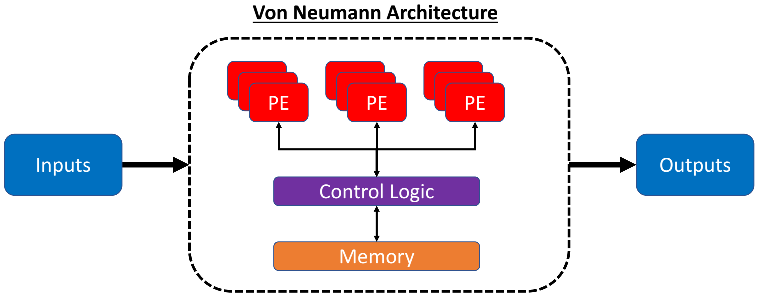


Figure 2.5: Traditional von-Neumann architecture requires the processing elements (PE) to query weights from the memory leading to increased latency and power consumption. GEMM and MAC operation energy cost in neural networks are dominated by weight access from memory.

current sneak paths, etc., but this is outside the scope of this work. Of more importance to this thesis are the attempts to model this analog noise as a noise

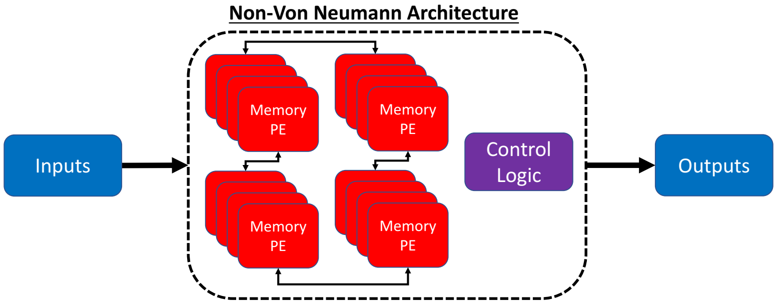


Figure 2.6: With an in-memory compute approach, the memory itself is the PE. This avoids costly data transfers. A network of PEs can be connected by the control logic to implement a desired network architecture.

perturbation to the weights. To this end several lumped models treat analog compute noise as additive Gaussian to the weights (Rekhi *et al.*, 2019; Joshi *et al.*, 2019). These approaches usually apply an STE-like proceduce for retraining the network for noise robustness. The weights are noised in the forward pass, but a noiseless version of the weights are updated. The noise in the weights is dependent on the type of memory device used, but it has been shown by Joshi *et al.* (2019) that a simple zero-mean additive Gaussian noise model is sufficient. Joshi *et al.* (2019) find that retraining with Gaussian noise provides robustness on actual hardware. For the parameters in a certain layer $\theta^{(l)}$ the noise is added layer-wise:

$$\theta^{(l)}_{noisy} = \theta^{(l)} + \mathcal{N}(0, \eta * |\max(\theta^{(l)}) - \min(\theta^{(l)})|) \tag{2.2}$$

Here η is a parameter that is intrinsic to the device that captures the amount of noise.

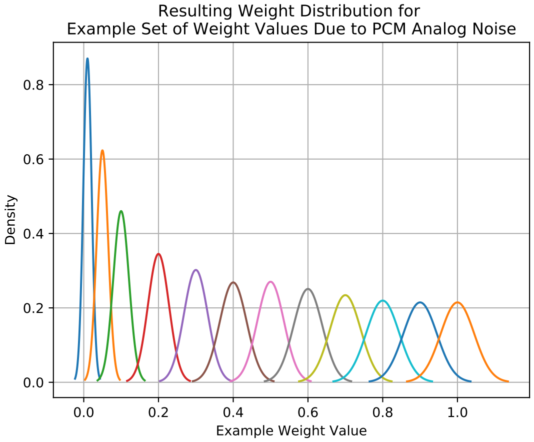


Figure 2.7: The figure above shows data from (Joshi *et al.*, 2019) on how noise in a phase change memory (PCM) accelerator affects weights. Despite the fact that it is clear that the standard deviation of the weights is dependent on the weight magnitude, a simple additive Gaussian with constant variance (Eq 3.89) can be an effective model of the noise.

2.3 Second Order Optimization for Deep Networks

First order methods such as stochastic gradient descent (SGD) and its variants minimize an objective function $L(\theta)$ by implicitly applying a local linear approximation around the current value of θ. Second order methods incorporate curvature information as well as information about the gradient by using a local quadratic model. An update Δ_θ is computed beginning with a local quadratic approximation of the loss about θ:

$$\underset{\Delta_\theta}{\operatorname{argmin}}\, L(\theta + \Delta_\theta) + \nabla L(\theta)^T \Delta_\theta + \frac{\lambda}{2} \Delta_\theta^T H \Delta_\theta \tag{2.3}$$

Differentiating with respect to Δ_θ equating to zero, and solving for Δ_θ yields:

$$\Delta_\theta = -\frac{1}{\lambda} H^{-1} \nabla L(\theta) \tag{2.4}$$

where H is the Hessian matrix. The update rule is:

$$\theta_t = \theta_{t-1} + \alpha \Delta_\theta \tag{2.5}$$

The Hessian for neural networks is a large $n \times n$ matrix where n is the number of parameters in the network. Computing the Hessian and its inverse is not feasible, and even computing approximates of the Hessian and its inverse can be computationally intensive. This problem can be solved by using “Hessian-free” optimization (Pearlmutter, 1994; Martens, 2010; Martens and Sutskever, 2011), applying a diagonal approximation on the Hessian (Kingma and Ba, 2014; Duchi *et al.*, 2011) or block diagonal approximations (Roux *et al.*, 2008; Martens and Grosse, 2015). In modern automatic differentiation packages (such as those available in Tensorflow and Pytorch) computing the product of the batch Hessian and a vector can be performed in the same time as computing the first derivatives, but these estimates can be noisy.

2.3.1 Natural Gradient Descent

In contrast to updating parameters in the direction of steepest descent of the loss in parameter space, *natural gradient descent* (NGD) chooses a descent direction that minimizes the steepest descent on the manifold of probability distributions (Amari, 1998). NGD suggests choosing updates for model parameters, θ, with an update, Δ_θ, such that KL-divergence $D_{KL}(p(y|x;\theta)||p(y|x;\theta+\Delta_\theta)) = c$. The model is updated ensuring that the KL-divergence between steps is constant:

$$\begin{gathered} \operatorname*{argmin}_{\Delta_\theta} L(\theta+\Delta_\theta) \\ s.t. D_{KL} = (p_\theta||p_{\theta+\Delta_\theta}) = c \end{gathered} \tag{2.6}$$

The loss can be rewritten using duality:

$$\operatorname*{argmin}_{\Delta_\theta} L(\theta+\Delta_\theta) + \frac{\lambda}{2}(D_{KL}(p_\theta||p_{\theta+\Delta_\theta}) - c) \tag{2.7}$$

As we will see in Section 3.3.2 the KL-divergence can be approximated with a second order Taylor series expansion using the *Fisher Information Matrix* (FIM), an information theoretic quantity describing how much information a parameter carries in modeling the output of the network. It is computed via the expected value of the Hessian of the loss for neural networks. Applying the FIM approximation

$$\underset{\Delta_\theta}{\operatorname{argmin}}\, L(\theta + \Delta_\theta) + \frac{\lambda}{2}\Delta_\theta^T F \Delta_\theta \tag{2.8}$$

The Fisher is popularly used as a curvature matrix in second order optimization methods (Pascanu and Bengio, 2013) and has been used for natural policy gradient in reinforcement learning (Kakade, 2002). Martens (2014) and Martens and Grosse (2015) outline how many second order methods often calculate estimates of the Fisher or its inverse due to the connection between the Fisher and the Hessian (Section 3.3.2).

Differentiating with respect to Δ_θ equating to zero, and solving for Δ_θ yields:

$$\Delta_\theta = -\frac{1}{\lambda}F^{-1}\nabla L(\theta) \tag{2.9}$$

NGD: Parameter Space versus Distribution Space

For a canonical example that demonstrates how NGD takes descent directions, consider the task of estimating the mean of two Gaussian random variables y and z with the same mean $\mu_y = \mu_z = 3$:

$$y \sim \mathcal{N}(\mu_y, \sigma_1^2) \tag{2.10}$$

$$z \sim \mathcal{N}(\mu_z, \sigma_2^2) \tag{2.11}$$

Assume that we begin with an initial estimate estimate $\mu_1 = 0$ for μ_y. The Euclidean distance between the true value and the estimated value is $||\mu_1 - \mu||_2$.

Then assume that the estimate $\mu_2 = \mu_1$ for μ_z, we can see that the KL divergence measures the overlap of the distributions:

$$D_{KL}\Big(\mathcal{N}(\mu_2, 2\sigma_2^2)||z\Big) < D_{KL}\Big(\mathcal{N}(\mu_1, \sigma_1^2)||y\Big) \tag{2.12}$$

But, the Euclidean metric simply measures difference between the means and is unable to incorporate the information about the overlap of the estimated and true distributions:

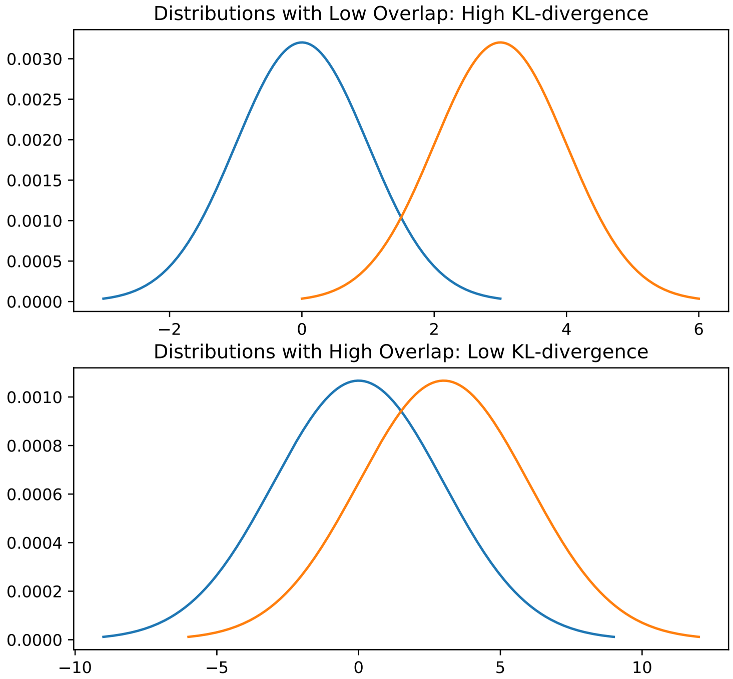
$$||\mu_1 - \mu_y||_2 = ||\mu_2 - \mu_z||_2 \tag{2.13}$$

Figure 2.8 shows how despite the fact that μ_1 and μ_2 are the same euclidean distance from the respective true values, in the KL sense, μ_1 is a much worse estimate.

2.3.2 Motivation for This Work: Second Order Information can be Used to Rank Parameter Importance

Second Order Methods and Network Compression

A central idea to several compression methods is applying some form of relative ranking of parameter importance. These methods are well established in the literature with methods such as Optimal Brain Damage (LeCun *et al.*, 1990), and Optimal Brain Surgeon (Hassibi *et al.*, 1994). Optimal Brain Damage relies on ranking the parameters in the network based on the diagonal of the Hessian of the loss function. If the curvature (Hessian) of loss with respect to a specific parameter is large, it indicates that the network is likely very sensitive to quantization or removal of that parameter. Inversely, if the curvature of the loss is small with respect to a parameter, the loss is not likely to change significantly if the parameter is removed or quantized. Optimal Brain Surgeon expands on this work by iterative


Distributions with Low Overlap: High KL-divergence

Distributions with High Overlap: Low KL-divergence

Figure 2.8: In contrast to simply updating the network based on the distance in parameter space, NGD can make clever updates by taking into account the difference in the output distributions of the model (blue) and the target (orange). In the figure above, both pairs of distributions have the same difference between the means, $||\mu_1 - \mu_y||_2 = ||\mu_2 - \mu_z||_2$ is the same. However, it is clear that in the bottom figure, the two distributions are more similar, and have higher overlap (lower KL-divergence). NGD could more aggressively update parameter in the top graph.

computation of the full Hessian matrix of the network, but suffers from the slowdown inherent to computing such a large matrix.

Hou *et al.* (2016) and Hou and Kwok (2018) use a proximal Newton method based on the diagonal of the Hessian for binarizing and ternarizing networks. Tu *et al.* (2016) rank the parameters of a neural network by their Fisher Information Matrix diagonal. They propose allocating a larger word length to parameters with high information and also prune parameters with low information. Beyond model compression, second order information is useful in transfer learning (Kirkpatrick *et al.*, 2017) and model-agnostic meta learning (Finn *et al.*, 2017).

Perturbation of Parameter Space. A network can be subject to noise such as quantization, noise from analog non-idealities due to a mixed-signal accelerator, and even perturbation from model transport in the case of transfer learning. A robust model will not significantly change its output distribution if the weights are perturbed.

Chapter 3

METHODS

In this section the regularization approach for model robustness is introduced. We begin by establishing notation and examine maximum likelihood estimation, Fisher information, and KL divergence. Subsequently, our Fisher Information based regularization scheme is developed. We demonstrate how Fisher Information can be computed for linear regression, logistic regression, softmax regression, and finally for neural networks. We propose modifications to SGD and ADAM that incorporate perturbation robustness for any arbitrary perturbation. An overview of the quantization methods and the noise model for analog accelerators is also provided.

3.1 Notation

To begin this section, we must establish some notation. Consider a model (such as a linear regression, logistic regression, DNN, etc) in the context of a classification problem. The output of this model is a distribution $p(y|x;\theta)$ where y is a random variable representing the output of this model, x is a random variable representing the input, and θ is a vector of the model parameters. Note that here, y itself is a function of x and θ, so we write $y(x,\theta)$. Define $d(\theta)$ to be some function or transformation of the model parameters. The respective output distributions of the original model and model with parameters modified by $d(\theta)$ are: $p(y|x;\theta)$ and $p(y|x;d(\theta))$, respectively. While the choice of $d(\theta)$ is arbitrary and depends on the application, in this work we choose $d(\theta)$ as the quantization function: $Q(\theta)$, or as a noising function $N(\theta)$ that subjects the parameters to analog mixed-signal noise.

3.2 Maximum Likelihood Estimation

In Maximum likelihood estimation (MLE), a given statistical model is assumed to have generated the observed data. The goal is to then select the parameters of the model, θ such that the probability of the model having produced the observations is maximized. Concretely, given a set of n independent, identically distributed (i.i.d.) observations $X_1, ..., X_n$, we have a *likelihood function*:

$$\begin{aligned} l(x;\theta) &= P(x_1 = X_1, ..., x_n = X_n) \\ &= p(x_1 = X_1;\theta)p(x_2 = X_2;\theta)...p(x_n = X_n;\theta) \\ &= \prod_{i=1}^{N} p(x_i;\theta) \end{aligned} \tag{3.1}$$

We wish find an optimal $\hat{\theta}$ that maximizes $l(\theta)$. To transform the product into a more tractable sum, the logarithm of the likelihood is taken.

$$L(\theta) = \log(\prod_{i=1}^{N} p(x_i;\theta)) = \sum_{i=1}^{N} \log p(x_i;\theta) \tag{3.2}$$

This is permissible since log is a monotonic function. Then:

$$\begin{aligned} \hat{\theta} &= \operatorname*{argmax}_{\theta} L(\theta) \\ &= \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \log p(x_i;\theta) \end{aligned} \tag{3.3}$$

The maximum likelihood estimate $\hat{\theta}$ is found by differentiating the log-likelihood and equating to zero:

$$0 = \frac{\partial}{\partial\theta} \sum_{i=1}^{N} \log p(x_i;\theta) \tag{3.4}$$

The Score Function

The derivative of the likelihood is known as the *score*. Note that the expected value of the score function is zero (under certain regularity conditions such as the assumption that the order of integration and differentiation can be reversed):

$$\begin{aligned}\mathbb{E}\left[\frac{\partial}{\partial\theta}\ \log p(x;\theta)\right] &= \int p(x;\theta)\frac{\partial}{\partial\theta}\log p(x;\theta)dx \\ &= \int p(x;\theta)\frac{\partial p(x;\theta)}{\partial\theta}\frac{1}{p(x;\theta)}dx \\ &= \frac{\partial}{\partial\theta}\int p(x;\theta)dx = 0\end{aligned} \tag{3.5}$$

3.2.1 Fisher Information

Given that the maximum likelihood estimate has been found $\hat{\theta}$, what is the uncertainty about this estimate? What is the variance of $\hat{\theta}$? Fisher Information is a useful way of answering this question. The Fisher information from random variable x about the parameter θ is defined as the covariance of the score:

$$F_\theta = \mathbb{E}_{p(x;\theta)}\left[\left(\frac{\partial}{\partial\theta}\log p(x;\theta)\right)^2\right] \tag{3.6}$$

Assuming certain regularity conditions, the Fisher can be rewritten as:

$$F_\theta = -\mathbb{E}_{p(x;\theta)}\left[\frac{\partial^2}{\partial\theta^2}\log p(x;\theta)\right] \tag{3.7}$$

Note that computing the expectation in 3.6 and 3.7 is intractable for most cases. We see in section (3.6) that even for a simple logistic classifier with Gaussian input data, computing the expectation in eq. (3.6) for the Fisher Information is not possible. Therefore, sample versions of the Fisher is used:

$$\hat{F} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial^2}{\partial\theta^2}\log p(x_i;\theta) \tag{3.8}$$

$$\hat{F} = -\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial}{\partial\theta}\log p(x_i;\theta)\right)^2 \tag{3.9}$$

It is known that near a minimum (3.9) approaches (3.6).

3.3 Regularization for Model Robustness

3.3.1 Model Robustness

If the parameters of the model, θ, are perturbed by some $d\theta$ (this perturbation could be due to a function such as $d(\theta)$), the output probability density of the model changes from $p(y|x;\theta)$ to $p(y|x;\theta + d\theta)$. In this section, we equivalently write p_θ and $p_{\theta+d\theta}$ for convenience.

For some sufficiently small ϵ, robust model satisfies the property:

$$D_{KL}(p_\theta || p_{\theta+d\theta}) = \epsilon \tag{3.10}$$

Regularizing the KL divergence between models for conferring robustness has been studied in the literature. Distillation is one such method that equivalent to minimizing the KL-divergence between the output PDF of some target model, and the model that is being trained. Distillation has been used for robustness to adversarial examples (Papernot *et al.*, 2016), and robustness against noise from analog computation (Anonymous, 2020). In fact, the standard cross entropy loss for training a classifier can be seen as roughly equivalent to minimizing the KL-divergence between model outputs and true labels. For a more in depth look at this topic refer to Section 3.8.

3.3.2 Fisher Approximation of KL-Divergence

As an alternative to directly applying the KL divergence as a regularizer between p_θ and $p_{\theta+d\theta}$, we propose using an approximation of the KL-divergence that has several interesting properties. We begin by expanding the KL-divergence and then applying Taylor expansion:

$$D_{KL}(p_\theta || p_{\theta+d\theta}) = \sum p_\theta \log \frac{p_\theta}{p_{\theta+d\theta}} \tag{3.11}$$

$$= \sum p_\theta \log p_\theta - \sum p_\theta \log p_{\theta+d\theta} \tag{3.12}$$

$$\tag{3.13}$$

In the following derivation kindly note the distinction between the partial differentiation $\partial\theta$ and the perturbation $d\theta$. The second term in the above equation can be expanded to:

$$
\begin{aligned}
D_{KL}(p_\theta, p_{\theta+d\theta}) &\approx \sum p_\theta \log p_\theta - \sum p_\theta \log p_{\theta+d\theta} && (3.14)\\
&= \sum p_\theta \log p_\theta - \Bigg[\sum p_\theta \log p_\theta + d\theta \sum \frac{\partial}{\partial\theta} p_\theta \log p_\theta + \sum p\theta \frac{\log p_\theta}{\partial\theta} \\
&\qquad + d\theta^2 \frac{\partial^2}{\partial\theta^2} \sum p_\theta \log p_\theta \\
&\qquad + \mathcal{H}.\mathcal{O}.\mathcal{T}\Bigg] && (3.15)\\
&= d\theta\Bigg[\frac{\partial}{\partial\theta}\sum p_\theta + d\theta^2 \frac{\partial^2}{\partial\theta^2}\sum p_\theta \log p_\theta\Bigg] && (3.16)\\
&= -d\theta^2 \sum \frac{\partial^2}{\partial\theta^2} p_\theta \log p_\theta && (3.17)\\
&= -d\theta^2\Bigg[\sum p_\theta\bigg(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial\theta}\frac{\partial}{\partial\theta}\bigg)\Bigg] && (3.18)\\
&= -d\theta^2\Bigg[\sum p_\theta\bigg(\frac{1}{p_\theta^2}\frac{\partial^2 p_\theta}{\partial\theta^2} + \frac{1}{p_\theta}\frac{\partial^2 p_\theta}{\partial\theta^2}\bigg)\Bigg] && (3.19)\\
&= -d\theta^2\Bigg[\sum \frac{\partial^2 p_\theta}{\partial\theta^2} + \sum p_\theta\bigg(\frac{1}{p_\theta^2}\frac{\partial^2 p_\theta}{\partial\theta^2}\bigg)\Bigg] && (3.20)\\
&= -d\theta^2\Bigg[\sum p_\theta\bigg(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial\theta}\bigg)^2\Bigg] && (3.21)\\
&= -d\theta^2\Bigg[\sum p_\theta\bigg(\frac{\partial}{\partial\theta}\log p_\theta\bigg)^2\Bigg] && (3.22)\\
&= -E\Bigg[\bigg(\frac{\partial}{\partial\theta}\log p_\theta\bigg)^2\Bigg] d\theta^2 \qquad = F d\theta^2 && (3.23)
\end{aligned}
$$

Where we have used that the Fisher Information F, is:

$$
F = -\mathbb{E}\Bigg[\bigg(\frac{\partial}{\partial\theta}\log p_\theta\bigg)^2\Bigg] = -\sum p_\theta \bigg(\frac{\partial}{\partial\theta}\log p_\theta\bigg)^2 \tag{3.24}
$$

Fisher Approximation of KL-divergence. A Fisher Information based approximation of the KL-divergence directly follows:

$$
D_{KL}(p_\theta || p_{\theta+d\theta}) \approx F d\theta^2 \tag{3.25}
$$

An Information Theoretic Regularizer

Beginning with the conditional output distribution of a model $p(y|x;\theta)$, a perturbation is applied to the weight space $p(y|x;\theta + d\theta)$. We attempt to answer the question: Can we force the output conditional distributions under a to be as "similar" as possible? A natural choice to measure distances between probability distributions is the KL-divergence. Therefore, the log-likelihood $L(\theta)$ is regularized with the KL-divergence between the model's output distribution $p(y|x;\theta)$ and the perturbed version of the mode's output distribution $p(y|x;\theta + d\theta)$:

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} D_{KL}\Big(p(y|x;\theta) || p(y|x;\theta + d\theta) \Big) \tag{3.26}$$

The multivariate version of the KL-divergence approximation in 3.25 can be used:

$$D_{KL}\Big(p(y|x;\theta) || p(y|x;\theta + d\theta) \Big) \approx \frac{1}{2} d\theta^T \mathbf{F} d\theta \tag{3.27}$$

As we have seen in 3.8 and 3.9 the Fisher can be approximated via the the Hessian of the loss, or by the square of the gradient. For computational reasons, we take a diagonal approximation to the Hessian. Therefore, our proposed cost function becomes:

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} d\theta^T \mathbf{F} d\theta \tag{3.28}$$

Connection to Constraint Based Parameter Robustness

Note that under the diagonal Fisher/Hessian approximation (see section 3.7), the regularized cost becomes:

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} \sum_i F_{ii} d\theta_i{}^2 = L(\theta) + \frac{\lambda}{2} \sum_i F_{ii} (\theta_i - Q(\theta_i))^2 \tag{3.29}$$

This has a rather elegant interpretation when transformed via Lagrangian Duality. For some constants b_i, we can rewrite (5):

$$\begin{aligned} \underset{\theta}{\operatorname{argmin}} \quad & L(\theta) \\ \text{subject to} \quad & d\theta_i \leq b_i, \ i = 1, \ldots, P. \end{aligned} \tag{3.30}$$

Starting with an information theoretic approach, we have arrived at a regularization scheme which constrains the perturbations on the weight space.

3.5 Fisher Information for Linear Regression

The following sections show how Fisher information can be computed analytically for some simple cases (linear regression with Gaussian data), but even a slightly more complicated case (logistic or softmax regression with Gaussian data) can lead to intractable computation for the Fisher. Therefore, sample estimates of the Fisher are discussed.

3.5.1 Linear Regression Notation

Consider the linear model:

$$t = y(x, \theta) + \epsilon \tag{3.31}$$

where x is a D-dimensional random variable representing the predictors, t is the observed response variable, θ are regression parameters, $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ is additive noise. Provided N observations X_i ($i \in [1, ..., n]$), the objective is to find a suitable set of parameters $\hat{\theta}$ for predicting t_i from X_i.

We have $x_n \in \mathbb{R}^D$, $X \in \mathbb{R}^{N \times D}$, and $X = \{x_1^T; ...; x_N^T\}$, $Y_i \in \mathbb{R}^{N \times 1}$, and target vector $T \in \mathbb{R}^{N \times 1}$. X is often referred to as the *design matrix* or data matrix. The parameters are $\theta \in \mathbb{R}^{D \times 1}$. Let us rewrite the parameter vector and design matrix such that $\theta = [\theta_0, \theta_1, ..., \theta_D]$ and let $X_i = [1, X_i]^T$ to simplify the analysis when including the bias term.

By definition:

$$\mathbb{E}[t] = y(x, \theta) \tag{3.32}$$

Also note that we define the variance of the noise as:

$$\sigma^2 = \beta^{-1} \tag{3.33}$$

3.5.2 Maximum Likelihood for Linear Regression

Assuming i.i.d. training data the goal is finding optimal parameter vector $\hat{\theta}$:

Since $t_i \sim \mathcal{N}(t|X_i^T\theta, \sigma^2)$.

$$p(t|X, \theta, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(t_i|Y(X_i, \theta), \sigma^2) \tag{3.34}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(T|X, \theta, \sigma) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} \mathcal{N}(t_i|Y(X_i, \theta), \sigma^2) \tag{3.35}$$

Then from 3.35:

$$p(T|X, \theta, \sigma) = \prod_{i=1}^{n} (2\pi\sigma^2)^{-}\frac{1}{2} \exp\{-\frac{1}{2\sigma^2}(t_i - \mathbb{E}[t_i])^2\} \tag{3.36}$$

The log-likelihood is:

$$\log p(T|X, \theta, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(t_i - y(X_i, \theta))^2 \tag{3.37}$$

Since the first term in 3.37 is constant w.r.t. θ, and because $\mathbb{E}[t_i] = X_i^T\theta$ see that Maximum Likelihood is equivalent to Least squares when finding $\hat{\theta}$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ \log p(T|X, \theta, \sigma^2) = \underset{\theta}{\operatorname{argmax}}\ -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(t_i - y(X_i, \theta))^2 \tag{3.38}$$

Thus:

$$\hat{\theta} = (X^T X)^{-1}(X^T T) \tag{3.39}$$

Similarly:

$$\hat{\beta}^{-1} = \hat{\sigma}^2 = \frac{1}{N}(T - X\theta)(T - X\theta) \tag{3.40}$$

3.5.3 Fisher Information from Likelihood

Fisher Information can be written in terms of the Hessian of the log-likelihood (assuming here that the variance σ^2 is known):

$$\mathbf{F}_\theta = -\mathbb{E}_{t|X,\theta,\sigma^2}\left[\mathbf{H}_\theta(\log p(t|X,\theta,\sigma^2))\right] \tag{3.41}$$

Analytical Calculation of Fisher Matrix: Define q, the distribution of the input $x_n \in \mathbb{R}^{D\times 1}$. The FIM is:

$$\mathbf{F}_\theta = \mathbb{E}_{x_n\sim q}\left[\mathbb{E}_{t\sim p(t|x_n,\theta,\sigma^2)}\left[\left(\frac{\partial \log p(t_n|x_n,\theta,\sigma^2)}{\partial\theta}\right)\left(\frac{\partial \log p(t_n|x_n,\theta,\sigma^2)}{\partial\theta}\right)^T\right]\right] \tag{3.42}$$

Then:

$$\mathbf{F}_\theta = \frac{1}{\sigma^2}\mathbb{E}_{x\sim q}[x_n x_n^T] \tag{3.43}$$

If we assume that $x_n \sim \mathcal{N}(0,I)$, i.e. $q(x) \sim \mathcal{N}(0,I)$,

$$\mathbf{F}_\theta = \mathbb{I} \tag{3.44}$$

Batched Fisher

If using SGD-like algorithms (such as Adam, Adagrad, SGD+momentum, etc), the algorithm will only have access to data from a specific batch. Therefore, Fisher information must be computed every batch. The batch estimate is:

$$\hat{\mathbf{F}}_\theta = \frac{1}{n_{batch}}\sum_{i=0}^{n_{batch}}\frac{1}{\sigma^2}X^T X \tag{3.45}$$

When the bias is included in the predictors: $X_i = [1, X_{i1}, X_{i2}, ..., X_{id}]$. The Hessian is:

$$\hat{\mathbf{H}} = N\hat{\mathbf{F}} =$$

$$-\frac{1}{\sigma^2}\begin{bmatrix} n & \sum_{i=1}^{n} X_{i1} & \sum_{i=1}^{n} X_{i2} & ... & \sum_{i=1}^{n} X_{id} \\ \sum_{i=1}^{n} X_{i1} & \sum_{i=1}^{n} X_{i1}^2 & \sum_{i=1}^{n} X_{i1}X_{i2} & ... & \sum_{i=1}^{n} X_{i1}X_{id} \\ ... & ... & ... & ... & \\ \sum_{i=1}^{n} X_{id} & \sum_{i=1}^{n} X_{id}X_{i1} & \sum_{i=1}^{n} X_{id}X_{i2} & ... & \sum_{i=1}^{n} X_{id}^2 \end{bmatrix}$$

$$H_{j,k} = -\frac{1}{\sigma^2}\sum_{i=1}^{N} X_{ij}X_{ik} \tag{3.46}$$

Therefore, the elements of the FIM are:

$$F_{j,k} = \frac{1}{N\sigma^2}\sum_{i=1}^{N} X_{ij}X_{ik} \tag{3.47}$$

3.6 Fisher Information for Softmax Regression

3.6.1 Logistic Regression Notation

Given a logistic function

$$y(x, \theta) = \sigma(\theta^T x) \tag{3.48}$$

Where $\sigma(a)$ is the sigmoid activation function. The input is a D-dimensional predictor $x \in \mathbb{R}^{D \times 1}$, and we observe n instances of x: $X \in \mathbb{R}^{N \times D}$, $X = \{x_1^T; ...; x_N^T\}$. The parameters are $\theta \in \mathbb{R}^{D \times K}$. The outputs are $y \in \mathbb{R}^{K \times N}$, and target vectors $T \in \mathbb{R}^{K \times N}$. Logistic/softmax regression estimates the class posteriors $p(C_k|x, \theta)$. When $k = 2$ we have the logistic regression case, where the class posterior probability is estimated using the sigmoid function:

$$p(C_1|x_n) = y_1(x_n, \theta) = \sigma(\theta^T x_n) = \frac{1}{1 + e^{-\theta^T x_n}} \tag{3.49}$$

For logistic regression $t_n \in \{0, 1\}$, and $p(t|X, \theta) = y_n$ represents the probability of observation x_n belonging to class 1. We write the following log-likelihood for logistic regression:

$$\log p(T|X, \theta) = \log \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n} = \sum_{n=1}^{N} t_n \log y_n + (1 - t_n) \log(1 - y_n) \tag{3.50}$$

3.6.2 Softmax Regression Notation

When considering the more general case of more than 2 classes ($k > 2$):

$$p(t_{nk} = 1|x_n) = p(C_k|x_n) = y_{nk}(x_n, \theta_k) = \frac{e^{-\theta_k^T x_n}}{\sum_j^K e^{-\theta_j^T x_n}} \tag{3.51}$$

where θ_k is the k^{th} column of the parameter vector $\theta \in \mathbb{R}^{D \times K}$. Note that $t_{n,k}$ is still a one hot vector indicating class membership and $y_{n,k}$ is still interpreted as the probability that observation x_n belongs to class k. We write the following log-likelihood for logistic regression:

$$\log\, p(T|x,\theta) = \log \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k|x_n,\theta)^{t_{nk}} = \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \log y_{nk}(x_n,\theta_k) \tag{3.52}$$

3.6.3 Fisher Information Matrix from Likelihood (Logistic Regression)

Beginning with eq. (3.50), we observe that there is no analytical solution to find θ that maximizes the expression.

Alternatively, consider the derivative of the log-likelihood:

$$\frac{\partial \log p(t_n|\theta)}{\partial \theta} = (t_n - y_n)x_n \tag{3.53}$$

Hessian is:

$$\mathbf{H} = \frac{\partial^2 \log p(t_n|\theta)}{\partial \theta^2} = -y_n(1-y_n)x_n^T x_n \tag{3.54}$$

Analytical Calculation of Fisher Matrix:

$$\mathbf{F}_\theta = \mathbb{E}_{x\sim q}\Bigg[\mathbb{E}_{t_n \sim p(t|x)}\Bigg[\bigg(\frac{\partial \log p(t_n|\theta)}{\partial \theta}\bigg)^T \bigg(\frac{\partial \log p(t_n|\theta)}{\partial \theta}\bigg)\Bigg]\Bigg] \tag{3.55}$$

$$\mathbf{F}_\theta = \mathbb{E}_{x\sim q}\Bigg[\bigg(\frac{\partial y_n}{\partial \theta}\bigg)^T \bigg(\frac{1}{y_n(1-y_n)}\bigg)\bigg(\frac{\partial y_n}{\partial \theta}\bigg)\Bigg] \tag{3.56}$$

$$\mathbf{F}_\theta = \mathbb{E}_{x\sim q}\Bigg[-y_n(1-y_n)xx^T\Bigg] \tag{3.57}$$

Since $x \sim q(x)$:

$$\mathbf{F}_\theta = \int_{-\infty}^{\infty} -y_n(1-y_n)xx^T q(x)dx \tag{3.58}$$

Even under the simple assumption about the distribution that generates x, namely that $q(x) \sim \mathcal{N}(0,1)$, the integral in eq (3.58) is not tractable (product of Gaussians and Sigmoids is not tractable). However, we may form a sample FIM from the observed data.

Sample Fisher Information Matrix (Logistic Regression):

Since:

$$\frac{\partial^2 \log p(t_n|\theta)}{\partial\theta^2} = -y_n(1-y_n)xx^T \tag{3.59}$$

and

$$\mathbf{F}_\theta = \mathbb{E}\bigg[-\frac{\partial^2 \log p(T|\theta)}{\partial\theta^2}\bigg] = \mathbb{E}\bigg[-y_n(1-y_n)xx^T\bigg] \tag{3.60}$$

$$\hat{\mathbf{F}}_\theta = -\frac{1}{N}\sum_{n=1}^{N} y_n(1-y_n)xx^T = -\frac{1}{N}X^T RX \tag{3.61}$$

Where $R = diag\{y_1(1-y_1), ..., y_N(1-y_N)\} =$
$diag\{\sigma(\theta^T x_1)(1-\sigma(\theta^T x_1)), ..., \sigma(\theta^T x_N)(1-\sigma(\theta^T x_N))\}$

3.6.4 Fisher Information Matrix from Likelihood (Softmax Regression)

We begin from the likelihood for softmax regression in eq (3.52). The derivative with respect to a column vector of parameters for a specific class θ_j is:

$$\frac{\partial \log p(t_k|x,\theta)}{\partial\theta_j} = (t_j - y_j)x \tag{3.62}$$

Each $D \times D$ block of the Hessian is (where I_{kj} is an element from the identity matrix).

$$\frac{\partial^2 \log p(t_k|x_n,\theta)}{\partial\theta_j\theta_k} = -y_k(I_{kj} - y_{nj})xx^T \tag{3.63}$$

A specific element $H_{k_{ii}}$of the Hessian is:

$$H_{k_{ii}} = \frac{\partial^2 \log p(t_k|x_n,\theta)}{\partial\theta_{ja}\theta_{kb}} = -y_k(I_{kj} - y_{nj})x_a x_b \tag{3.64}$$

The Hessian is a $KD \times KD$ matrix, but we only consider the main diagonal:

$$H_{k_{ii}} = -y_k(1-y_k)x_i^2 \tag{3.65}$$

where i ranges from $1, 2, ..., D$, and k ranges from $1, 2, ..., K$.

Again, note that similar to eq (3.58) an expectation of the above equation will not be tractable since the expectation of the product of a sigmoid and Gaussian is not tractable. Therefore we use the sample Fisher Matrix.

Sample Fisher Information Matrix (Softmax Regression):

The sample FIM, $\hat{\mathbf{F}}$, is a $KD \times KD$ matrix:

$$\hat{F}_{k_{ii}} = \frac{1}{N}\sum_{n=1}^{N} -y_{nk}(1-y_{nk})x_{ni}^2 \tag{3.66}$$

$$\hat{F}_{k_{ii}} = \frac{1}{N}\sum_{n=1}^{N} -y_{nk}(1-y_{nk})x_{ni}^2 \tag{3.67}$$

The FIM for softmax regression is made up of K blocks of the form

$$\mathbf{F}_k = X^T R_k X \tag{3.68}$$

where $R_k = diag\{y_{1k}(1-y_{1k}), ..., y_{Nk}(1-y_{Nk})\}$

3.7 Fisher Information for Neural Networks

3.7.1 Observed Information Matrix

For classification problems, neural networks are typically trained with the cross entropy loss:

$$L(\theta) = \mathbb{E}\Bigg[\log p(y|x;\theta)\Bigg] \tag{3.69}$$

This expectation is typically computed via a sample estimate over a batch

$$L_{batch}(\theta) = \frac{1}{N}\sum_{i=1}^{N_{batch}} \log p(y_i|x_i;\theta) \tag{3.70}$$

We can then notice that the expectation in (3.69) is equivalent to the log-likelihood. Thus, the Fisher information for a neural network can be computed with (3.7) or (3.9).

3.7.2 Fisher Computation from Squared Gradient

A common method of estimating the expected value in (3.6) is to use a method similar to the ADAM optimization algorithm (Algorithm 1). ADAM keeps exponentially decaying moving averages of the squared gradient (line 4 of the algorithm). This means that the expectation calculated by ADAM is calculated over values of θ_t. If stochastic gradient descent (SGD) is used instead of ADAM, the same method can be used for estimating the Fisher information. Thus, we conveniently obtain the diagonal of Fisher by setting $\hat{F} = v_t$.

The Empirical Fisher Approximation

It is important to note one caveat - deep learning libraries traditionally provide *batched gradients*: $\frac{1}{N}\sum \frac{\partial}{\partial\theta}\log p(y_i|x_i;\theta)$. In practice, this *Empirical Fisher* must be carefully applied as there can be significant mismatch between the Empirical Fisher and the proper estimate of the Fisher (Kunstner *et al.*, 2019).

Concretely, to compute the expected value of the squared gradient over a batch (following Eq. 3.6) we should have:

$$\hat{F} = \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \left(\frac{\partial}{\partial\theta} \log p(y_i|x_i;\theta) \right)^2 \tag{3.71}$$

but, the Empirical Fisher approximation is:

$$\hat{F}_{empirical} = \left(\frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \frac{\partial}{\partial\theta} \log p(y_i|x_i;\theta) \right)^2 \tag{3.72}$$

3.7.3 Fisher Regularizer Gradient Update Rule

In this section, we derive the update rule with the Fisher regularization term and provide the details of modified versions of SGD and ADAM (Algorithm 3, 2) that incorporate the regularizer.

Algorithm 1 ADAM optimization algorithm, excerpt from (Kingma and Ba, 2014)

```
Require: step size $\alpha$, exponential decay rates $\beta_1 = .9$, $\beta_2 = .999$, $\epsilon = 1E{-}8$
Given initial parameter vector $\boldsymbol{\theta}_0$, initial first and second moment vectors $\boldsymbol{m}_0 \leftarrow 0$ and $\boldsymbol{v}_0 \leftarrow 0$ and initial timestep $t = 0$
    $\odot$ denotes an element-wise product between two vectors.
While $\boldsymbol{\theta}_t$ not converged do:
    1. $t \leftarrow t + 1$
    2. $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}_{t-1})$ (Get gradients)
    3. $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ (compute first moment)
    4. $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$ (compute second moment [Fisher diagonal])
    5. $\hat{\boldsymbol{m}}_{\boldsymbol{t}} \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$
    6. $\hat{\mathbf{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$
    7. $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \hat{\boldsymbol{m}}_{\boldsymbol{t}}/(\sqrt{\hat{\mathbf{v}}_{\mathbf{t}}} + \epsilon)$
end while
return $\boldsymbol{\theta}_t$
```

The regularized loss is:

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} d\theta^T \mathbf{F} d\theta \tag{3.73}$$

with the gradient:

$$\frac{\partial \tilde{L}(\theta)}{\partial \theta} = \nabla_\theta L(\theta) + \lambda \frac{\partial^T d\theta}{\partial \theta} \mathbf{F} d\theta \tag{3.74}$$

We will assume that the Fisher information, $\mathbf{F}$, is a constant with respect to the parameters. Therefore, there is no need to use the chain rule to compute $\frac{\partial \mathbf{F}}{\partial \theta}$. This assumption is also extended to the derivative of the perturbation, $\frac{\partial d\theta}{\partial \theta}$. For our application, the perturbation is defined with respect to a quantized version of the parameters $Q(\theta)$:

$$d\theta = \theta - Q(\theta) \tag{3.75}$$

Since the straight-through estimator assumption is $\frac{\partial Q(\theta)}{\partial \theta} = 1$, the derivative of the perturbation (and therefore the regularizer) would become:

$$\frac{\partial d\theta}{\partial \theta} = 1 - \frac{\partial Q(\theta)}{\partial \theta} = 0 \tag{3.76}$$

Thus, we take $\frac{\partial d\theta}{\partial \theta} = 1$.

Under these assumptions, the gradient of the regularized loss is:

$$\frac{\partial \tilde{L}(\theta)}{\partial \theta} = \nabla_\theta L(\theta) + \lambda \mathbf{F} d\theta \tag{3.77}$$

and the weight update rule becomes:

$$\theta_t \leftarrow \theta_t - \alpha\Bigg(\nabla_\theta L(\theta_{t-1}) + \lambda \mathbf{F} d\theta_{t-1}\Bigg) \tag{3.78}$$

R-SGD: Proposed Modified SGD for Training a Robust Neural Network

Algorithm 2 Robust SGD with Momentum

```
Require: learning rate $\alpha$, momentum $\beta_1 = .9$, damping factor $\eta$, exponential averaging parameter $\beta_2 = .999$, $\gamma$ a constant controlling the strength of the regularizer

Given: initial parameter vector $\boldsymbol{\theta}_0$, initial first and second moment vectors $\boldsymbol{m}_0 \leftarrow 0$ and $\boldsymbol{v}_0 \leftarrow 0$ and initial timestep $t = 0$, $\eta$ and $\gamma$ to be set by the user ($\eta = 0$ by default), $f$ a function that perturbs the parameters
While $\boldsymbol{\theta}_t$ not converged do:
    1. $t \leftarrow t + 1$
    2. $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}_{t-1})$ (Get gradients)
    3. $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \eta)\boldsymbol{g}_t$ (compute momentum)
    4. $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$ (compute Fisher diagonal)
    5. $\boldsymbol{p}_t \leftarrow \theta_t - f(\theta_t)$ (compute perturbation)
    7. $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha\Bigg(\boldsymbol{m_t} + \gamma \boldsymbol{v_t} \odot \boldsymbol{p_t}\Bigg)$ (regularized update)
end while
return $\boldsymbol{\theta}_t$
```

R-ADAM: Proposed Modified ADAM Training a Robust Neural Network

Algorithm 3 Robust ADAM with Momentum

```
Require: learning rate $\alpha$, momentum $\beta_1 = .9$, damping factor $\eta$, exponential averaging parameter $\beta_2 = .999$, $\gamma$ a constant controlling the strength of the regularizer

Given: initial parameter vector $\boldsymbol{\theta}_0$, initial first and second moment vectors $\boldsymbol{m}_0 \leftarrow 0$ and $\boldsymbol{v}_0 \leftarrow 0$ and initial timestep $t = 0$, $\eta$ and $\gamma$ to be set by the user ($\eta = 0$ by default), $f$ a function that perturbs the parameters
While $\boldsymbol{\theta}_t$ not converged do:
    1. $t \leftarrow t + 1$
    2. $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}_{t-1})$ (Get gradients)
    3. $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ (compute first moment)
    4. $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$ (compute second moment [Fisher diagonal])
    5. $\hat{\boldsymbol{m}}_{\boldsymbol{t}} \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$
    6. $\hat{\mathrm{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$
    7. $\boldsymbol{p}_t \leftarrow \theta_t - f(\theta_t)$ (compute perturbation)
    8. $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha\Big(\hat{\boldsymbol{m}}_{\boldsymbol{t}}/(\sqrt{\hat{\mathbf{v}}_{\mathbf{t}}} + \epsilon) + \gamma \boldsymbol{v}_{\boldsymbol{t}} \odot \boldsymbol{p}_{\boldsymbol{t}}\Big)$ (regularized update)
end while
return $\boldsymbol{\theta}_t$
```

In accordance with Loshchilov and Hutter (2017), the regularizer and weight decay (if present) are added directly in the weight update step after computing the momentum and first/second moments (Step 8) rather than to the gradients, $\mathbf{g}_{\mathbf{t}}$, before computing the first and second moments (adding in step 2). This is done for both R-SGD and R-ADAM. The authors of that work demonstrate the subtle but important fact that $L2$ regularization and weight decay are not identical. Namely, $L2$ regularization functions by adding the wight decay to the gradients, $\mathbf{g_t}$, rather than directly during the weight update. This is problematic especially for R-ADAM, because if the Fisher regularizer is added to the gradients in Step 2, it would subsequently be scaled by the square root of the inverse of the Fisher in Step 8. The update would therefore be incorrect. They also show that show that hyperparameter tuning is far more forgiving under this modification as the resulting accuracy is more stable to changes in hyperparameters.

3.8 Direct KL-Divergence Regularization of Weight Perturbations with Distillation

In Distillation (Hinton *et al.*, 2015), a *student model* with reduced capacity is trained using the logits of a *teacher model.* The teacher is typically a large, high capacity model that achieves high performance on the target dataset. The student model typically has reduced capacity which may be due to effects like pruning, quantization, reduced model size (fewer layers, parameters), noisy weights, or missing data. Distillation trains the student with the outputs of the teacher using a temperature softmax:

$$q_i^t = \frac{\exp(z_i^t/T)}{\sum_k \exp(z_k^t/T)} \tag{3.79}$$

The logits of the teacher z_i^t are input to the temperature softmax and produce, q_i^t, the estimated class probabilities. The resultant distribution over the outputs can be “smoother” (Table 3.8), and can allow the student to learn relationships between classes from the teacher. The typically used softmax is recovered by setting $T = 1$.

The distillation loss is the cross entropy between the temperature softmax probabilities of the teacher and the student:

Class	$T=1$	$T=2$	$T=4$	$T=8$	$T=12$
Pedestrian	0.95	0.76	0.56	0.44	0.41
Biker	0.04	0.16	0.26	0.3	0.31
Dog	0.01	0.08	0.18	0.26	0.28

Table 3.1: Toy example to demonstrate effect of increasing temperature in softmax (Eq. 3.79) for a 3 class problem. Increasing the temperature smoothens the distribution over the logits. Suppose that the training example we are considering has label "pedestrian", but also has a bike in the image. When T is correctly chosen this ambiguity is more accurately reflected in the label. Thus, the labels can provide the student model with more information about the relationship between classes.

$$L_{dist}(q^t, q^s; \theta.T) = H(q^t, q^s) \tag{3.80}$$

Therefore, the student tries to match the outputs of the teacher. L_{dist} can be used as a regularizer in addition to the typical loss $L(\theta)$. Therefore:

$$\tilde{L}_{dist}(\theta) = L(\theta) + \eta L_{dist}(\theta, T) \tag{3.81}$$

3.8.1 Distillation as KL-Divergence Minimization

Minimizing L_{dist} can be seen as equivalent to minimizing the KL-divergence between the student and the teacher models:

$$\begin{aligned} L_{dist}(q^t, q^s; \theta, T) &= H(q^t, q^s) \\ &= H(q^t) + D_{KL}(q^t || q^s) \\ &= H(p_t(y|x;\theta_t)) + D_{KL}\Bigg(p_t(y|x;\theta_t) || p_s(y|x;\theta_s) \Bigg) \end{aligned} \tag{3.82}$$

Here we have used that p_t and p_s are the output distribution of the teacher and student respectively. Notice that in our case, the teacher is a pre-trained model that is not updated. This means that $H(p_t(y|x;\theta_t)) = c$ for some constant c, and is therefore not targeted by the minimization. When the student model is a quantized version of the teacher, $\theta_s = Q(\theta_t)$. Also, we can set $d\theta_t = \theta_t - Q(\theta_t)$:

$$\begin{aligned}
L_{dist}(q^t, q^s; \theta, T) &= H(p_t(y|x;\theta_t)) + D_{KL}\Bigg(p_t(y|x;\theta_t)||p_s(y|x;\theta_s)\Bigg) \\
&= D_{KL}\Bigg(p_t(y|x;\theta_t)||p_s(y|x;\theta_s)\Bigg) + c \\
&= D_{KL}\Bigg(p_t(y|x;\theta_t)||p_s(y|x;Q(\theta_t))\Bigg) + c \\
&= D_{KL}\Bigg(p_{\theta_t}||p_{\theta_t+d\theta}\Bigg)
\end{aligned} \tag{3.83}$$

Therefore, if the student and teacher model share the same architecture, but the student is a quantized or noised version of the student (a version of the teacher with weights perturbed by $d\theta$), Distillation minimizes the KL-divergence between the model and the perturbed version of the model.

Distillation in Practice

In practice, the teacher model is an FP32 model and the student model has been independently trained for quantization using straight-through estimator, so strictly speaking θ_s is not perturbed version of θ_t. However, the $D_{KL}(p_{\theta_t}||p_{\theta_s})$ is minimized.

3.9 Quantization

Quantization is a nonlinear operation that maps a continuous input to a discrete set of values. A quantization function $q(\theta) : \mathbb{R} \rightarrow \mathbb{Z}$ for a set of levels $\{L_1, ..., L_Q\}$ given a set of thresholds $A_1, ..., A_{Q-1}$ is:

$$q(\theta) = \begin{cases} L_1 & -\infty < \theta \leq A_1 \\ L_2 & A_1 < \theta \leq A_2 \\ ... & ... \\ L_Q & A_{Q-1} < \theta \leq \infty \end{cases}$$

The levels and thresholds can be selected based on a prior assumption of the weight distribution (such as Gaussian quantization), selecting levels based on clustering, or even learning the levels during the training. In this work weights and activations are quantized using uniform quantization between two levels $[-a, b]$. Thus, for an n-bit quantizer, there are $k = 2^n$ centroids where
$L_k = -a + \frac{k(b-a)}{2^n}$. The $k - 1$ thresholds are selected uniformly in between the levels.

Quantization of Weights. To determine a and b for weights for n-bit quantization, we begin with a pretrained network and perform k-means on the resultant weights with $|\mathcal{S}| = 2^n$ levels. Subsequently we choose $a = \min(\mathcal{S})$ and $b = \max(\mathcal{S})$. This method is used for Lenet-5.

For ResNet-18, we again begin with a pretrained network and select $a_l = \min(\theta^{(l)})$ and $b_l = \max(\theta^{(l)})$ where $\theta^{(l)}$ is the weight of a specific layer. Binary quantization of ResNet-18 weights is done by selecting $[-1, 1]$ as quantizer levels.

Quantization of activations. To quantize activations $X^{(l)}$, one of two approaches is used. During training, the minimum and maximum of the activations are learned with a momentum parameter such that:

$$\tilde{a}_l = (1 - \alpha) * \min(X_l) + \alpha * \tilde{a}_l$$
$$\tilde{b}_l = (1 - \alpha) * \max(X_l) + \alpha * \tilde{b}_l \tag{3.84}$$

during training, and

$$a_l = \tilde{a}_l$$
$$b_l = \tilde{b}_l \tag{3.85}$$

for inference. A value of $\alpha = 0.9$ is used during training.

Perturbation due to Quantization

The perturbation $d\theta$ from quantization is easily calculated as:

$$d\theta^{(l)} = \theta^{(l)} - Q(\theta^{(l)}) \tag{3.86}$$

3.10 Noise Model for In-Memory Compute

The noise model used is similar to (Joshi *et al.*, 2019). Beginning with the parameters of a specific layer $\theta^{(l)}$,

$$\theta^{(l)}_{noisy} = \theta^{(l)} + \mathcal{N}(0, \eta * |\max(\theta^{(l)}) - \min(\theta^{(l)})|) \tag{3.87}$$

When $\theta^{(l)}$ is quantized, this is equivalent to scaling the noise with respect to the quantization bins if $a^{(l)}$ and $b^{(l)}$ are the learned minimum and maximum value of the quantization levels respectively:

$$\theta^{(l)}_{noisy} = \theta^{(l)} + \mathcal{N}(0, \eta * |b^{(l)} - a^{(l)}|) \tag{3.88}$$

η is a parameter that is dependent on the specific device or memory technology that is used. This parameter captures the inherent noise of the NVM memory. If the parameters are quantized, the noise is added on top of the quantized parameters, not the full precision parameters, since the quantized parmeters would actually be implemented in the hardware.

Perturbation Due to Mixed-Signal Accelerator Noise

It is obvious that the perturbation term is the additive noise itself:

$$z^{(l)} \sim \mathcal{N}(0, \eta * |b^{(l)} - a^{(l)}|) \tag{3.89}$$

$$d\theta^{(l)} = z^{(l)} \tag{3.90}$$

Chapter 4

RESULTS AND DISCUSSION

4.1 Comparison of Diagonal Approximation and Hessian Vector Product for Estimating Fisher

Despite the prevalence of using diagonal approximations for the curvature matrix, it is known that neural network loss landscapes have a significant number of important off diagonal elements (Martens and Grosse, 2015). So in this section, the diagonal approximation of the Fisher is compared with using a Hessian-vector product (HVP). An HVP allows extremely efficient computation of $Fd\theta$, the gradient of the regularizer, exactly without any approximation. To compare the HVP and diagonal approximation the following experiment was conducted:

Procedure - Comparison of Regularization with Diagonal Approximation vs HVP on a Two Layer MLP on MNIST

1. Train multilayer perceptron (MLP) with two hidden layers and ReLU activations on the MNIST dataset.
   (Architecture: $784 - 512FC - 512FC - 10$)

2. Compute the test accuracy after applying a perturbation to the weights $\mathcal{N}(\mu, \sigma^2 = .005)$ (repeating 100 times to ensure enough draws from the noise distribution).

3. Sweep μ in the range $[.01, .1]$

4. Characterize the test accuracy after regularizing the network with the Fisher update rule (eq. 3.78) using both the HVP and diagonal approximation.

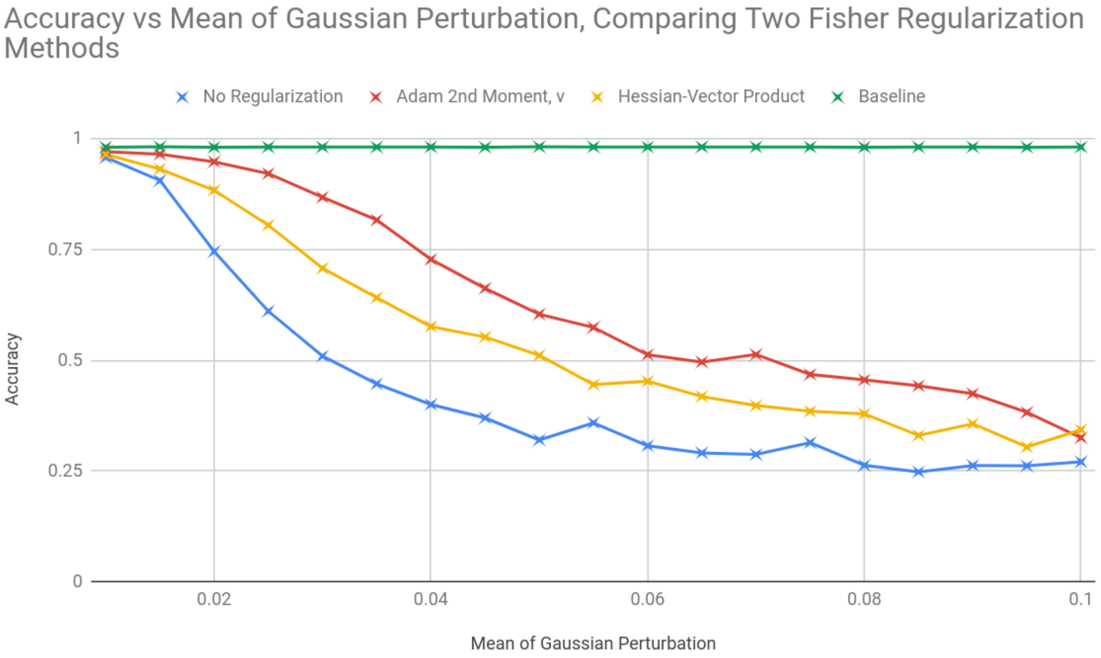


Figure 4.1: Accuracy of a two layer MLP on MNIST as a function of the mean of a Gaussian perturbation to the weights, $\mathcal{N}(\mu, 0.005)$. This plot shows that Regularizing with either methods of estimating the Fisher provide a benefit over no regularization. However, the diagonal approximation of the Fisher computed by the second moment of ADAM is more effective at regularizing a model against this perturbation.

In figure 4.1 we find that using the ADAM diagonal approximation to the Fisher provides more robustness to perturbations than using the HVP. It is reasonable to expect that since the HVP implicitly computes off diagonal elements of the Fisher information that ADAM does not, HVP should prove to be a better regularizer. However, this is likely overshadowed by the fact that the HVP is very noisy and varies greatly from batch to batch. On the other hand, the ADAM estimate is a moving average over many batches which leads to a more stable Fisher estimate.

The estimate of the Fisher provided by ADAM can be viewed as a high bias estimate, while the estimate provided by the HVP can be viewed as a high variance estimate.

4.2 Fisher Information Can Rank the Sensitivity of Parameters to Perturbation

The key foundation of this work is that Fisher information is a useful way of ranking parameter importance. To test this hypothesis we begin with a simple experiment. This hypothesis is tested on a v1 ResNet-18 (He *et al.*, 2016) trained on CIFAR-10.

Procedure - Verify Fisher Information Measures Parameter Importance:

1. Begin with a trained network with weights $\hat{\theta}$.
2. Repeat $N_{MC} = 30$ times to adequately sample the noise:
    (a) Perturb the parameters with noise $z \sim \mathcal{N}(0, \beta I)$
    (b) Record the average magnitude of the noise.
    (c) Record test accuracy with $\theta_{inference} = \hat{\theta} + z$
    (d) Sweep β.
3. Repeat step (2), but perturb the parameters with noise proportional to the Fisher $z \sim \mathcal{N}(0, \hat{F})$

The purpose of this experiment is to compare the performance degradation of the network under two cases. We start by considering the performance degradation under increasing Gaussian noise perturbations where the covariance of the noise is the identity matrix (i.e. in expectation every parameter is perturbed with the same magnitude of noise). Then, the network is perturbed such that magnitude of the

Gaussian perturbation is proportional to the Fisher information. The results of this are shown in Figure 4.2. As described in Section 3.7, the Fisher information is estimated via the main diagonal of the ADAM optimizer.

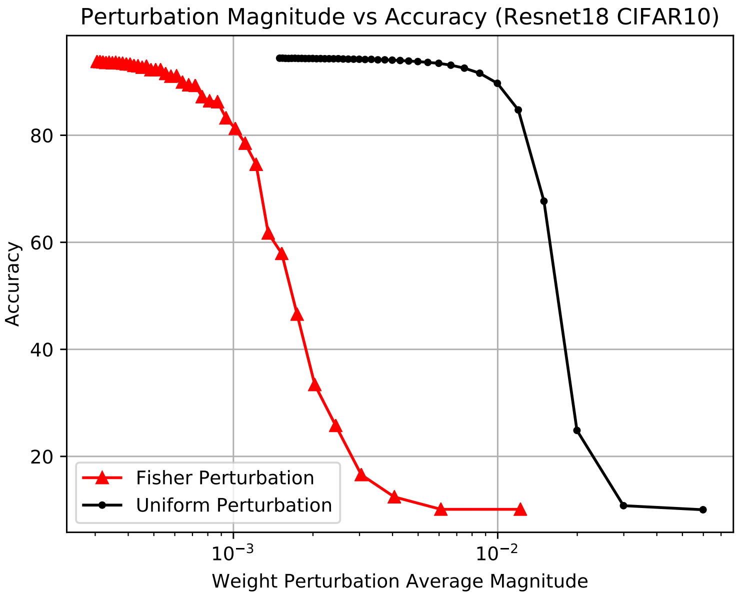


Figure 4.2: Accuracy of ResNet-18 on CIFAR-10 as a function of average magnitude of perturbation applied to the weights. The perturbations are normally distributed with identity covariance and covariance equivalent to Fisher information.

As expected, adding perturbations that are proportional to the Fisher information significantly harms the performance of the network (more so than using an identity covariance). This is because we are adding high variance noise to the most important parameters while adding low variance noise to the least important parameters (red curve in Fig. 4.2). This confirms the hypothesis that the loss landscape is more curved in the parameters with large Fisher values. These parameters can be interpreted as being less robust, and perturbations in these directions will result in a larger change to the loss function.

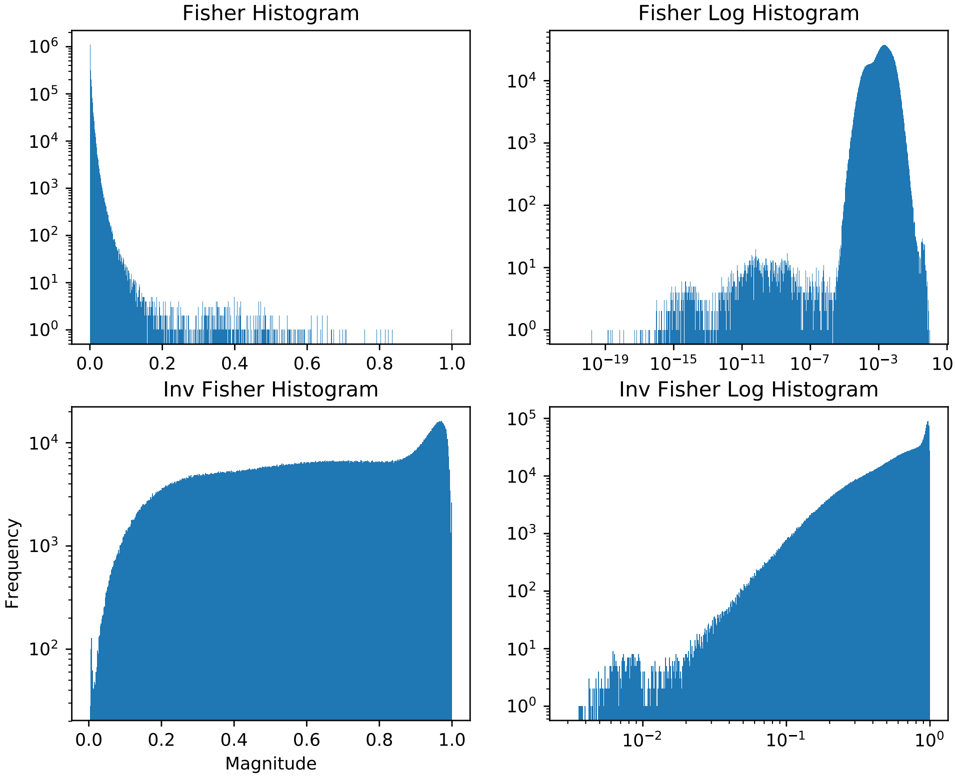


Figure 4.3: Histogram of the normalized Fisher Information diagonal values and normalized inverse Fisher (left two plots) for a ResNet-18 model trained on CIFAR-10. The same two histograms are also plotted with x-axis in log scale (right two plots).

Mismatch of Fisher Approximation of KL-Divergence

In addition to establishing that Fisher information is useful for ranking parameter importance, it is important to understand how accurately the KL-divergence is approximated with the Fisher approximation (Fig. 4.4). It is clear that the approximation is a good fit near small perturbations: $d\theta \approx 3 \times 10^{-3}$. As the perturbation size increases, the Fisher approximation can significantly deviate from the KL-divergence. For small $d\theta$ there is also a mismatch - the Fisher term is smaller than D_{KL}, likely due to the fact that $\sum_i F_{ii} d\theta_i^2$ will be very small if $d\theta$ is small since the perturbations are squared.

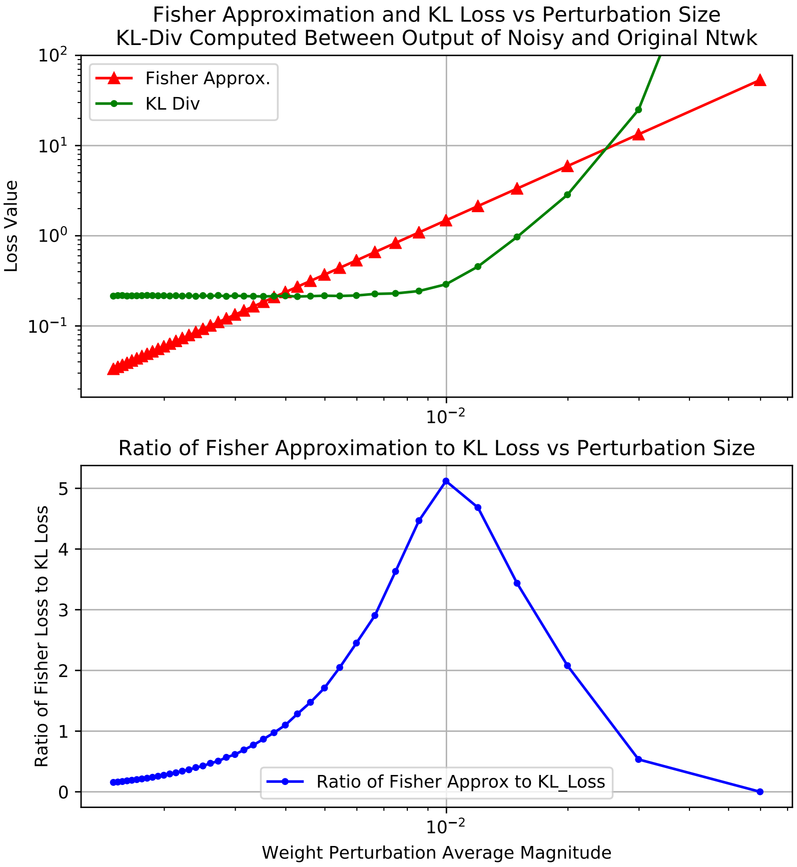


Figure 4.4: Plot of KL-divergence between the output of the perturbed and original network (green line), and the Fisher approximation to the KL divergence(red line). In blue the ratio of the Fisher approximation to the KL-divergence is plotted.

Lenet Acc (%) Bits Act/Wt	**STE**	**Fisher**	**MSQE**	**Distillation**
FP-32	92.84 ± .14	–	–	–
8/8	92.7 ± .1	92.64± .13	92.54 ± .16	**93.09 ± .06**
4/4	92.3 ± .13	92.28±.21	92.52± .16	**92.8±.06**
4/4	92.0 ± .11	92.3± .26	92.25±.21	92.21±.03
2/2	88.0 ± 1.03	89.99±.4	89.95±.36	*90.06±.26*
32/1	90.8 ± .11	**91.6 ± .21**	91.27 ± .11	91.32±.36
4/1	90.6 ± .19	91.17 ± .09	91.1 ± .21	*91.2*± .2
2/1	85.5 ± 0.9	86.92±1.3	86.8±.7	**88.19±.094**

Table 4.1: Accuracy of Lenet-5 on FashionMNIST for selected regularization schemes averaged over 5 trials (for each point). The baseline is training with straight-through estimator (STE). Bold indicates method is the best by a statistically significant margin, italics indicates method performed best, but not statistically significant.

We begin practical experiments by evaluating our method with the Lenet5 network on the FashionMNIST dataset. After training with the straight-through estimator for T epochs, we obtain a solution $\hat{\theta} = \theta_T$. Subsequently, the network is trained for R epochs with regularization. During these R epochs, $\hat{F}(\theta_T)$ is used for the updates. Instead of computing the Fisher Information every epoch, we keep the Fisher information estimate obtained at the maximum likelihood estimate $\hat{\theta}$. Therefore $\hat{F}(\hat{\theta} = \theta_T)$. This preserves the convergence property of the empirical Fisher, namely that it converges to the true Fisher since it is computed at a minimum (Pascanu and Bengio, 2013). For numerical stability, diagonal loading is performed such that $\hat{F}_T = (\hat{F}(\hat{\theta}_T) + \lambda I)$ where $\lambda = .005$.

Procedure

1. Train quantized network with straight-through estimator for $T = 30$ epochs to obtain parameters $\hat{\theta}$. To train this network R-ADAM (Algorithm 3) is used.

2. Compute Fisher Information at $\hat{\theta}$: $\hat{F}_T = (\hat{F}(\hat{\theta}_T) + \lambda I)$.

3. Train network for an additional $R = 25$ epochs using the following regularization: Fisher, mean squared quantization error, distillation.

4. For distillation, a full precision teacher model is used whose accuracy is reported in the table. We choose a temperature of 4.

Regularizing Mean Squared Quantization Error(MSQE). The essence of the Fisher regularization method is that a weighted sum of squared quantization errors is minimized: $\sum F_i(\theta_i - Q(\theta_i))^2$. It is important to compare regularization using the FIM versus using identity matrix because the identity matrix regularizes the perturbations for all θ regardless of whether the loss highly curved or flat with respect to θ_i. Using an identity matrix is equivalent to minimizing the mean squared quantization error: $MSQE(\theta) = \sum_i (\theta_i - \theta)^2$.

Discussion of Results for Lenet5. Distillation often has the highest accuracy. This is due to two factors. First, distillation is *directly* minimizing the KL-divergence between the full precision model and the perturbed quantized model. Secondly, the teacher model is proving information through its logits about the relationship between categories that that MSQE, and Fisher methods don't have access to (see Sec 3.8). Despite this disadvantage, the Fisher regularization scheme performs best for 32b activations and 1b weights. But, in most cases the resultant accuracy of Fisher regularization is similar to the accuracy from using MSQE.

4.4 Quantizing ResNet-18 for CIFAR-10

ResNet-18 Acc (%) Bits Act/Wt	**STE**	**Fisher**	**MSQE**	**Distillation**
FP-32	94.4 ± .18	-	-	
4/1	90.65 ± .38	91.2 ± .19	91.3 ± .28	*91.39± .24*
4/4	93.44 ± .17	93.4 ± .18	93.4± .17	**94.19 ± .14**

Table 4.2: Accuracy of Resnet-18 on CIFAR-10 for selected regularization schemes averaged over 5 trials (for each point). The baseline is training with the straight-through estimator (STE). Bold indicates method is the best by a statistically significant margin, italics indicates method performed best, but not necessarily statistically significant.

A similar trend to the results for Lenet5 on FashionMNIST is observed here, where distillation provides the highest accuracy. The other three regularization methods do not perform any better than baseline.

Procedure

1. Train quantized network with straight-through estimator for $T = 300$ epochs to obtain parameters $\hat{\theta}$. R-SGD (Algorithm 2) is used to train the network with $\gamma = 0.$ for the first T epochs. A cosine learning rate decay is used starting at a learning rate of $\alpha = .1$.

2. Compute Fisher Information at $\hat{\theta}$: $\hat{F}_T = (\hat{F}(\hat{\theta}_T) + \lambda I)$.

3. Train network for an additional $R = 100$ epochs using the following regularization: Fisher, mean squared quantization error, distillation.

4. For distillation, a full precision teacher model is used whose accuracy is reported in the table. We choose a temperature of 4.

4.4.1 *Effect of Regularization Methods on Weight Distribution*

Figure 4.5 and Figure 4.7 demonstrate how the different regularization schemes affect the parameter space. The weight PDF is plotted as a function of training iteration for ResNet-18 under binary quantization of the weights to $[-1, 1]$. It is clear that as training iteration increases (back to front), each regularization method has a drastically different effect on the weight PDFs. MSQE regularization (right plots, blue) affects the parameter space distribution the most; this is expected since the MSQE regularizer *only* has knowledge of the parameter space and aggressively pushes θ towards the quantized bins of $Q(\theta)$. Next, consider the Fisher regularization (left plots, light orange). It is apparent that while some of the probability mass is transferred to the quantizer bins, there is still significant probability mass in between the quantizer levels. This indicates that the Fisher regularizer has incorporated some information about the statistical manifold of the model.

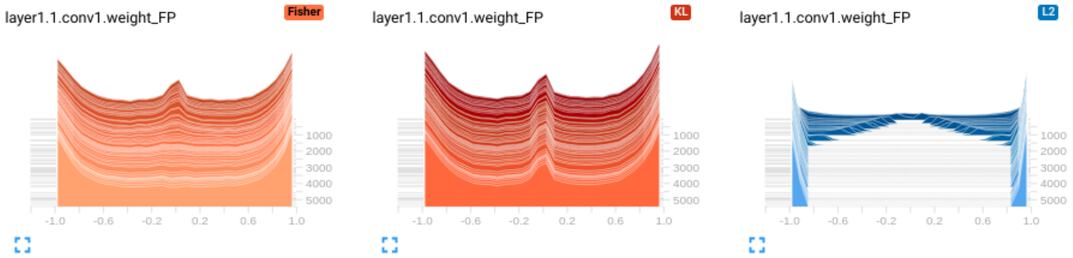


Figure 4.5: Weight PDF of first conv layer of ResNet-18 under binary quantization [-1, 1] for three different regularization methods (from left to right: Fisher, Distillation , MSQE) as a function of iteration.

4.4.2 *Straight-Through Estimator Limits Accuracy Gains*

One reason that there might be no accuracy increase provided by any of the methods that use a squared error quantization constraint is the use of the

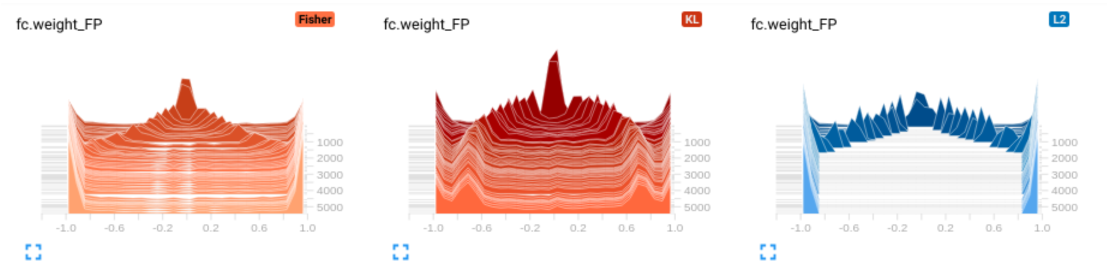


Figure 4.6: Weight PDF of last fully connected layer in ResNet-18 under binary quantization [-1, 1] for three different regularization methods (from left to right: Fisher, Distillation, MSQE) as a function of iteration.

straight-through estimator. In the forward pass, the weights are already quantized to the bin that they are closest to. This essentially means that in the forward pass $d\theta = 0$. If the regularizer pushes the weight towards the quantizer bin that it already belongs to, there will be no change in the output of the network. This is regardless of whether uninformative parameters are more harshly regularized.

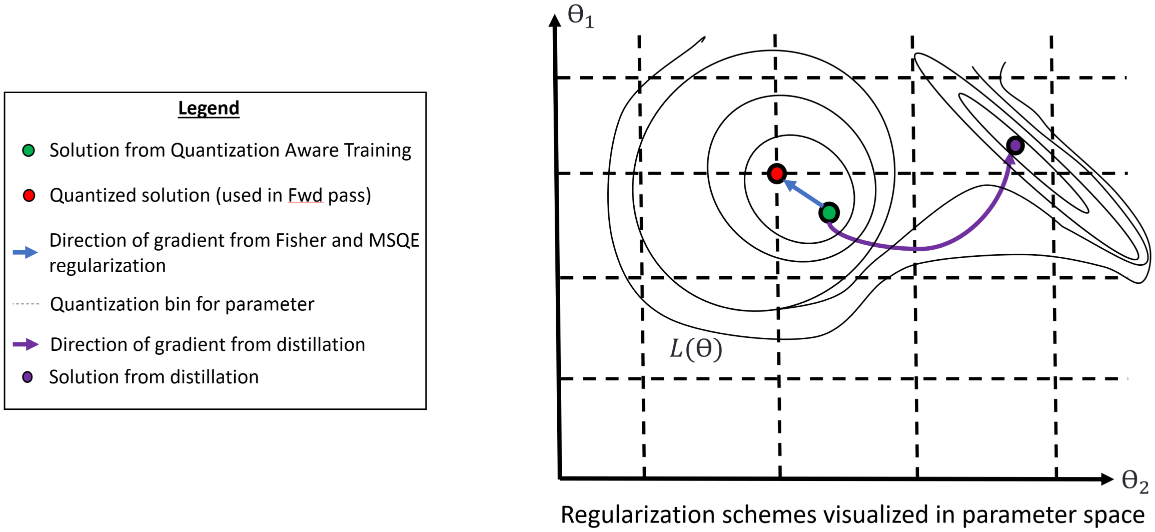

Legend
● Solution from Quantization Aware Training
● Quantized solution (used in Fwd pass)
Direction of gradient from Fisher and MSQE regularization
Quantization bin for parameter
Direction of gradient from distillation
● Solution from distillation
Θ1
Θ2
L(Θ)
Regularization schemes visualized in parameter space

Figure 4.7: Visualization of an example loss surface in parameter space. When the straight-through estimator is used in the forward pass, the weights are always quantized when performing inference. Therefore, when a network's optimial parameters $\hat{\theta}$ (shown in green) are found, during inference $Q(\theta)$ (shown in red) is actually used. Therefore, it makes minimal difference to the output of the network if the difference between $Q(\theta)$ and $\hat{\theta}$ is minimized.

Parameter Space versus Probability Space Regularization

Consider the weight distributions under distillation (middle plots, dark orange). The conv layer weight PDF hardly changes at all and the FC layer's weight PDF is only partially pushed towards the quantizer's bins. Upon examination of the distillation loss (Table 4.3) notice that it has no explicit dependence on the parameters. Rather, it is dependent only on the probability that the network assigns to each class. Distillation directly regularizes the logits of the quantized student model and matches them to the FP32 teacher's logits. In contrast, the parameter space regularization methods hope to indirectly increase the accuracy of the perturbed network through reducing the size of the perturbation that is applied to the parameters.

In Figure 4.8 the Fisher regularization term (Fisher-weighted MSQE) and the MSQE are plotted normalized to their maximum value during the R epochs that the model is regularized. The MSQE for the three parameter based regularization approaches decreases. As expected, using Fisher regularization priorities minimizing the Fisher MSQE the most, but also has an impact on the MSQE.

Counter intuitively, the MSQE increases and Fisher MSQE greatly increases when regularizing with distillation. It is not surprising given our observations of the weight PDFs of regularizing with distillation that the MSQE may increase. But, one might expect that because distillation minimizes the KL-divergence, the Fisher MSQE should also decrease since the Fisher MSQE is an approximation of the KL-divergence. A plot of the the trace of the Fisher information (FIM) is provided in Figure 4.11 and reveals that regularizing with distillation massively increases the trace of the FIM, while the three parameter space based approaches only slightly increase the trace or keep it constant. The trace of the FIM is a crude way to measure the curvature of the loss surface since we sum the curvature with respect to each individual parameter. This suggests that though the solution found by distillation is at a sharper minimum (higher trace, larger curvature), it still generalizes better. Regularizing with either of the three parameter space based approaches

Reglarization Method	**Minimized Quantity**
Fisher	$\sum_i F_{ii}(\theta_i - Q(\theta_i))^2$
MSQE	$\sum_i(\theta_i - Q(\theta_i))^2$
Distillation	$H(p_\theta, p_{Q(\theta)}) \equiv D_{KL}(p_\theta \| p_{Q(\theta)})$

Table 4.3: A comparison of the optimization targets for the four regularizers that are studied.

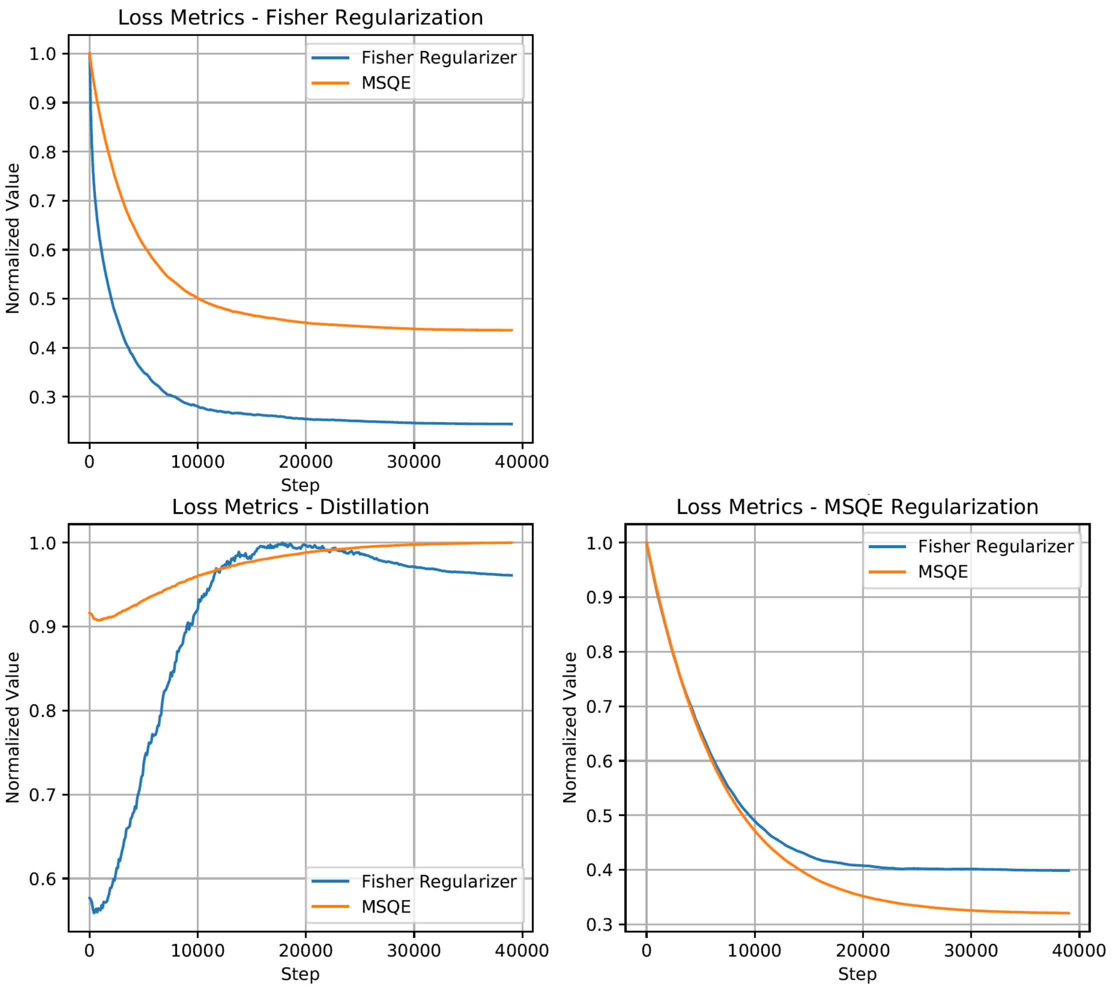


Figure 4.8: Plot of mean squared quantization error, $\sum_i(\theta_i - Q(\theta_i))^2$, and Fisher regularizer (Fisher MSQE), $\sum_i(F_{ii} + \lambda I)(\theta_i - Q(\theta_i))$, normalized to their maximum value. These two metrics are plotted for each regularization method for a ResNet-18 trained for 4b/4b quantization on CIFAR-10.

Comparison With State of the Art Methods for Very Low Bit Width Quantization

In this section, the performance of ResNet-18 under quantization to a very low number of bits (2 bit activations and weights) is studied. Our methods are compared with the method proposed by Choi *et al.* (2019). One challenge with quantization to below 4-bits in our approach is that the ReLu activation function's

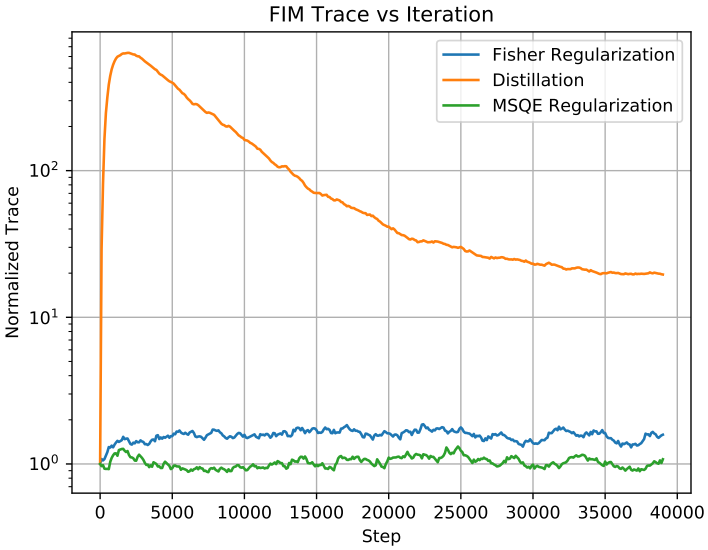


Figure 4.9: Plot of trace of FIM versus iteration.

output is unbounded. Therefore choosing the quantizer based on the maximum and minimum of the activation is no longer feasible for low bit width. Instead, the levels of the quantizer are set prior to training at $[-.25, .25]$. Note that Choi *et al.* (2019) use their own solution of parameterized activation clipping to overcome this problem.

ResNet-18 **Acc (%)** Bits Act/Wt	**STE**	**MSQE**	**Choi** **(2019)**	**Fisher**	**Distillation**
FP-32	94.4 ± .18	-	-	-	
2/2	89.26 ± .15	89.35 ± .15	90.8	89.43± .4	89.89 ± 0.16
32/2	93.52 ± .21	93.61 ± .21	91.6	93.587 ± .23	93.947 ± .25

Table 4.4: Accuracy of Resnet-18 on CIFAR-10 for selected regularization schemes averaged over 5 trials (for each point). The baseline is training with the straight-through estimator (STE). A comparison with Choi *et al.* (2019) is included. Each point is the average of 5 trials.

Notice that when the activation is quantized very aggressively, the accuracy of our quantization method (learning the minimum and maximum value of the activations at each layer and using it as a basis for a uniform quantizer) results in significantly reduced baseline accuracy using straight through estimator. This limits the performance of MSQE, Fisher, and distillation, and explains why (Choi *et al.*, 2019) outperforms our method for quantizing both weights and activations to 2 bits. However, when allowing 32 bit activations, the method used by (Choi *et al.*, 2019) performs the worst due to their lower baseline.

4.5 Effect of Analog Hardware Noise from NVM Accelerator on Lenet-5

Procedure - Regularizing Network from Analog Noise

1. Begin with a pre-trained Lenet5 network.

2. Select a range of noise levels $\eta \in [.01, .1]$.

3. Characterize accuracy of network at various noise levels by sweeping η and adding noise layer-wise to the weights as described in Sec 3.10: $z^{(l)} \sim \mathcal{N}(0, \eta|\max(\theta^{(l)} - \min(\theta^{(l)})|)$. Then perform inference with $\theta^{(l)}_{noisy} = \theta^{(l)} + z^{(l)}$. At every noise level test accuracy is computed 20 times to adequately draw from noise distribution.

4. Retrain the network with noise injection for 25 epochs. Apply regularization methods (Fisher, mean squared error, distillation) in conjunction to the noise injection

5. The perturbation that is regularized is $z^{(l)} = (\theta^{(l)} - \theta^{(l)}_{noisy})$. Train with regularizer for $T = 25$ epochs. Use $\gamma = .1$, distillation temperature of 4.

6. Characterize test accuracy at each noise level for each regularization method.

The results of regularizing for robustness against noise from our model of an NVM compute cell are shown in figure 4.10. Distillation provides the most robustness - for an iso-accuracy of 90%, distillation can tolerate nearly 20% more noise than other methods. It is interesting to note that the perturbation based regularization schemes (Fisher, MSE) fare no better than retraining with noise and all perform very similarly. The perturbation that the network is being regularized against is simply a zero mean Gaussian random variable ($z^{(l)}$). This is in contrast to the

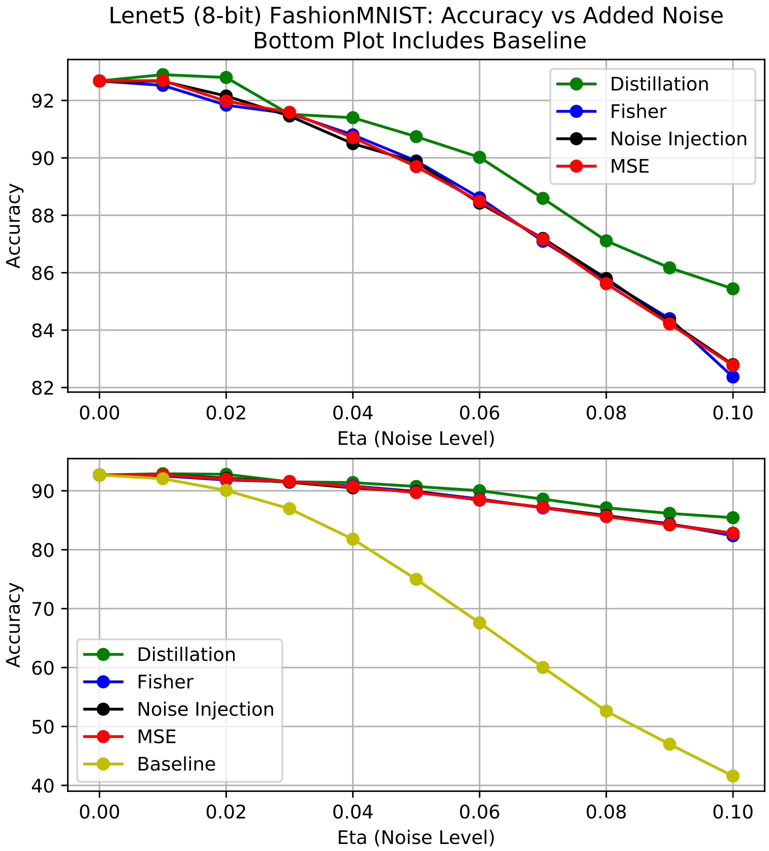


Figure 4.10: Plot of the effect of NVM accelerator noise on a Lenet-5 network trained on FashionMNIST. Baseline indicates that the pretrained network was evaluated without any retraining or regularization. Noise Injection corresponds to adding noise to weights during training time. Noise was added to weights in addition to regularization for each of the regularization methods. Refer to Section 3.10 for an overview of the noise model.

previous section where the perturbation was the difference between a weight and its quantized value.

Regularizer Has No Effect on Gradient When Perturbation is Zero Mean

The expected value of the regularized gradient is simply equal to the original gradient. Let us examine the expected value of the gradient under Fisher regularization, where we have used that $d\theta = z$.

$$\begin{aligned} \mathbb{E}[\nabla L(\theta) + \gamma(F + \lambda I)z] &= \mathbb{E}[\nabla L(\theta)] + \mathbb{E}[\gamma(F + \lambda I)z] \\ &= \mathbb{E}[\nabla L(\theta)] + (F + \lambda I)\mathbb{E}[z] \\ &= \mathbb{E}[\nabla L(\theta)] + 0 \end{aligned} \tag{4.1}$$

This is in contrast to quantization and the experiment in Section 4.1 since in those cases, the perturbations do not have zero mean. Therefore, we can conclude that zero-mean perturbations are not effectively regularized by the Fisher or MSE method.

4.6 Effect of Analog Hardware Noise from NVM Accelerator on ResNet-18

The experiment for the previous section is repeated on a more challenging dataset and network. However, we only consider noise injection and distillation, since we showed that the other regularization methods are ineffective at providing robustness to zero mean noise. Again, we notice that distillation outperforms noise injection. For an iso-accuracy of 92%, distillation can tolerate ∼12% more noise. For iso-accuracy of 91% distillation can tolerate ∼15% more noise. Notice that as noise level increases distillation outperforms noise injection by an increasing margin.

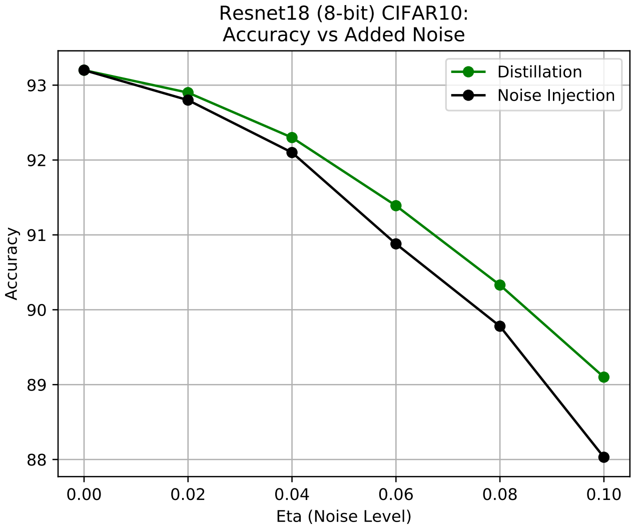


Figure 4.11: Plot of accuracy versus NVM hardware noise for an 8-bit ResNet-18.

Chapter 5

CONCLUSION

In this work, we have evaluated the efficacy of a natural-gradient inspired regularization scheme that penalizes a deep neural network for sensitivity to perturbations in its parameter space. The Fisher Information Matrix was used to approximate the KL-divergence between the perturbed and original model. It was shown that by using information about the curvature of the loss, the Fisher can effectively measure the relative importance of a parameter and regularize the model according to this importance criterion. We also demonstrated knowledge distillation as way of minimizing the KL-divergence. Despite the fact that the curvature matrices of neural network loss landscapes are highly non-diagonal, a diagonal approximation was found to be more effective at regularizing the model from perturbations than using a Hessian-vector product. This is likely because the diagonal approximation we used was more stable than using a batched HVP. Experiments showed that while the Fisher approximation to the KL-divergence was crude, it was able to regularize networks more intelligently than a simple mean squared quantization error constraint.

The study of perturbation of deep networks was motivated by the need for efficient, low power deep learning inference. To this end the perturbations to the network that we considered were perturbation due to quantization quantization of the weights and due to noise from mixed-signal analog hardware. A simple layer-wise additive Gaussian based on the density of the weights was used to model mixed-signal noise.

Distillation proved to be an extremely powerful method of directly regularizing the output distribution of the model. Comparing a mean squared error constraint, distillation, and Fisher regularizer we showed that Fisher regularization was be weakly dependent on the parameterization, but that distillation is largely independent of the parameterization. Fisher regularization outperformed distillation and mean squared error constrained quantization on Lenet-5 with binarized weights. Distillation consistently outperformed all other methods on most configurations of quantization and provided a 0.75% increase in performance on ResNet-18 quantized to 4bit activations and binary weights over training with straight-through estimator. Fisher regularization did not provide additional robustness to noise from mixed-signal analog hardware than simply training with noise injection, since regularizing a zero-mean perturbation does not effect the expected value of the gradient. However, distillation provided 12-20% more tolerance to analog hardware noise than retraining with noise injection on an 8-bit ResNet-18.

There are several avenues of future work. It is important to improve the estimation of the FIM. Many recent Hessian-Free optimization methods such as KFAC (Martens and Grosse, 2015) have been developed to efficiently calculate the FIM and its inverse and are directly applicable to this approach. This could improve the accuracy approximation of the KL-divergence that the Fisher regularizer computes. The Fisher regularizer may be more suited to post-training quantization of a FP32 network rather than regularization of a network trained for quantization with straight through estimator, since moving weights towards their quantized bins has no effect on the output distribution if the network is already quantized in the forward pass. Another area of focus could be refining the noise model for the analog hardware. If certain areas of the accelerator are more susceptible to noise, a reasonable method might assign weights with low saliency weights (as measured by

Fisher information) to those high noise sections of the accelerator. It may also be possible to combine the Fisher regularization with distillation by performing natural gradient descent on the distillation loss rather than on the cross entropy loss.

REFERENCES

Agarwal, S., S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James and M. J. Marinella, "Resistive memory device requirements for a neural algorithm accelerator", in "2016 International Joint Conference on Neural Networks (IJCNN)", pp. 929–938 (IEEE, 2016).

Amari, S.-I., "Natural gradient works efficiently in learning", Neural computation **10**, 2, 251–276 (1998).

Anonymous, "Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation", in "Submitted to International Conference on Learning Representations", (2020), URL `https://openreview.net/forum?id=BklxN0NtvB`, under review.

Balzer, W., M. Takahashi, J. Ohta and K. Kyuma, "Weight quantization in boltzmann machines", Neural Networks **4**, 3, 405–409 (1991).

Banner, R., Y. Nahshan, E. Hoffer and D. Soudry, "Aciq: Analytical clipping for integer quantization of neural networks", arXiv preprint arXiv:1810.05723 (2018).

Bengio, Y., N. Léonard and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation", arXiv preprint arXiv:1308.3432 (2013).

Binas, J., D. Neil, G. Indiveri, S.-C. Liu and M. Pfeiffer, "Precise deep neural network computation on imprecise low-power analog hardware", arXiv preprint arXiv:1606.07786 (2016).

Choi, J., S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang and P. Chuang, "Accurate and efficient 2-bit quantized neural networks", in "Proceedings of the 2nd SysML Conference", (2019).

Choi, Y., M. El-Khamy and J. Lee, "Towards the limit of network quantization", arXiv preprint arXiv:1612.01543 (2016).

Choi, Y., M. El-Khamy and J. Lee, "Learning low precision deep neural networks through regularization", arXiv preprint arXiv:1809.00095 (2018).

Courbariaux, M., Y. Bengio and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations", in "Advances in neural information processing systems", pp. 3123–3131 (2015).

Duchi, J., E. Hazan and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", Journal of Machine Learning Research **12**, Jul, 2121–2159 (2011).

Dundar, G. and K. Rose, "The effects of quantization on multilayer neural networks", IEEE Transactions on Neural Networks **6**, 6, 1446–1451 (1995).

Fedorov, I., R. P. Adams, M. Mattina and P. N. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers", (2019).

Finn, C., P. Abbeel and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks", in "Proceedings of the 34th International Conference on Machine Learning-Volume 70", pp. 1126–1135 (JMLR. org, 2017).

Han, S., H. Mao and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", arXiv preprint arXiv:1510.00149 (2015).

Hassibi, B., D. G. Stork and G. Wolff, "Optimal brain surgeon: Extensions and performance comparisons", in "Advances in neural information processing systems", pp. 263–270 (1994).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 770–778 (2016).

Hinton, G., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", arXiv preprint arXiv:1503.02531 (2015).

Hou, L. and J. T. Kwok, "Loss-aware weight quantization of deep networks", arXiv preprint arXiv:1802.08635 (2018).

Hou, L., Q. Yao and J. T. Kwok, "Loss-aware binarization of deep networks", arXiv preprint arXiv:1611.01600 (2016).

Howard, A., M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3", arXiv preprint arXiv:1905.02244 (2019).

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations", The Journal of Machine Learning Research **18**, 1, 6869–6898 (2017).

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations", Journal of Machine Learning Research (2018).

Jain, S., A. Sengupta, K. Roy and A. Raghunathan, "Rx-caffe: Framework for evaluating and training deep neural networks on resistive crossbars", arXiv preprint arXiv:1809.00072 (2018).

Joshi, V., M. L. Gallo, I. Boybat, S. Haefeli, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory", arXiv preprint arXiv:1906.03138 (2019).

Kakade, S. M., "A natural policy gradient", in "Advances in neural information processing systems", pp. 1531–1538 (2002).

Kingma, D. P. and J. Ba, “Adam: A method for stochastic optimization”, arXiv preprint arXiv:1412.6980 (2014).

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, “Overcoming catastrophic forgetting in neural networks”, Proceedings of the national academy of sciences **114**, 13, 3521–3526 (2017).

Kunstner, F., L. Balles and P. Hennig, “Limitations of the empirical fisher approximation”, arXiv preprint arXiv:1905.12558 (2019).

LeCun, Y., J. S. Denker and S. A. Solla, “Optimal brain damage”, in “Advances in neural information processing systems”, pp. 598–605 (1990).

Lee, J., C. Kim, S. Kang, D. Shin, S. Kim and H.-J. Yoo, “Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision”, IEEE Journal of Solid-State Circuits **54**, 1, 173–185 (2018).

Liu, Z., B. Wu, W. Luo, X. Yang, W. Liu and K.-T. Cheng, “Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm”, in “Proceedings of the European Conference on Computer Vision (ECCV)”, pp. 722–737 (2018).

Loshchilov, I. and F. Hutter, “Fixing weight decay regularization in adam”, arXiv preprint arXiv:1711.05101 (2017).

Martens, J., “Deep learning via hessian-free optimization.”, (2010).

Martens, J., “New insights and perspectives on the natural gradient method”, arXiv preprint arXiv:1412.1193 (2014).

Martens, J. and R. Grosse, “Optimizing neural networks with kronecker-factored approximate curvature”, in “International conference on machine learning”, pp. 2408–2417 (2015).

Martens, J. and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization”, in “Proceedings of the 28th International Conference on Machine Learning (ICML-11)”, pp. 1033–1040 (Citeseer, 2011).

Mishra, A., E. Nurvitadhi, J. J. Cook and D. Marr, “Wrpn: wide reduced-precision networks”, arXiv preprint arXiv:1709.01134 (2017).

Papernot, N., P. McDaniel, X. Wu, S. Jha and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks”, in “2016 IEEE Symposium on Security and Privacy (SP)”, pp. 582–597 (IEEE, 2016).

Pascanu, R. and Y. Bengio, “Revisiting natural gradient for deep networks”, arXiv preprint arXiv:1301.3584 (2013).

Patil, A. D., H. Hua, S. Gonugondla, M. Kang and N. R. Shanbhag, "An mram-based deep in-memory architecture for deep neural networks", in "2019 IEEE International Symposium on Circuits and Systems (ISCAS)", pp. 1–5 (IEEE, 2019).

Pearlmutter, B. A., "Fast exact multiplication by the hessian", Neural computation **6**, 1, 147–160 (1994).

Polino, A., R. Pascanu and D. Alistarh, "Model compression via distillation and quantization", arXiv preprint arXiv:1802.05668 (2018).

Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", in "European Conference on Computer Vision", pp. 525–542 (Springer, 2016).

Rekhi, A. S., B. Zimmer, N. Nedovic, N. Liu, R. Venkatesan, M. Wang, B. Khailany, W. J. Dally and C. T. Gray, "Analog/mixed-signal hardware error modeling for deep learning inference", in "Proceedings of the 56th Annual Design Automation Conference 2019", p. 81 (ACM, 2019).

Roux, N. L., P.-A. Manzagol and Y. Bengio, "Topmoumoute online natural gradient algorithm", in "Advances in neural information processing systems", pp. 849–856 (2008).

Shafiee, A., A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars", ACM SIGARCH Computer Architecture News **44**, 3, 14–26 (2016).

Song, L., X. Qian, H. Li and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning", in "2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)", pp. 541–552 (IEEE, 2017).

Sze, V., Y.-H. Chen, T.-J. Yang and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey", Proceedings of the IEEE **105**, 12, 2295–2329 (2017).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1–9 (2015).

Tu, M., V. Berisha, Y. Cao and J.-s. Seo, "Reducing the model order of deep neural networks using information theory", in "2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)", pp. 93–98 (IEEE, 2016).

Xie, Y. and M. A. Jabri, "Analysis of the effects of quantization in multilayer neural networks using a statistical model", IEEE Transactions on Neural Networks **3**, 2, 334–338 (1992).

Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients”, arXiv preprint arXiv:1606.06160 (2016).

Zoph, B. and Q. V. Le, “Neural architecture search with reinforcement learning”, arXiv preprint arXiv:1611.01578 (2016).