ReLeQ: An Automatic Reinforcement Learning Approach for Deep Quantization of Neural Networks

Amir Yazdanbakhsh[*†] **Ahmed T. Elthakeb**[*] **Prannoy Pilligundla**
FatemehSadat Mireshghallah **Hadi Esmaeilzadeh**
Alternative **C**omputing **T**echnologies (**ACT**) Lab
[†]Google Brain University of California, San Diego
ayazdan@google.com a1yousse@eng.ucsd.edu ppilligu@eng.ucsd.edu
fmireshg@eng.ucsd.edu hadi@eng.ucsd.edu

Abstract

Despite numerous state-of-the-art applications of Deep Neural Networks (DNNs) in a wide range of real-world tasks, two major challenges hinder further advances in DNNs: hyperparameter optimization and constrained power resources, which is a significant concern in embedded devices. DNNs become increasingly difficult to train and deploy as they grow in size due to both computational intensity and the large memory footprint. Recent efforts show that quantizing weights of deep neural networks to lower bitwidths takes a significant step toward mitigating the mentioned issues, by reducing memory bandwidth and using limited computational resources which is important for deploying DNN models to devices with limited resources. This paper builds upon the algorithmic insight that the bitwidth of operations in DNNs can be reduced without compromising their classification accuracy. Deep quantization (quantizing bitwidths below eight) while maintaining accuracy, requires magnificent manual effort and hyper-parameter tuning as well as re-training. This paper tackles the aforementioned problems by designing an end to end framework, dubbed ReLeQ, to automate DNN quantization. We formulate DNN quantization as an optimization problem and use a state-of-the-art policy gradient based Reinforcement Learning (RL) algorithm, Proximal Policy Optimization (PPO) to efficiently explore the large design space of DNN quantization and solve the defined optimization problem. To show the effectiveness of ReLeQ, we evaluated it across several neural networks including MNIST, CIFAR10, SVHN. ReLeQ quantizes the weights of these networks to average bitwidths of 2.25, 5 and 4 respectively while maintaining the final accuracy loss below 0.3% .

1 Introduction

Deep Neural Networks (DNN) have been widely adopted for numerous applications such as pattern recognition, video analytics, data classification, etc. DNN applications are constrained by the vast amount of compute intensive multiply-accumulate operations. In turn, energy consumption and latency of DNN systems can become extremely large. Specifically, training is shown to be not suitable for execution on general purpose embedded processors.

In many applications, however, performing inference on such embedded or memory constrained devices is a requirement. As an example, personal assistants such as Amazon's Alexa or Apple's Siri,

[*]Both authors contributed equally to this work.
[†]Amir Yazdanbakhsh is a Google AI resident.

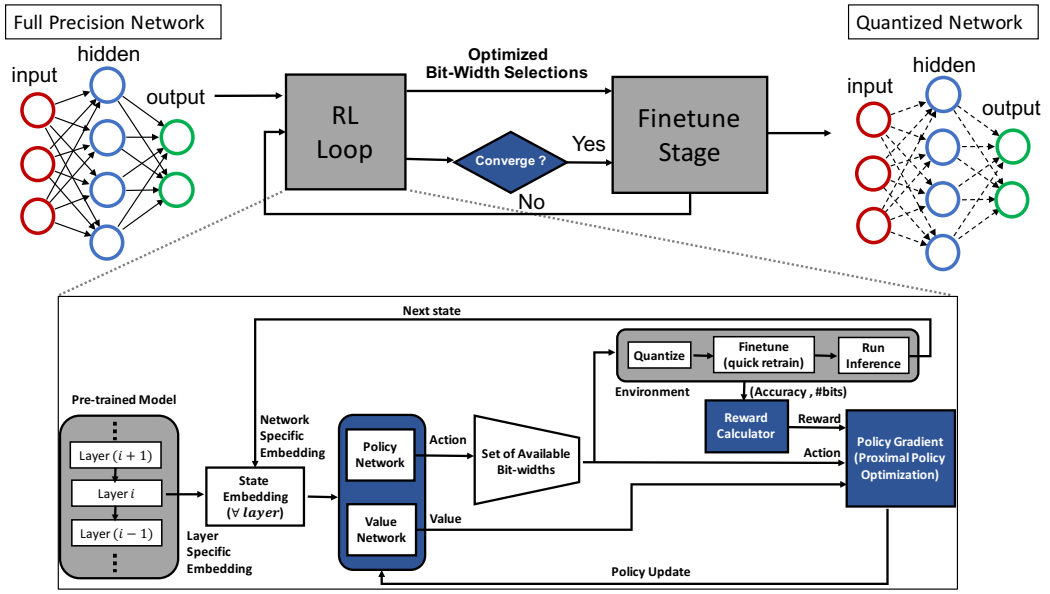


Figure 1: Overview of ReLeQ framework.

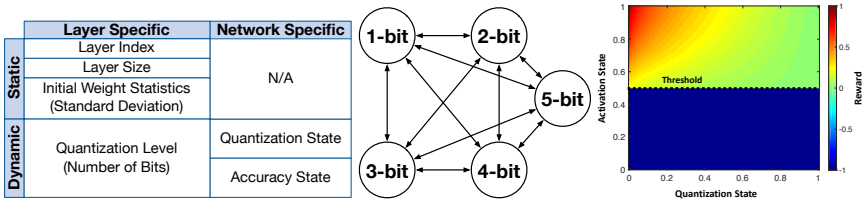
which need to perform speech recognition and employ conversational networks using an embedded system. For these types of applications of DNNs, it is important to decrease the amount of data and computation required for inference to avoid the high costs of data transfer. One prominent approach to this problem is quantizing the parameters of DNNs [1] which utilize low precision weights and activations to reduce compute requirements.

In most quantization approaches for DNNs, all layers of a given network are quantized to the same bitwidth. However, with each layer in a neural network playing different roles and having unique properties in terms of weight distribution, over-quantizing an important layer can result in unnecessary pressure on subsequent layers to maintain accuracy. Such pressure leads to longer re-training and fine tuning times, as well as potentially sub-optimal accuracy due to the fact that the sensitivity of layers to accuracy is not being accounted for.

To solve this issue, different combinations of quantization bitwidths can be tested for each layer of a DNN. However, evaluating different quantization combinations requires iterating over a prohibitively large search space, making it impractical to exhaustively assess and determine the optimum quantization level for each layer. As such, this paper sets out to automate and speedup the search process in deep quantization for neural networks using reinforcement learning. We propose a framework, dubbed ReLeQ, that learns the sensitivity of final classification accuracy with respect to the quantization level of each layer's weights , determining the bitwidth of each layer while keeping classification accuracy within a percent of the full-precision accuracy. Figure 1 shows an overview of the proposed ReLeQ framework. Observing that the quantization bitwidth for a given layer in a neural network affects the accuracy of subsequent layers, our framework implements an architecture which allows it to take actions (e.g., select quantization levels) with the context of previous layers' bitwidths. The ReLeQ framework receives a full precision network and outputs a quantized network after multiple iterations of reinforcement learning training. We show that ReLeQ agent is effective in distinguishing the sensitivity of individual layers while quantizing the entire network to the minimum bitwidth and minimizing the loss in accuracy.

2 Methodology: Reinforcement Learning for Quantization

To determine the quantization level for each layer of a neural network, we train a reinforcement learning agent which explores the search space of quantization levels for each layer within a neural network. The agent utilizes a reward function which seeks to minimize the average bitwidth of the neural network layers while minimizing the accuracy loss relative to full-precision accuracy.



(a) State Embedding (b) Action Space (c) Reward Function

Figure 2: Reinforcement formulation summary: (a) state embedding, (b) action space; showing the flexibility of moving across different bitwidths, (c) reward shaping; on the X-axis the quantization state is displayed and on the Y-axis the accuracy state is shown. The color indicates intensity of reward.

The ReLeQ agent is trained by determining the quantization bitwidth (q_b), which belongs to the set $\{1, 2, 3, 4, 5\}$, of each layer in order, taking into account previous layers' quantization at each step. After each step, the ReLeQ agent receives a reward which is determined by both the accuracy of inference for the current quantization level and average of quantized bitwidths (more details in Sec 2.3). Therefore, the reward function encourages both accuracy preservation and reduction of overall bitwidth. With accuracy preservation being a primary component of ReLeQ's reward function, retraining of a quantized neural network is required in order to properly evaluate the effectiveness of deep quantization. Such re-training is a time-intensive process, which undermines the search process efficiency. To mitigate this issue, we reward the agent with an estimated accuracy after re-training for a shortened amount of epochs which may or may not have converged to a final accuracy. After the agent has converged quantization levels for each layer within the network, we perform a retraining step on the quantized model and then obtain the final accuracy. Figure 1 shows an overview of the proposed framework, ReLeQ. In the following sections, we overview each stage of the ReLeQ framework.

2.1 State Space

The ReLeQ agent quantizes each layer based on the sensitivity of the final classification accuracy relative to the quantization of a layer. Determining the accuracy sensitivity for a given DNN layer requires knowledge of previous layers' bitwidths, layer indices, layer sizes, and statistics (e.g., standard deviation) about the distribution of weights. As such, we use an LSTM network to consider this dependency between layers. Figure 2a shows the state space of ReLeQ agent categorized as follows: (i) Layer-specific parameters which are unique per layer vs. network-specific parameters which characterize the entire network as the agent steps forward during training process. (ii) Static parameters that do not change during training process. Dynamic parameters that change during training process depending on the actions taken by the agent while it explores the design space. Lastly, in addition to the layers' characteristics, the state parameters reflect some indication about the quantization and accuracy states which are defined as follows

$$Quantization\ State = \frac{\sum_{i=1}^{Number\ of\ Layers} bitwidth_i}{Max\ Bitwidth \times Number\ of\ Layers}$$

$$Accuracy\ State = \frac{Current\ Accuracy}{Full\ Precision\ Accuracy}$$

2.2 Action Space

We provide a discrete set of quantization bitwidths from which the ReLeQ agent chooses in order to obtain a reward. Specifically, the set of bitwidths used for our implementation was 1,2,3,4,5, but any set of bitwidths can be passed based on the desired quantization levels.

2.3 Reward Formulation

Reward formulation is a critical component of any RL problem and the convergence behavior highly depends on it. We define the reward as a function of two terms, (a) accuracy state, (b) quantization

state. The following snippet shows the formulation of ReLeQ's reward function and Figure 2c shows a visualization.

Reward Shaping:
$reward = 1 - (quantization\_state)^a$
if $(accuracy\_state < threshold)$ **then**
  $reward = -1$
else
  $accuracy\_discount = max(accuracy\_state, th)^{(b/max(accuracy\_state, th))}$
  $reward = reward \times accuracy\_discount$
end if

In the above function, a and b are hyperparameters and th is accuracy state threshold. Formulating the reward in this manner produces a smooth reward gradient as the agent approaches the optimum quantization combination. In addition, the varying 2-dimensional gradient speeds up the agent's convergence time. Task completion is further reduced by setting a threshold to prevent unnecessary or undesirable exploration. The threshold also enables the agent to explore more relevant regions within the design space.

2.4 Learning Procedure

As shown in Figure 1, the agent steps through all layers one by one, determining the quantization level for the layer at each step. After every action for a given step, we perform a shortened re-training process, and use the resulting validation accuracy in formulating the reward. To encourage lower bitwidth quantization and therefore reduced cost, the RL agent factors the average quantization across all layers into the reward in combination with the aforementioned re-trained accuracy. A positive accuracy reward is determined by an improved or maintained validation accuracy relative to the full precision accuracy, whereas the quantization reward is positive if the agent reduces the bitwidth of a layer. The interplay between accuracy reward and quantization reward enables the agent to deeply quantize the network with minimal loss in accuracy.

2.5 Policy and Value Networks

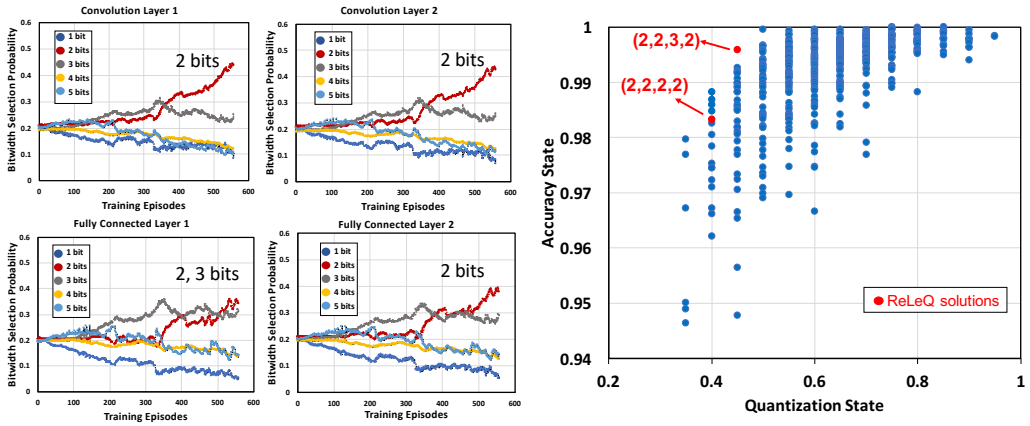
The agent is comprised of two networks, a policy network and a value network. All the state embeddings are fed as input to the LSTM layer and this acts as the first hidden layer for both policy and value networks. Based on our evaluations, LSTM enables the ReLeQ agent to converge almost $\times 1.33$ faster than without LSTM. Apart from the LSTM, policy network has two fully connected hidden layers of 128 neurons each and the number of neurons in the final output layer is equal to the number of available bitwidths the agent can choose from. Whereas the Value network has two fully connected hidden layers of 128 and 64 neurons each. We use Proximal Policy Optimization (PPO) [2], a state-of-the-art approach in reinforcement learning, to update policy and Value networks. We implemented all the networks in TensorFlow framework and used Adam Optimizer with an initial learning rate of 10^{-4} to minimize the loss function. Details of the parameters used can be seen in Table1.

Table 1: PPO Hyperparameters used in ReLeQ.

Hyperparameter	Value
Adam Step Size	1×10^{-4}
Discount Factor	0.9
GAE Parameter	0.99
Number of Epochs	3
Clipping Parameter	0.3

3 Evaluation

We evaluated our approach on several neural networks for different datasets: MNIST, CIFAR10, SVHN. The ReLeQ agent quantizes these networks to average bitwidths of 2.25, 5 and 4 respectively



(a) (b)

Figure 3: (a) Evolution of the ReLeQ agent bitwidth selection probabilities on MNIST dataset. (b) Design space enumeration highlighting the effectiveness of ReLeQ solutions.

Table 2: Summary of quantization for different networks. The MNIST[3], CIFAR-10[4] and SVHN[4] dataset and networks have been used for benchmarking ReLeQ.

Dataset	Network Architecture		Average Quantized Bitwidth for Weights	Network Accuracy	
	# of Conv. Layers	# of FC Layers		Baseline	ReLeQ
MNIST	2	2	2.25	99.02%	98.73%
CIFAR-10	2	2	5	86.00%	85.70%
SVHN	8	2	4	93.33%	93.33%

with less than 0.3% accuracy loss in all cases. Table 2 provides a summary for the evaluated datasets, networks and the results with respect to quantization and the classification accuracy after deep quantization. In the following subsections we provide details and analysis of the conducted experiments and the results.

3.1 MNIST Analysis

Figure 3a shows the evolution of ReLeQ agent's bitwidth selection probabilities for all layers over time (number of episodes), which reveals how the agent's policy changes with respect to selecting a bitwidth per layer. An episode is defined as a single pass through all layers in the neural network. As indicated on the graph, the end results suggest the following quantization patterns, $(2, 2, 2, 2)$ or $(2, 2, 3, 2)$ bits. For the first two convolution layers (Convolution Layer 1, Convolution Layer 2), the agent ends up assigning the highest probability for two bits and its confidence increases with increasing number of training episodes. For the third layer (Fully Connected Layer 1), the probabilities of two bits and three bits are very close. Lastly, for the fourth layer (Fully Connected Layer 2), the agent again tends to select two bits, however, with relatively smaller confidence compared to layers one and two.
With these observations, we can infer that bitwidth probability profiles are not uniform across all layers and that the agent distinguishes between the layers, understands the sensitivity of the layers to accuracy and chooses the bitwidths accordingly. To assess the solution quality of the RL agent, since LeNet is a small network, we enumerate all the possible design points over the available bitwidths for the agent. As shown in Figure 3b, ReLeQ agent solution (highlighted in red) achieves a good tradeoff between accuracy state and quantization state. This demonstrates the efficiency in exploring the underlying design space.

Looking at the agent's selection for the third layer (fc1) and recalling the initial problem formulation of quantizing all layers while preserving the initial full precision accuracy, it is logical that the probabilities for two and three bits are very close. Going further down to two bits was beneficial in terms of quantization while staying at three bits was better for maintaining good accuracy. *This points out the importance of tailoring the reward function and the role it plays in controlling such trade-offs.*

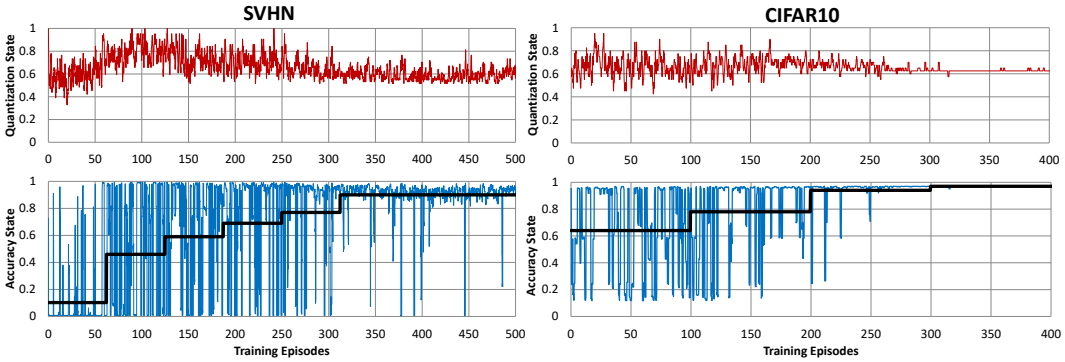
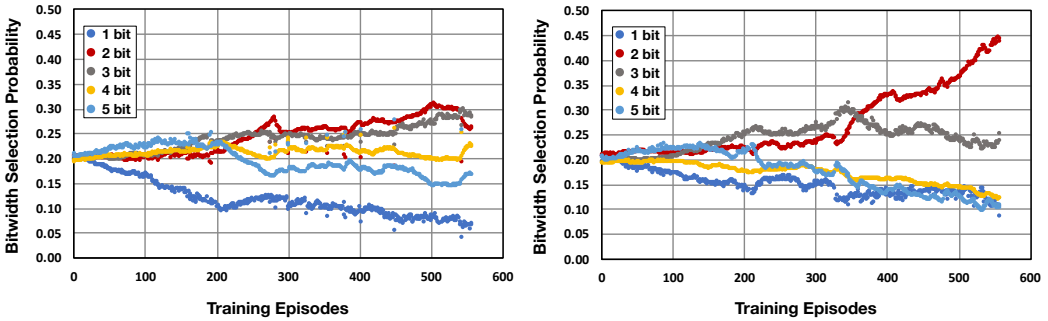


Figure 4: ReLeQ agent performance on SVHN and CIFAR-10 quantization. Solid lines indicate the moving average.



(a) Excluding weight variance in the state embedding (b) Including weight variance in the state embedding

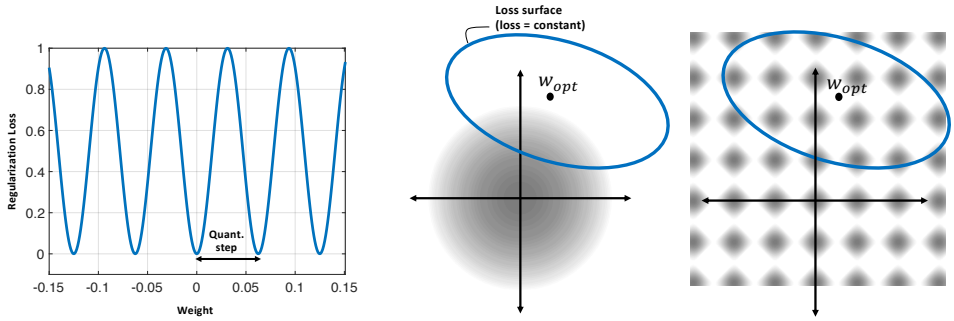
Figure 5: Comparison of ReLeQ convergence performance for LeNet (convolution layer 1) on MNIST dataset (a) when weight variances are excluded from the state embedding and (b) when weight variances are included in the state embedding .

3.2 CIFAR-10 and SVHN Analysis

A major indicator of the correctness of a formulated reinforcement learning problem is the ability of the agent to consistently yield improved solution quality as it learns the correct underlying policy and gradually transition from the exploration to the exploitation phase. Figure 4 shows the accuracy state (normalized accuracy) and quantization state for CIFAR-10 and SVHN experiments. We overlay the moving average of the accuracy state for the agent's solutions as episodes evolve. As the graph shows, the agent consistently yields solutions that increasingly preserve the accuracy, while seeking to minimize the number of bits assigned to each layer (minimizing the quantization state) at the same time and eventually converges to a stable optimum solution. For the SVHN experiment, we run multi-digit recognition (five digits with ten classes for each digit) using a ten layer network. During the experiment, we set the first and last layers as full precision (excluding them from the search space). In terms of convergence analysis, it can be noticed that the agent in both experiments manages to converge in around 300 episodes. Despite the fact that SVHN experiment (considering eight layers for quantization) is relatively larger than the CIFAR-10 experiment (considering five layers for quantization), the convergence speed in both experiments is comparable (slightly faster for the CIFAR-10). Such convergence behavior demonstrates the agent's ability to efficiently explore huge search spaces for much deeper and wider networks. For CIFAR-10 and SVHN experiments, the agent eventually converged to same bitwidth for all layers (all five bits for CIFAR-10 and all four bits for SVHN).

3.3 Evaluation of State Embeddings

State embeddings are an essential component in determining and controlling the convergence behavior of the RL agent. We carried out experiments to evaluate the effectiveness of the used state embeddings.



(a) (b) (c)

Figure 6: Quantization friendly regularization.

In every iteration, we repeat the same experiment while omitting one state embedding at a time and keeping the rest. Based on our experiments, we observe that the layer's size/dimensions turn out to be the most important embedding for convergence. Standard deviation of the layer's weights comes as the second most important parameter. Figure 5a shows convergence behavior for LeNet with variance of weights as part of the state embedding (baseline result) and Figure 5b shows the behavior when variance of weights was excluded from the state embedding. The rest of the state embeddings help in speeding up the convergence process.

4 Accelerated Finetuning

Loss function of neural networks. Neural networks' loss functions are known to be highly non-convex and generally very poorly understood. It has been empirically verified that loss surfaces for large neural networks have many local minima that are essentially equivalent in terms of test error [5], [6]. Moreover, converging to one of the many good local minima proves to be more useful as compared to struggling to find the global minimum of the accuracy loss on the training set (which often leads to overfitting). This opens up and encourages a possibility of adding extra custom objectives to optimize for (during the training process), in addition to the original one (accuracy loss). The added custom objective could be with the purpose of increasing generalization performance or imposing some preference on the weights values. Regularization is one of the major techniques that makes use of such facts as discussed in the following subsection.

Quantization friendly regularization. Neural networks often suffer from redundancy of parameterization and consequently they commonly tend to overfit. Regularization is one of the commonly used techniques to enhance generalization performance of neural networks. Regularization effectively constrains weight parameters by adding a term (regularizer) to the objective function that captures the desired constraint in a soft way. This is achieved by imposing some sort of preference on weights updates during the optimization process. As a result, regularization seamlessly leads to unconditionally constrained optimization problem instead of explicitly constrained which, in most cases is much more difficult to solve. The most commonly used regularization technique is known as *weight decay*, which aims to reduce the network complexity by limiting the growth of the weights. It is realized by adding a term to the objective function that penalizes large weight values [7]:

$$E(w) = E_o(w) + \frac{1}{2}\lambda \sum_i {w_i}^2$$

where E_o is the original loss measure, and λ is a parameter governing how strongly large weights are penalized. w is a vector containing all free parameters of the network. Here, we propose a new type of regularization that is friendly to quantization. The proposed regularization is realized by adding a periodic function (regularizer) to the original objective function, Figure 6a. The periodic regularizer has a periodic pattern of minima that correspond to the desired quantization levels. Such correspondence is achieved by matching the period to the quantization step based on a particular

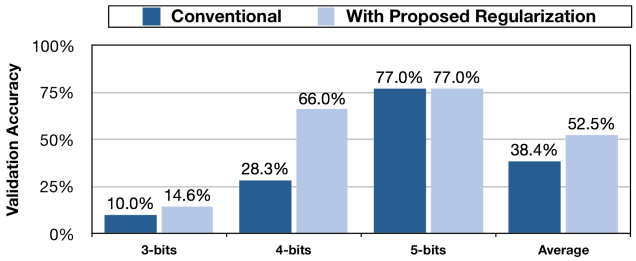


Figure 7: Comparison of validation accuracy of CIFAR-10 after retraining for 10k steps at different quantization bitwidths for weights (assuming same bitwidth for all layers).

number of bits for a given layer.

$$E(w) = E_o(w) + \frac{1}{2}\lambda_q \sum_i \sin^2\left(\frac{\pi w_i}{2^{-qbits}}\right)$$

where E_o is the original loss measure, and λ_q is a parameter governing how strongly weight quantization errors are penalized. w is a vector containing all free parameters of the network.

For the sake of simplicity and clarity, Figures 6b and 6c depict a geometrical sketch for a hypothetical loss surface (original objective function to be minimized) and an extra regularization term in 2-D weight space. For weight decay regularization, in Figure 6b, the faded circular contours show that as we get closer to the origin, the regularization loss is minimized. Optimum solution is achieved by striking a balance between the original loss term and the regularization loss term. In a similar way, Figure 6c shows a representation of the proposed regularization. A periodic pattern of minima pockets is seen surrounding the original optimum point. The objective of the optimization problem is to find the best solution that is the closest to one of those minima pockets where weight values are nearly matching the desired quantization levels, hence the name quantization-friendly.

Evaluation. To demonstrate the effectiveness of the proposed regularization, Figure 7 shows a comparison of the validation accuracy for CIFAR-10 using conventional finetuning approach versus finetuning with the proposed approach. It can be noticed that, under same limited amount of retraining steps, medium number of bits seem to benefit the most. Observed performance improvements could differ across different networks and different durations of retraining. It should be noted that, using the proposed regularization approach, the weight updates occur in full precision as opposed to clipping the weights to the quantized values, yet the weights eventually converge to desired quantization levels.

5 Related Work

Extensive work [8, 9, 1, 10] has been done for quantizing neural networks. However, all of them assume a predetermined target bitwidth for quantization. On the other side, many studies have leveraged reinforcement learning algorithms in the context of hyperparameter optimization such as neural network architecture designing [11, 12]. Further, [13] employed reinforcement learning to automatically find the compression ratio for different layers. Compared to previous works, ours is the first study that combines the two approaches and proposes an automated reinforcement learning based approach for bitwidth selection in the context of neural network quantization.

6 Future Work and Conclusion

In this work, we introduce a framework for automatic deep quantization of neural networks using reinforcement learning, dubbed ReLeQ. To consider the effect of quantization of each layer on the final classification accuracy, we use a LSTM based network for the ReLeQ agent. The RL agent takes into account both the layer's sensitivity to overall accuracy and the quantized bitwidths of the preceding layers while predicting a quantization level for a layer. Our preliminary results across multiple neural networks show that ReLeQ is effective in deep quantization of neural networks with minimum degradation on the final classification accuracy. For future work, we are working

on providing more effective in-the-loop feedback for the RL agent to further expedite the deep quantization process for larger networks. The combination of our RL based quantization and a more efficient re-training methodology will enable us to achieve better accuracy for larger networks using fewer re-training passes. The proposed technique also opens up the possibility of performing heterogeneous quantization of the network (i.e., quantizing each layer to different bitwidth) as the reinforcement learning agent learns the sensitivity of each layer with respect to accuracy in order to perform quantization of the entire network.

References

[1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *JMLR*, vol. 18, no. 1, pp. 6869–6898, 2017.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[3] Y. LeCun, C. Cortes, and C. Burges, "MNIST Handwritten Digit Database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[4] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning*, vol. 2011, p. 5, 2011.

[5] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in *Artificial Intelligence and Statistics*, pp. 192–204, 2015.

[6] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets," in *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

[7] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," in *Neural Information Processing Systems*, pp. 950–957, 1991.

[8] 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*, 2015.

[9] M. Rastegari, 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, 2016.

[10] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[11] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv preprint arXiv:1611.01578*, 2016.

[12] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," *arXiv preprint arXiv:1611.02167*, 2016.

[13] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *European Conference on Computer Vision*, pp. 815–832, 2018.