Stochastic Layer-Wise Precision in Deep Neural Networks

Griffin Lacey*
NVIDIA

Graham W. Taylor
University of Guelph
Vector Institute for Artificial Intelligence
Canadian Institute for Advanced Research

Shawki Areibi
University of Guelph

Abstract

Low precision weights, activations, and gradients have been proposed as a way to improve the computational efficiency and memory footprint of deep neural networks. Recently, low precision networks have even shown to be more robust to adversarial attacks. However, typical implementations of low precision DNNs use uniform precision across all layers of the network. In this work, we explore whether a heterogeneous allocation of precision across a network leads to improved performance, and introduce a learning scheme where a DNN stochastically explores multiple precision configurations through learning. This permits a network to learn an optimal precision configuration. We show on convolutional neural networks trained on MNIST and ILSVRC12 that even though these nets learn a uniform or near-uniform allocation strategy respectively, stochastic precision leads to a favourable regularization effect improving generalization.

1 INTRODUCTION

Recent advances in deep learning, and convolutional neural networks (CNN) in particular, have led to well-publicized breakthroughs in computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), and natural language processing (NLP) (Bahdanau et al., 2014). Modern CNNs, however, have increasingly large storage and computational requirements (Canziani et al., 2016). This has limited the application scope to data centres that can accommodate clusters of massively parallel hardware accelerators, such as graphics processing units (GPUs). Still, GPU training

of CNNs on large datasets like ImageNet (Deng et al., 2009) can take hours, even on networks with hundreds of GPUs, and achieving linear scaling beyond these sizes is difficult (Goyal et al., 2017). As such, there is a growing interest in investigating more fine-grained optimizations, especially since deployment on embedded devices with limited power, compute, and memory budgets remains an imposing challenge.

Research efforts to reduce model size and speed up inference have shown that training networks with binary or ternary weights and activations (Courbariaux and Bengio, 2016; Rastegari et al., 2016; Li et al., 2016a) can achieve comparable accuracy to full precision networks, while benefiting from reduced memory requirements and improved computational efficiency using bit operations. They may even confer additional robustness to adversarial attacks (Galloway et al., 2018). More recently, the DoReFa-Net model has generalized this finding to include different precision settings for weights vs. activations, and demonstrated how low precision *gradients* can be also employed at training time (Zhou et al., 2016).

These findings suggest that precision in deep learning is not an arbitrary design choice, but rather a dial that controls the trade-off between model complexity and accuracy. However, precision is typically considered at the design level of an entire model, making it difficult to consider as a tunable hyperparameter. We posit that considering precision at a finer granularity, such as a layer or even per-example could grant models more flexibility in which to find optimal configurations, which maximizes accuracy and minimizes computational cost. To remain deterministic about hardware efficiency, we aim to do this for fixed budgets of precision, which have predictable acceleration properties.

In this work we consider learning an optimal precision configuration across the layers of a deep neural network, where the precision assigned to each layer may be different. We propose a stochastic regularization technique

---

*Work completed while at the University of Guelph

akin to Dropout (Srivastava et al., 2014) where a network explores a different precision configuration per example. This introduces non-differentiable elements in the computational graph which we circumvent using recently proposed gradient estimation techniques.

2 RELATED WORK

Recent work related to efficient learning has explored a number of different approaches to reducing the effective parameter count or memory footprint of CNN architectures. Network compression techniques (Han et al., 2015a; Wang and Liang, 2016; Choi et al., 2016; Agustsson et al., 2017) typically compress a pre-trained network while minimizing the degradation of network accuracy. However, these methods are decoupled from learning, and are only suitable for efficient deployment. Network pruning techniques (Wan et al., 2013; Han et al., 2015b; Jin et al., 2016; Li et al., 2016b; Anwar et al., 2017; Molchanov et al., 2016; Tung et al., 2017) take a more iterative approach, often using regularized training and retraining to rank and modify network parameters based on their magnitude. Though coupled with the learning process, iterative pruning techniques tend to contribute to slower learning, and result in sparse connections, which are not hardware efficient.

Our work relates primarily to low precision techniques, which have tended to focus on reducing the precision of weights and activations used for deployment while maintaining dense connectivity. Courbariaux et al. were among the first to explore binary weights and activations (Courbariaux et al., 2015; Courbariaux and Bengio, 2016), demonstrating state-of-the-art results for smaller datasets (MNIST, CIFAR-10, and CVHN). This idea was then extended further with CNNs on larger datasets like ImageNet with binary weights and activations, while approximating convolutions using binary operations (Rastegari et al., 2016). Related work (Kim and Smaragdis, 2016) has validated these results and shown neural networks to be remarkably robust to an even wider class of non-linear projections (Merolla et al., 2016). Ternary quantization strategies (Li et al., 2016a) have been shown to outperform their binary counterparts, moreso when parameters of the quantization module are learned through backpropagation (Zhu et al., 2016). Cai et al. have investigated how to improve the gradient quality of quantization operations (Cai et al., 2017), which is complimentary to our work which relies on these gradients to learn precision. Zhou et al. further explored this idea of variable precision (i.e. heterogeneity across weights and activations) and discussed the general trade-off of precision and accuracy, exploring strategies for training with low precision gradients (Zhou et al., 2016).

Our approach for learning precision closely resembles BitNet (Raghavan et al., 2017), where the optimal precision for each network layer is learned through gradient descent, and network parameter encodings act as regularizers. While BitNet uses the Lagrangian approach of adding constraints on quantization error and precision to the objective function, we allocate bits to layers through sampling from a Gumbel-Softmax distribution constructed over the network layers. This has the advantage of accommodating a defined precision budget, which allows more deterministic hardware constraints, as well as a wider range of quantization encodings through the use of non-integer quantization values early in training. In the allocation of bits on a budget, our work resembles (Wang and Liang, 2016), though we allow more fine-grained control over precision, and prefer a gradient-based approach over clustering techniques for learning optimal precision configurations.

To the best of our knowledge, our work is the first to explore learning precision in deep networks through a continuous-to-discrete annealed quantization strategy. Our contributions are as follows:

- We experimentally confirm a linear relationship between total number of bits and speedup for low precision arithmetic, motivating the use of precision budgets.

- We introduce a gradient-based approach to learning precision through sampling from a Gumbel-Softmax distribution constructed over the network layers, constrained by a precision budget.

- We empirically demonstrate the advantage of our end-to-end training strategy as it improves model performance over simple uniform bit allocations.

3 EFFICIENT LOW PRECISION NETWORKS

Low precision learning describes a set of techniques that take network parameters, typically stored at native 32-bit floating point (FP32) precision, and quantize them to a much smaller range of representation, typically 1-bit (binary) or 2-bit (ternary) integer values. While low precision learning could refer to any combination of quantizing weights (W), activations (A), and gradients (G), most relevant work investigates the effects of quantizing weights and activations on model performance. The benefits of quantization are seen in both computational and memory efficiency, though generally speaking, quantization leads to a decrease in model accuracy (Zhou et al., 2016). However, in some cases, the effects of quantization can be lossless or even slightly improve accu-

racy by behaving as a type of noisy regularization (Zhu et al., 2016; Yin et al., 2016). In this work, we adopt the DoReFa-Net model (Zhou et al., 2016) of quantizing all network parameters (W,A,G) albeit at different precision. Table 1 demonstrates this trade-off of precision and accuracy for some common low precision configurations.

The justification for this loss in accuracy is the efficiency gain of storing and computing low precision values. The computational benefits of using binary values are seen from approximating expensive full precision matrix operations with binary operations (Rastegari et al., 2016), as well as reducing memory requirements by packing many low precision values into full precision data types. For other low precision configurations that fall between binary and full precision, a similar formulation is used. The bit dot product equation (Equation 1) shows how both the logical `and` and `bitcount` operations are used to compute the dot product of two low-bitwidth fixed-point integers (bit vectors) (Zhou et al., 2016). Assume $c_m(x)$ is a placewise bit vector formed from a sequence of M-bit fixed-point integers $x = \sum_{m=0}^{M-1} c_m(x)2^m$ and $c_k(y)$ is a placewise bit vector formed from a sequence of K-bit fixed-point integers $y = \sum_{k=0}^{M-1} c_k(y)2^k$, then

$$x \cdot y = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \, \text{bitcount}[\text{and}(c_m(x), c_k(y))],$$
$$c_m(x)_i, c_k(y)_i \in \{0,1\} \forall i, m, k. \tag{1}$$

Since the computational complexity of the operation is $\mathcal{O}(MK)$, the speedup is a function of the total number of bits used to quantize the inputs. Since matrix multiplications are simply sequences of dot products computed over the rows and columns of matrices, this is also true of matrix multiplication operations. As a demonstration, we implement this variable precision bit general matrix multiplication (bit-GEMM) using CUDA, and show the GPU speedup for several configurations in Figure 1.

As seen in Figure 1, there is a correlation between the total number of bits used in each bit-GEMM (shade) and the resulting speedup (point size). As such, from a hardware perspective, it is important to know how many total bits are used for bit-GEMM operations to allow for budgeting computation and memory. It should also be noted that, in our experiments, operations with over 16 total bits of precision were shown to be slower than the full precision equivalent. This is due to the computational complexity of the worst case of $\mathcal{O}(8 \times 8)$ being slower than the equivalent full precision operation. We therefore focus on operations with 16 or less total bits of precision.

A natural question to follow is then, for a given budget of precision (total number of bits), how do we most
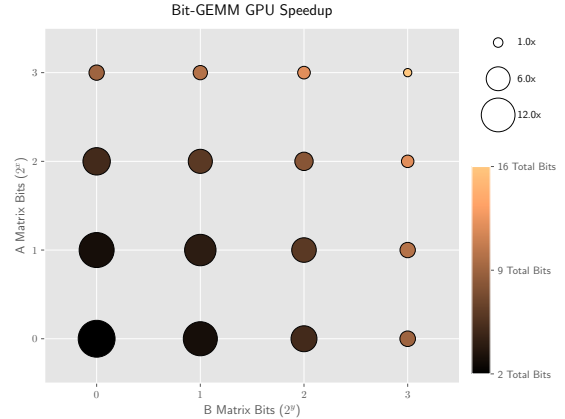

Figure 1: GPU-based bit-GEMM speedup for low precision matrices A and B. Results are compared with a similarly optimized 32-bit GEMM kernel, and run on a NVIDIA Tesla V100 GPU.

efficiently allocate precision to maximize model performance? We seek to answer this question by parametrizing the precision at each layer and learning these additional parameters by gradient descent.

3.1 LEARNING PRECISION

The selection of precision for variables in a model can have a significant impact on performance. Consider the DoReFa-Net model — if we decide on a budget of the total number of bits to assign to the weights layer-wise, and train the model under a number of different manual allocations, we obtain the training curves in Figure 2. For each training curve, the number of bits assigned to each layer's weights are indicated by the integer at the appropriate position (e.g. **444444** indicates 6 quantized layers, all assigned 4 bits).

Varying the number of bits assigned to each layer can cause the error to change by up to several percent, but the best configuration of these bits is unclear without exhaustively testing all possible configurations. This motivates *learning* the most efficient allocation of precision to each layer. However, this is a difficult task for two important reasons:

- unconstrained parameters that control precision will only grow, as higher precision leads to a reduction of loss; and

- quantization involves discrete operations, which are non-differentiable and therefore unsuitable for naïve backpropagation.

The first issue is easily addressed by fixing the total net-

Table 1: DoReFa-Net single-crop top-1 validation error for common weight (W), activation (A), and gradient (G) quantization configurations on the ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC12) dataset. The results are slightly improved over the results originally reported and were reproduced by us based on the public DoReFa-Net (Zhou et al., 2016) codebase.

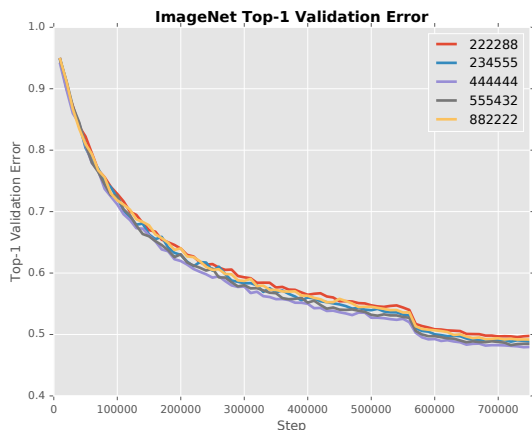| Model	W	A	G	Top-1 Validation Error
AlexNet (Krizhevsky et al., 2012)	32	32	32	41.4%
BWN (Courbariaux and Bengio, 2016)	1	32	32	44.3%
DoReFa-Net (Zhou et al., 2016)	1	2	6	47.6%
DoReFa-Net (Zhou et al., 2016)	1	2	4	58.4%

Figure 2: The training error for a variety of DoReFa-Net manual precision allocations is plotted for each weight update. The uniform distribution of bits (i.e. **444444**) leads to the lowest error, while the least uniform configurations (e.g. **222288**, **882222**) lead to the highest error.

work precision (i.e. the sum of the bits of precision at each layer) to a budget B, similar to the $B = 24$ configurations seen in Figure 2. The task is then to learn the allocation of precision across layers. The second issue: the non-differentiable nature of quantization operations is an unavoidable problem, as transforming continuous values into discrete values must apply some kind of discrete operator. We avoid this issue by employing a kind of stochastic allocation of precision and rely on recently developed techniques from the deep learning community to backpropagate gradients through discrete stochastic operations.

Intuitively, we can view the precision allocation procedure as sequentially allocating one bit of precision to one of L layers. Each time we allocate, we draw from a categorical variable with L outcomes, and allocate that bit to the corresponding layer. This is repeated B times to match the precision budget. An allocation of B bits corresponds to a particular precision configuration, and

we sample a new configuration for each input example. The idea of stochastically sampling architectural configurations is akin to Dropout (Srivastava et al., 2014), where each example is processed by a different architecture with tied parameters.

Different from Dropout, which uses a fixed dropout probability, we would like to parametrize the categorical distribution across layers such that we can learn to prefer to allocate precision to certain layers. Learning these parameters by gradient descent requires backprop through an operator that samples from a discrete distribution. To deal with its non-differentiability, we use the Gumbel-Softmax, also known as the Concrete distribution (Jang et al., 2016; Maddison et al., 2016), which, using a temperature parameter, can be smoothly annealed from a uniform continuous distribution on the simplex to a discrete categorical distribution from which we sample precision allocations. This allows us to use a high temperature at the beginning of training to stochastically explore different precision configurations and use a low temperature at the end of training to discretely allocate precision to network layers according to the learned distribution.

Though non-integer bits of precision can be implemented (detailed below), integer bits are more amenable to hardware implementations, which is why we aim to converge toward discrete samples. Table 2 shows examples of sampling from this distribution at different temperatures for a three class distribution. It should be noted that in order to perform unconstrained optimization we parametrize the unnormalized logits (also known as log-odds) instead of the probabilities themselves.

Since the class logits π_i control the probability of allocating a bit of precision to a network layer l_i, at low temperatures the one-hot samples will allocate bits of precision to the network according to the learned parameters π_i. However, at high temperatures we allocate partial bits to layers. This is possible due to our quantization straight-through estimator (STE), `quantize`, adopted

Table 2: Examples of single samples drawn from a Gumbel-Softmax distribution, parametrized by logits for three classes π_1, π_2, π_3 and a variety of temperatures τ. The probability of allocating a bit to layer l_i is impacted by the logit π_i, where higher values correspond to higher probabilities. The Gumbel-Softmax interpolates between continuous densities on the simplex (at high temperature) and discrete one-hot-encoded categorical distributions (at low temperature).

τ	Class 1 ($\pi_1 = 1.00$)	Class 2 ($\pi_2 = 2.00$)	Class 3 ($\pi_3 = -0.50$)
100.0	0.33	0.33	0.33
10.0	0.33	0.41	0.26
1.00	0.31	0.60	0.09
0.10	0.00	1.00	0.00

from (Zhou et al., 2016):

Forward:

$$r_o = \texttt{quantize}(r_i) = \frac{1}{2^k - 1}\text{round}\left((2^k - 1)r_i\right), \quad (2)$$

Backward: $\dfrac{\partial c}{\partial r_i} = \dfrac{\partial c}{\partial r_o} \qquad (3)$

where r_i is the real number input, r_o is the k-bit output, and c is the objective function. Since $\texttt{quantize}$ produces a k-bit value on $[0,1]$, quantizing to non-integer values of k simply produces a more fine-grained range of representation compared to integer values of k. This is demonstrated in Table 3.

Table 3: The possible output values of the $\texttt{quantize}$ operation are shown for a variety of values of k, clipped between 0 and 1. Non-integer values of k provide a more fine-grained range of representation between successive integer values (underlined).

k	1	1.50	2	2.25	2.50	2.75	3
	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	1.00	0.55	0.33	0.26	0.21	0.17	0.14
		1.00	0.66	0.53	0.42	0.34	0.28
			1.00	0.80	0.64	0.52	0.42
				1.00	0.85	0.69	0.57
					1.00	0.87	0.71
						1.00	0.85
							1.00

Using real values for quantization also provides useful gradients for backpropagation, whereas the small finite set of possible integer values would yield zero gradient almost everywhere.

3.2 PRECISION ALLOCATION LAYER

We introduce a new layer type, the precision allocation layer, to implement our precision learning technique. This layer is inserted as a leaf in the computational graph, and is executed on each forward-pass of training. The precision allocation layer is parametrized by the learnable class probabilities π_i, which define the Gumbel-Softmax distribution. Each class probability is associated with a layer l_i, so the samples assigned to each class are allocated as bits to the appropriate layer. This is illustrated in Figure 3 and stepped through in Example 1.

It should be noted that during the early stages of training before the network performance has converged, allowing the temperature to drop too low results in high-variance gradients while also encouraging largely uneven allocations of bits. This severely hurts the generalization of the network. To deal with this, we empirically observe a temperature where the class probabilities have sufficiently stabilized, and perform hard assignments of bits of precision based on these stabilized class probabilities. To do this, we sample from the Gumbel-Softmax a large number of times and average the results, in order to converge on the expected class sample assignments. Once we have these precision values, we fix the layers at these precision values for the remainder of training. We observe that the regularization effects of stochastic bit allocations are most useful during early training, and performing hard assignments greatly improves generalization performance. For all experiments considered, we implement a hard assignment of bits after the temperature drops below 3.0.

4 EXPERIMENTS

We evaluate the effects of our precision layer on two common image classification benchmarks, MNIST and ILSVRC12. We consider two separate CNN architectures, a 5-layer network similar to LeNet-5 (Lecun et al., 1998) trained on MNIST, and the AlexNet network (Krizhevsky et al., 2012) trained on ILSVRC12. We use the top-1 single-crop error as a measure of performance, and quantize both the weights and activations in all experiments considered. As in previous works (Zhou et al., 2016), the first layer of AlexNet was not quantized. Initial experiments showed that the effects of learning precision were less beneficial to gradients, so we leave them at full precision in all reported experiments.

Since our motivation is to show the precision layer as an improvement over uniformly quantized low precision models, we compare our results to networks with evenly distributed precision over all layers. We consider three common low precision budgets as powers of
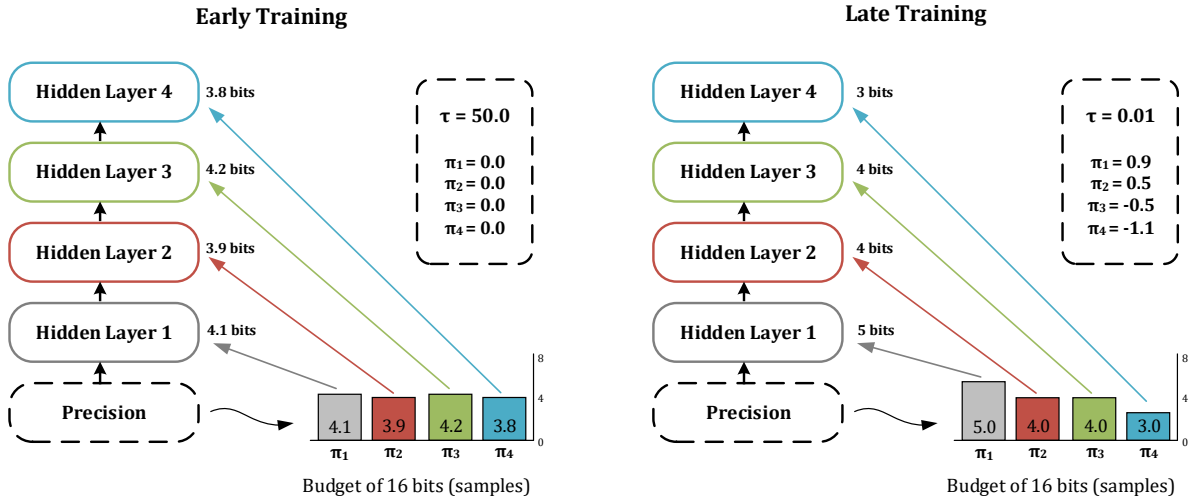
Early Training

Hidden Layer 4 — 3.8 bits

Hidden Layer 3 — 4.2 bits

Hidden Layer 2 — 3.9 bits

Hidden Layer 1 — 4.1 bits

Precision

$\tau = 50.0$

$\pi_1 = 0.0$
$\pi_2 = 0.0$
$\pi_3 = 0.0$
$\pi_4 = 0.0$

4.1 — 3.9 — 4.2 — 3.8

π_1 π_2 π_3 π_4

Budget of 16 bits (samples)

Late Training

Hidden Layer 4 — 3 bits

Hidden Layer 3 — 4 bits

Hidden Layer 2 — 4 bits

Hidden Layer 1 — 5 bits

Precision

$\tau = 0.01$

$\pi_1 = 0.9$
$\pi_2 = 0.5$
$\pi_3 = -0.5$
$\pi_4 = -1.1$

5.0 — 4.0 — 4.0 — 3.0

π_1 π_2 π_3 π_4

Budget of 16 bits (samples)

Figure 3: Early in training, the Gumbel-Softmax class probabilities π_i are initialized to 0 while the temperature τ is high, generally resulting in uniform allocations of real-valued bits of precision. Later in training, with a low temperature, the learned class probabilities usually result in some layers being allocated more or less discrete bits of precision. In practice, the computational overhead of the precision layer is not noticeable.

Example 1. Consider a network with 4 layers, each denoted by l_i. To learn the precision of these layers, we add a precision layer which constructs a Gumbel-Softmax distribution with 4 classes π_i, where each class is assigned to a layer. Each class probability is initalized to 0.0, and the temperature is initialized to 50.0. For a budget of 16 total bits, two example iterations representative of early training (first iteration of epoch 0) and late training (first iteration of epoch 50) are shown below:

Epoch = 0, $\pi_1 = 0.0$, $\pi_2 = 0.0$, $\pi_3 = 0.0$, $\pi_4 = 0.0$, $\tau = 50.0$

- The precision layer is executed first, which means we sample from the Gumbel-Softmax 16 times because our budget is 16 bits, and accumulate the results of the 16 samples. Each individual sample gives us 4 class outputs which sum to 1 (e.g. [0.25, 0.25, 0.25, 0.25]), so sampling 16 times and accumulating the results for each class means our final results will add to 16. Since the class probabilities are initialized to 0.0, and the temperature is high, the expected samples will be continuous and relatively uniform across all classes.
    - The samples associated with each layer l_i are: $l_1 = 4.1$, $l_2 = 3.9$, $l_3 = 4.2$, $l_4 = 3.8$.
    - We now assign 4.1 bits to layer 1, 3.9 bits to layer 2, 4.2 bits to layer 3, and 3.8 bits to layer 4. These bits are assigned to the appropriate layer by applying the `quantize` operation of Equation 2 to the desired parameters (e.g. weights) with k as the appropriate bit assignment (e.g. $k = 4.1$ for l_1).
    - The `quantize` operation will transform the layer parameters to one of several discrete positive quantities between 0 and 1 (see Table 3). Though these are fractional bit assignments, the `quantize` operations works the same as if these were integer bit assignments.
    - The iteration then proceeds as normal, with the quantized parameters and class probabilities π_i updated during back-propagation.

Epoch = 50, $\pi_1 = 0.9$, $\pi_2 = 0.5$, $\pi_3 = -0.5$, $\pi_4 = -1.1$, $\tau = 0.01$

- Similar to before, we begin by sampling from the Gumbel-Softmax 16 times because our budget is 16 bits, and accumulate the results. However, the class probabilities have now changed such that π_1 corresponds to the most likely class, and π_4 the least likely class, so the distribution is no longer uniform. As well, since the temperature is low ($\tau = 0.01$), the samples will now approach discrete.
    - Samples: $l_1 = 5.0$, $l_2 = 4.0$, $l_3 = 4.0$, $l_4 = 3.0$.
    - We now assign 5.0 bits to layer 1, 4.0 bits to layer 2, 4.0 bits to layer 3, and 3.0 bits to layer 4, similar to before. Since these bit assignments are integer values, the activity of the `quantize` operation is more intuitive.
    - Similar to before, the `quantize` operation will transform the layer parameters to one of several discrete positive quantities between 0 and 1 (see Table 3).
    - The forward-pass then proceeds as normal, with the quantized parameters and class probabilities π_i updated during back-propagation. Since this is late training, parameter updates will be smaller in magnitude than in early training.
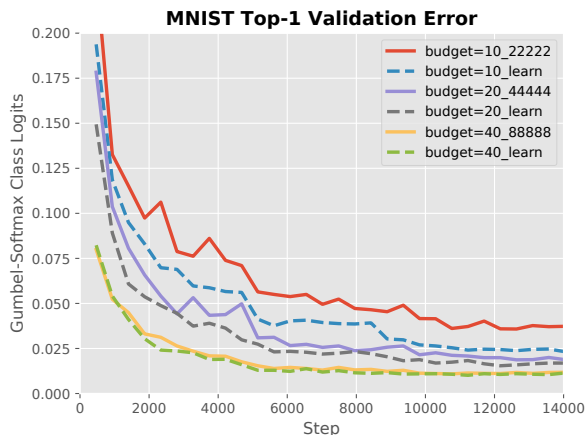
MNIST Top-1 Validation Error

Figure 4: The top-1 errors for the MNIST networks are shown. Both baseline networks with uniform precision allocation (i.e. **budget=10_22222**) and learned precision networks (i.e. **budget=10_learn**) are assigned the same total precision budget, indicated by the prefix.

2, which would provide efficient hardware acceleration, where each layer is allocated 2, 4, or 8 bits. Baseline networks with uniform precision allocation are denoted with the allocation for each layer and the budget (e.g. **budget=10_22222** denotes a baseline network with 2 bits manually allocated to each of 5 layers for a budget of 10 bits), while the networks with learned precision are denoted with the precision budget and *learn* (e.g. **budget=10_learn** denotes a learned precision network with a budget of 10 total bits, averaging 2 bits per layer).

4.1 MNIST

The training curves for the MNIST-trained models are shown in Figure 4. The learned Gumbel-Softmax class logits are shown in Figure 5.

We observe that the models with learned precision converge faster and reach a lower test error compared to the baseline models across all precision budgets considered, and the relative improvement is more substantial for lower precision budgets. From Figure 5, we observe that the models learn to assign fewer bits to early layers of the network (conv0, conv1) while assigning more bits to later layers of the network (conv2, conv3), as well as preferring smoother (i.e. more uniform) allocations. This result agrees with the empirical observations of (Raghavan et al., 2017). The results on MNIST are summarized in Table 4. Uncertainty is calculated by averaging over 10 runs for each network with different random initializations of the parameters.

4.2 ILSVRC12

The results for ILSVRC12 are summarized in Table 5. Again, models that employ stochastic precision allocation converge faster and ultimately reach a lower test error than their fixed-precision counterparts on the same budget. We observe that networks trained with stochastic precision learn to take bits from early layers and assign these to later layers, similar to the MNIST results. While the class logits for the MNIST network were similar to the ILSVRC12 results, they were not substantial enough to cause changes in bit allocation during our hard assignments. However, the ILSVRC12-trained networks actually make non-uniform hard assignments. This suggests that the precision layer has a larger affect on more complex networks.

5 CONCLUSION AND FUTURE WORK

We introduced a precision allocation layer for DNNs, and proposed a stochastic allocation scheme for learning precision on a fixed budget. We have shown that learned precision models outperform uniformly-allocated low precision models. This effect is due to both learning the optimal configuration of precision layer-wise, as well as the regularization effects of stochastically exploring different precision configurations during training. Moreover, the use of precision budgets allow a high level of hardware acceleration determinism which has practical implications.

While the present experiments were focused on accuracy rather than computational efficiency, future work will examine using GPU bit kernels in place of the full precision kernels we used in our experiments. We also intend to investigate stochastic precision in the adversarial setting. This is inspired by Galloway et al. (2018), who report that stochastic quantization at test time yields robustness towards iterative attacks.

Finally, we are interested in a variant of the model where rather than directly parametrizing precision, precision is conditioned on the input. While this reduces hardware acceleration determinism in real-time or memory-constrained settings, it would enable a DNN to dynamically adapt its precision configuration to individual examples.

References

E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool. Soft-to-hard vector quantization for end-to-end learned compression of images and neural networks. *arXiv preprint arXiv:1704.00648*, 2017.
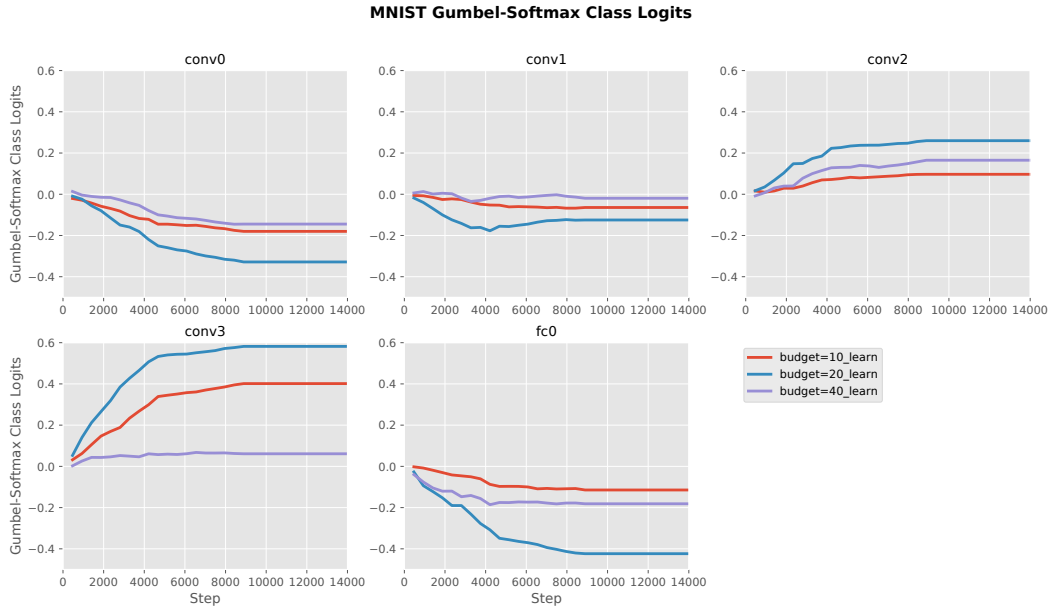
MNIST Gumbel-Softmax Class Logits

Figure 5: The learned Gumbel-Softmax class logits for different precision budgets.

Table 4: Final training results and top-1 error for the MNIST baseline and learned precision models.

Network	Precision Budget	Val. Error	Final Bit Allocation				
			l_1	l_2	l_3	l_4	l_5
budget=10_22222	10 Bits	$3.73\% \pm 0.3$	2	2	2	2	2
budget=10_learn	10 Bits	$2.32\% \pm 0.3$	2	2	2	2	2
budget=20_44444	20 Bits	$1.88\% \pm 0.2$	4	4	4	4	4
budget=20_learn	20 Bits	$1.69\% \pm 0.2$	4	4	4	4	4
budget=40_88888	40 Bits	$1.18\% \pm 0.1$	8	8	8	8	8
budget=40_learn	40 Bits	$1.14\% \pm 0.1$	8	8	8	8	8

Table 5: Final training results and top-1error for the ILSVRC12 baseline and learned precision models.

Network	Precision Budget	Val. Error	Final Bit Allocation						
			l_1	l_2	l_3	l_4	l_5	l_6	l_7
budget=14_2222222	14 Bits	$49.68\% \pm 0.3$	2	2	2	2	2	2	2
budget=14_learn	14 Bits	$48.54\% \pm 0.3$	2	2	2	2	2	2	2
budget=28_4444444	28 Bits	$47.99\% \pm 0.3$	4	4	4	4	4	4	4
budget=28_learn	28 Bits	$47.69\% \pm 0.3$	3	3	4	4	5	5	4
budget=56_8888888	56 Bits	$47.70\% \pm 0.3$	8	8	8	8	8	8	8
budget=56_learn	56 Bits	$47.46\% \pm 0.3$	7	7	8	8	9	9	8

S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *arXiv preprint arXiv:1702.00953*, 2017.

A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

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

M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

A. Galloway, G. W. Taylor, and M. Moussa. Attacking binarized neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015b.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144*, 2016.

X. Jin, X. Yuan, J. Feng, and S. Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

M. Kim and P. Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

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

F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016a.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016b.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

A. Raghavan, M. Amer, S. Chai, and G. W. Taylor. BitNet: Bit-regularized deep neural networks. *arXiv preprint arXiv:1708.04788*, 2017.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

F. Tung, S. Muralidharan, and G. Mori. Fine-Pruning: Joint Fine-Tuning and compression of a convolutional network with bayesian optimization. In *British Machine Vision Conference (BMVC)*, 2017.

L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect.

In *International Conference on Learning Representations (ICLR)*, pages 1058–1066, 2013.

X. Wang and J. Liang. Scalable compression of deep neural networks. In *Proceedings of the ACM on Multimedia Conference*, pages 511–515, 2016.

P. Yin, S. Zhang, Y. Qi, and J. Xin. Quantization and training of low Bit-Width convolutional neural networks for object detection. *arXiv preprint arXiv:1612.06052*, 2016.

S. Zhou, 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.

C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.