VQN: Variable Quantization Noise for Neural Network Compression

Anonymous ACL submission

Abstract

Quantization refers to a set of methods that compress a neural network by representing its parameters with fewer bits. However, applying quantization to a neural network after training often leads to severe performance regressions. Quantization Aware Training (QAT) addresses this problem by applying simulated training-time quantization for the model to learn robustness to inference-time quantization. One key drawback of this approach is that quantization functions induce biased gradient flow through the network during backpropagation, thus preventing the network from best-fitting to the learning task. Fan et al. addressed this issue by proposing *Quant-Noise*, in which simulated quantization is applied to a fixed proportion, called the quantization noise rate, of parameters during training. Our study, Variable Quantization Noise (VQN)[1], builds upon their technique by exploring a variable quantization noise rate instead of a fixed one. We craft three candidate functions to vary noise rate during training and evaluate the variants with 3 datasets and 3 quantization schemes for each dataset. First, we report negative results on our hand-crafted candidate functions. Second, we observe somewhat positive results on a method, originally intended as an ablation study, of randomly varying the noise rate during training. This method outperforms Quant-Noise on two out of three quantization schemes for all three tested datasets. Moreover, on two of the datasets, this method at 4x compression matches or exceeds performance of even the uncompressed model. Future work should determine whether these unexpected results hold for more datasets and quantization schemes, as well as investigating other schemes for varying the noise rate during training.

[1]The code implementation is on GitHub

1 Introduction

Modern deep learning architectures are getting larger, with recent models spanning many billions of parameters [1]. By contrast, many NLP and SLP use-cases involve deployment to embedded devices (e.g. voice assistants, IoT devices), where massive models are prohibitively memory intensive and computationally expensive.

Model compression techniques aim to address this limitation by creating compact representations of models that can attain the same level of performance as their larger counterparts. For instance, pruning methods do so by removing extraneous weights and activations to produce sparse architectures with reduced parameter counts [2]. Our work is an investigation into another such compression technique: quantization. Rather than removing entire units as pruning does, scalar quantization (used in this work) shrinks networks by reducing the bit-widths of their parameters [3]. In addition to a reduced memory footprint, computations can, likewise, be sped up with the appropriate hardware accelerators [4]. Throughout this paper, we will refer to scalar quantization to n-bit integers as int$n$ quantization, e.g. "`int8` quantization."

We aim to improve upon Quant-Noise, a state-of-the-art (SotA) quantization method developed by Facebook AI Research (FAIR) [3]. Quant-Noise falls under the umbrella of methods that simulate quantization at train-time, so the model learns "robustness" to quantization. A core challenge to training-time quantization is that quantization functions are non-differentiable. The functions necessary to approximate gradients for these non-differentiable operations, such as the straight through estimator [5], inherently produce biased gradients during backpropagation. Quant-Noise [3] addresses this limitation by quantizing only a random subset of the total network during each training step, allowing some unbiased gradients to flow while still teaching the model to be resilient to the distortion. Note that Quant-Noise selects this subset using a Bernoulli trial for each parameter, where the probability input (labelled the *noise rate*) is constant across all examples and epochs.

We suggest that using a varying noise rate can lead to further robustness to quantization. We term this "second-order noise," since we are adding noise to the noise rate itself.

We propose several novel formulations where the quantization noise rate changes throughout training. We evaluate these methods on three tasks (RTE, MRPC, and HarperValleyBank), with three quantization schemes apiece (`int8`, `int4`, and `int1`). We use three random seeds for most of our studies; counting each (task, method, quantization scheme, seed) tuple as a single experiment, we perform a total of 129 experiments in this study.

Our contributions can be summarized under two categories:

Research

- *Second-order noise has the potential to outperform Quant-Noise.* We craft three candidate functions within the VQN family that demonstrate promise in improving upon Quant-Noise. These candidates are high-variance and do not seem to significantly outperform baselines. However, as part of an ablation study, we identify a method we call "random jitter" that surpasses Quant-Noise performance in two out of three quantization schemes for three tasks in the low data regime. We stress that further experiments are necessary to confirm that these results are not due to variance, and these findings can be used to inform and guide future investigations into improving quantization noise.

- *Quantization improves performance on a small speech dataset.* For a small speech dataset, HarperValleyBank, `int4` and `int8` quantization (using either post-training quantization or training-time quantization noise) outperforms the uncompressed baseline. Based on follow-up analyses on our speech dataset and prior work, we suspect that quantization has a regularizing effect for this dataset.

Engineering

- *New Quant-Noise-enabled modules*. We enabled Quant-Noise for layers in RoBERTa and CTC architectures; this involved custom adaptations of stacked LSTMs and objective functions.

- *New quantization schemes*. We added support for `int4` and `int1` (previous lowest was `int8`) quantization in FairSeq.

- *VQN*. We added functionality to vary the quantization noise rate per example as a function of previous loss, converged uncompressed loss, or schedule; these involved modifications to the base modules and the trainer that coordinates them.

2 Related Work

Model compression methods reduce the memory requirements for neural networks. Pruning and knowledge distillation are forms of compression that reduce the number of network weights. Pruning is a method that removes weights based on their network-level importance. For instance, magnitude pruning introduced in [6] removes weights with low magnitude. Knowledge distillation, as first named in [7], begins with a large pre-trained teacher model, and trains a compact student model by exposing it to raw predictions generated by the teacher model on an unlabeled dataset.

Quantization, by contrast, minimizes the number of bits needed to represent each weight. Standard neural networks use 32-bit floating point precision; scalar quantization replaces these weights with an N bit representation. Applying quantization methods like `int1`, `int4`, or `int8` as a post-training step causes errors to accumulate at each layer in the model, leading to significantly worse performance. Quantization Aware Training (QAT) is a method proposed in [8] that applies simulated quantization *during training*, so the network learns robustness to the quantization transformations. However, quantization operators tend to be stepwise functions and thus have null gradient; therefore, during back-propagation, gradients are approximated by a straight through estimator (STE) [9]. The STE is a simple idea: treat the stepwise function as if it were the identity function [9]. As the original authors acknowledge, this estimator is "clearly...biased" [9]. In high compression regimes like `int4`, the error due to biased gradient flow from STE is severe.

Quant-Noise seeks to reduce this error [3]. Quant-Noise only applies simulated quantization to a random subset of weights during training at a "noise rate", allowing some unbiased gradient flow in backpropagation, improving upon the QAT method. Note that the noise rate is fixed across all examples and all epochs, and is tuned empirically by the authors in [3]. However, it is reasonable to think that a model might benefit from using different noise rates for different examples or different epochs. For instance, a model might benefit from using harder, or earlier, examples to learn the task's objective and using easier, or later, examples to learn robustness to quantization.

3 Approach

3.1 Quant-Noise and VQN

The noise rate p is the proportion of weights that undergo simulated quantization during training. Formally, we partition a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ in $r \times s$ blocks $\boldsymbol{b}_{ij}(k, e)$ at location (i, j) for a given training example k for a given epoch e:

$$\boldsymbol{W} = \begin{bmatrix} b_{11}(k,e) & \cdots & b_{1s}(k,e) \\ \vdots & \ddots & \vdots \\ b_{r1}(k,e) & \cdots & b_{rs}(k,e) \end{bmatrix}. \quad (1)$$

Let φ be a noise function which simulates quantization during training and let $p(k, e)$ be the noise rate for example k at epoch e. In the case of scalar quantization, each block is simply a single weight. *A1* in the Appendix has additional information on scalar quantization.

Then, for a forward pass, Quant-Noise applies the following function to each block $\boldsymbol{b}_{ij}(k, e)$ in the weight matrix:

$$\psi(\boldsymbol{b}_{ij}(k,e)) = \begin{cases} \varphi(\boldsymbol{b}_{ij}(k,e)) & \text{with prob. } p \\ \boldsymbol{b}_{ij}(k,e) & \text{otherwise.} \end{cases} \quad (2)$$

Notice that all examples, at every epoch, share the same p value. We hypothesize that the gradients induced by harder examples should propagate less-biased gradient flow. Thus, it would be beneficial to set a lower $p(k, e)$ for these examples so fewer computational nodes require bias-inducing STE. Conversely, easier examples should have a higher value of $p(k, e)$ to teach the network to be robust to quantization. In other words, our training procedure will leverage hard examples–in the active learning sense–to teach the network how best reach its objective and leverage easy examples to maintain performance–in the QAT sense–after compression.

We propose the following modification to Quant-Noise, which we call VQN:

$$\psi(\boldsymbol{b}_{ij}(k,e)) = \begin{cases} \varphi(\boldsymbol{b}_{ij}(k,e)) & \text{with prob. } \pi(k,e) \\ \boldsymbol{b}_{ij}(k,e) & \text{otherwise} \end{cases} \quad (3)$$

where $\pi(k, e)$ is some function that takes in example k and an epoch e. For Quant-Noise baselines, we set $p(k, e)$ to be a constant value, which we call $\hat{p}$. For our adaptive methods, we experiment with three variants of $\pi(k, e)$.

Variant 1. $\pi_{\text{hard1}}(k, e) = \hat{p} - \lambda h(k, e)$, where $h(k, e)$ quantifies the *hardness* of the example for the quantized model and λ is a positive scaling factor. In these experiments, we use loss $\mathcal{L}_{k,e-1}$ on example k and previous epoch $e - 1$ as a proxy for $h(k, e)$. The loss is min-max scaled to the range $[-1, 1]$ using loss for all examples at epoch $e - 1$. We add $\hat{p}$ so the distribution of $\pi(k, e)$ is centered around the noise rate we use for Quant-Noise experiments.

Variant 2. $\pi_{\text{hard2}}(k, e) = \hat{p} - \gamma g(k)$, where $g(k)$ quantifies the hardness of the example for the *unquantized* model and γ is a positive scaling factor. This variant is analogous to the previous; however, we measure hardness using the unquantized model. That is, the hardness proxy $g(k)$ is measured by the loss of the uncompressed model $\tilde{\mathcal{L}}_k$ which is only a function of example k and thus constant across all epochs. We normalize $g(k)$ using the loss for all examples of the unquantized model at convergence.

Variant 3. $\pi_{\text{sched}}(k, e) = \hat{p} - \alpha z(e)$, where $z(e)$ quantifies the current position in training and α is a positive scaling factor. This approach is based on scheduled learning rates where the optimal learning rate early on in training is distinct from the optimal learning rate in later stages [10]. We hypothesize that quantization noise rate benefits from a similar schedule. Low values of p in the beginning let the network learn the task. Then, when the model has had the chance to reasonably fit to the training data, it can begin developing resilience to quantization. In these experiments, we choose $z(e)$ to be a linear function such that π_{sched} takes on a mean value of $\hat{p}$ over all epochs. More precisely, we set $z(e) = 1 - \frac{2e}{m}$, where m is set number of epochs the model will train for.

Finally, in what was originally an ablation study, we propose "random jitter," which adds a random amount of noise to the noise rate, drawing from a uniform distribution $\pi \sim \mathcal{U}(\hat{p} - \beta, \hat{p} + \beta)$, where β is a tunable hyperparameter.

3.2 Model Architectures

For our NLP experiments, we fine-tune a RoBERTa base (available at [11]) with an added classification head. For each of RTE and MRPC, we establish baselines with uncompressed models, post-training quantization, and Quant-Noise. Each baseline is measured using `int8`, `int4`, and `int1` schemes.

For our SLP experiments, we train (from scratch) an RNN-based network tasked on a multi-task objective. We adopt the training framework proposed in [12] and open-sourced at [13] that jointly optimizes CTC loss on the task of Automatic Speech Recognition (ASR) and Cross-Entropy (CE) loss on the auxiliary task of intent classification. Henceforth, we refer to this architecture as the CTC model. Note that our reported downstream performance is measured with respect to the task of intent classification since its validation curve converged 3x faster than the curve for ASR. The CTC model is composed of a recurrent layer, a stacked LSTM with $depth = 2$, and a softmax layer. Unlike many of the layers required for RoBERTa, LSTMs are not supported off-the-shelf by Quant-Noise. In addition, PyTorch also does not release its LSTM implementation since much of it is in C for optimization purposes. Enabling quantization noise with LSTMs for our experiments required adapting and integrating a native PyTorch LSTM frame from the GitHub repository located at [14].

4 Experiments

4.1 Data

For NLP branch of experiments, we focus on binary classification problems of (1) detecting textual entailment and (2) determining semantic equivalence. The former involves determining whether a given text fragment is entailed by another text fragment [15]; we use the RTE dataset (2.5k fragment pairs). The latter involves determining whether a pair of text fragments have semantic equivalence; we use the MRPC dataset (5.8k sentence pairs [16]). Both datasets are drawn from the GLUE benchmark [17]. For our SLP branch of experiments, we focus on the task of intent classification. Our chosen dataset, HarperValleyBank (1.4k customer-agent conversations) [12], frames this as an 8-class (e.g. *order checks*, *transfer money*) classification problem. We take as input Mel-frequency cepstral coefficients (MFCCs), which are encoded representations of raw audio signal. MFCC feature generation is the standard process of encoding audio in speech.

4.2 Evaluation method

We consider both the task-specific evaluation metrics (accuracy for all three tasks considered) and the compression ratio (*original_model_size / quantized_model_size*).

4.3 Experimental details

NLP We slightly modify the hyperparameters suggested in [18] under RoBERTa fine-tuning. First, we remove settings related to fp16. Second, due to memory constraints, we were only able to train with a batch size of 4 instead of 16. We set our parameter update frequency to 4 instead of the default 1 so that gradient updates would still occur at an effective batch size of $4 * 4 = 16$. All of our experiments use $\hat{p} = 0.5$, a reasonable default choice, as indicated by [18] and [3]. We also set $\lambda = 0.125$ for π_{hard1}, $\gamma = 0.125$ for π_{hard2}, $\alpha = 0.4$ for π_{sched}, and $\beta = 0.125$. We train for 10 epochs. These experiments were carried out on 6 Tesla K80s through the Azure Cloud Computing Services, totaling approximately 500 GPU hours.

SLP We largely maintain the same hyperparameter settings as HarperValleyBank's baselines (found at [13]). Data augmentation is also performed (reasoning explained in *Results*) with time and frequency masks applied with a probability of 0.5 for each training example at every epoch. See Appendix Figure 1 for an example. Note that we used SpecAugment to apply these augmentations [19]. Another deviation from the original model configuration is that we train for 40 epochs instead of 200. As with our NLP experiments, we set $\hat{p} = 0.5$ when evaluating Quant-Noise. These experiments were carried out on 6 Tesla K80s through the Azure Cloud Computing Services, totaling approximately 100 GPU hours.

4.4 Results

First, a general note on figures: dashed lines indicate performance of the uncompressed model; error bars represent standard deviations based on the three seeds for each trial.

4.5 NLP Results

We display our results in Appendix 1.2. We report average performance across three training runs, each with a different seed, with the exception of the uncompressed model and post-training quantization, for which we only train one model.

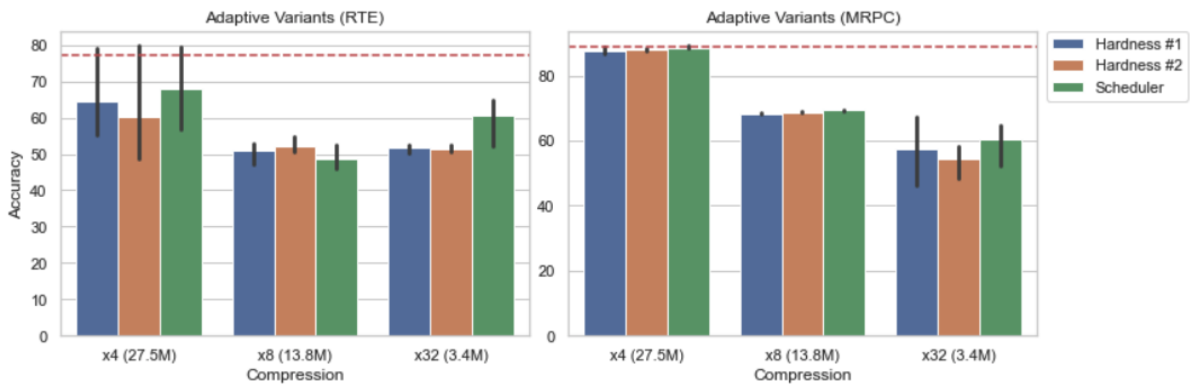

Figure 1: We compare performance of our three adaptive variants. The scheduler variant appears to perform best.

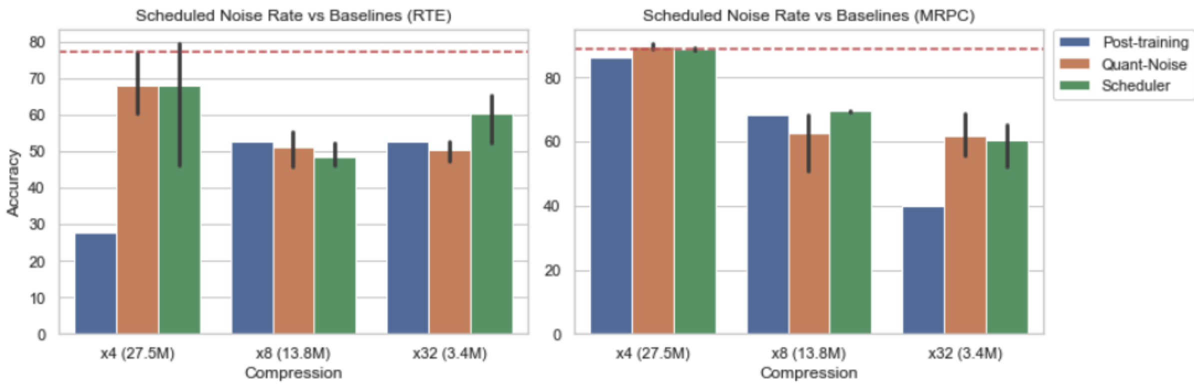

Figure 2: We compare the performance of our scheduler against Quant-Noise and the post-training quantization baseline. The scheduler variant is roughly even with Quant-Noise and outperforms post-training quantization.

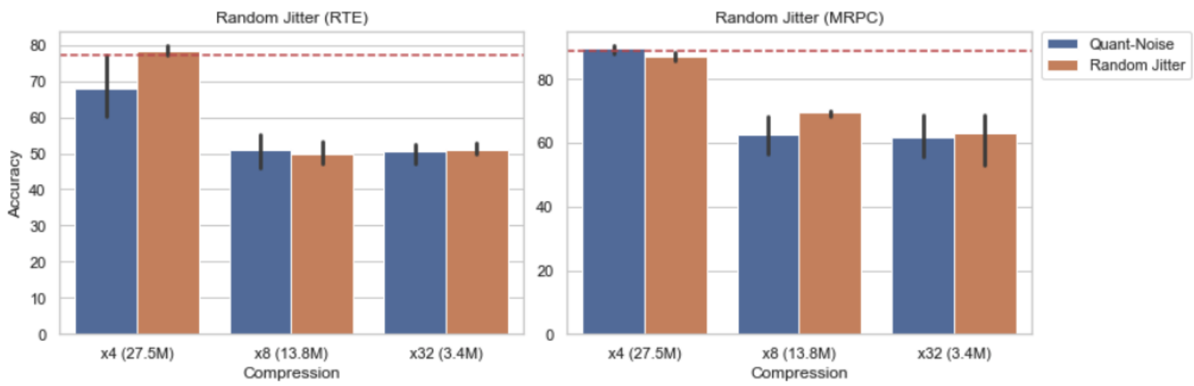

Figure 3: We compare the performance of "random jitter," our best method from the VQN family, against Quant-Noise. Random jitter outperforms Quant-Noise.

As corroborated by Figure 1, we find that of the 3 different adaptive variants, the scheduled adaptive Quant-Noise variant outperforms the other adaptive variants for 2 of the 3 quantization schemes on RTE. Namely, the schedule adaptive variant attains an accuracy of 67.99, 48.49, and 60.46 for `int8`, `int4`, and `int1`, respectively. Likewise, on MRPC, the adaptive scheduler variant consistently outperforms other adaptive variants for all quantization schemes.

Furthermore, as shown in Figure 2, we compare the adaptive scheduler variant to our baselines of Quant-Noise and post-training quantization. Notably, the adaptive scheduler variant outperforms most post-training quantization and some Quant-Noise baselines, attaining a higher average accuracy than Quant-Noise on 3 of the 6 datapoints for RTE and MRPC.

On RTE, we attain an average accuracy of 67.99 for `int8` with maximum accuracy of 79.42 on seed 1, which outperforms both baselines. Similarly, on `int1`, we obtain an average accuracy of 60.46, with a maximum accuracy of 65.2 on seed 3. On the other hand, we slightly underperform for `int4` with an average accuracy of 48.49, below both the post-training quantization and Quant-Noise baselines with 52.71 and 51.14, respectively. On MRPC, the adaptive scheduler variant outperforms the baselines with `int4` quantization, attaining an average accuracy of 69.36.

Most surprisingly, we found that fine-tuning RoBERTa base using Quant-Noise + random jitter performs better than expected; this outperforms the Quant-Noise baseline on 4 of 6 (task, quantization scheme) pairs, as shown in Figure 3. In particular, as illustrated in Table 1, random jitter on RTE substantially outperforms Quant-Noise for RTE on `int8` and MRPC on `int4` (78.46% vs. 67.87%; 69.53% vs. 62.50%, respectively).

The introduction of second-order noise with random jitter leads to improvements upon Quant-Noise for `int1`, `int4`, and `int8` quantization schemes. By jittering the noise rate, all examples have the same expected noise rate, but the amount of noise varies randomly throughout training. We have observed that under these conditions, randomly jittering the noise rate during training appears to slightly outperform Quant-Noise. While we observe positive results under these conditions, we note that these experiments were run on relatively small datasets, and we had no compelling reason *a priori* to believe that random jitter should improve performance. Thus, before making general claims about this method, it is important for future work to extend our work to larger datasets.

4.6 SLP Results

Our group's custom implementation of the CTC model with quantization noise attains an accuracy score of 39.6% (see Appendix Figure 2), which is within reasonable range of the baseline reported in the original HVB study [12]. The subsequent baselines we establish for *Post-training Quantization* and *Quant-Noise* have unexpected results. One would expect the former to lead to heavy regressions in performance while the latter is shielded from much of it. The Appendix displays a contradictory trend: the two methods perform equally as well for all compression levels and, even more remarkably, improves on uncompressed accuracy with 4x and 8x scalar quantization. This is a surprising result and indicates that quantization can actually have a helpful effect on performance for this dataset; we explore this more in *SLP Analysis*.

Figure 4 illustrates the results of a follow-up experiment with data augmentation added. Stronger regularization removed the unexpected positive correlation between accuracy and compression ratios. `int8` quantization (4x compression) produced a slight drop in accuracy, similar to what is observed in [3]. `int4` quantization (8x compression) continues to perform just as well as the `int8` model. The lack of a performance gap between post-training quantization and VQN is still evident- even with `int4` quantization where [3] observed significant gains with using quantization noise.

We again observe that adding second-order noise in the form of random jitter improves upon the Quant-Noise baseline. It does so for both `int8` ($\times$4) and `int1` ($\times$32) quantization, as can be seen in Table 1.

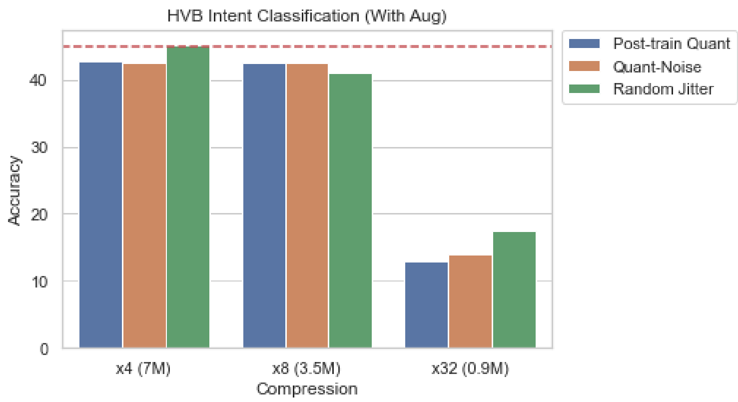

Figure 4: Repeat *Post-training Quantization* vs *Quant-Noise* experiment with time and frequency mask augmentations.

5 Analysis

5.1 NLP Analysis

One of the underlying assumptions for our adaptive variants was that unbiased gradient flow on an example would help the model better learn that example. Is this borne out by our results? In this section, we conduct qualitative analyses to answer this question.

In the Appendix, we display a scatterplot with the loss of the uncompressed examples on RTE on

Table 1: Quant-Noise versus Random Jitter with `int1`, `int4`, and `int8` quantization schemes.

Quantization	**Scheme/compression rate on RTE**			**Scheme/compression rate on MRPC**			**Scheme/compression rate on HVB**		
	int8/× 4	int4/× 8	int1/× 32	int8/× 4	int4/× 8	int1/× 32	int8/× 4	int4/× 8	int1/× 32
Random jitter	**78.46**	49.82	**51.02**	86.93	**69.53**	**62.83**	**45.1**	41.1	**17.4**
Quant-Noise	67.87	**51.14**	50.54	**89.46**	62.50	61.85	42.5	**42.6**	14.1

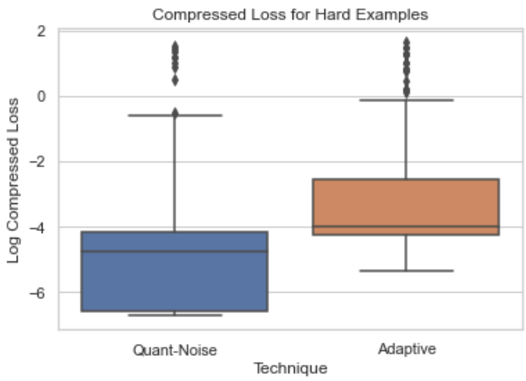

Figure 5: We compare loss on the hardest training examples in RTE (90th percentile of loss of the uncompressed model) between a Quant-Noise model and an adaptive model (variant 2).

the x-axis, and the loss of the adaptive (variant 2) model and the Quant-Noise model on the y-axis. Both y-axis models use `int8` quantization, and we plot losses on training examples. Our scatterplot indicates the opposite of what we expected; loss does not appear lower for the "harder" (higher uncompressed loss) examples on our adaptive variant 2 model as compared to Quant-Noise.

We also create a boxplot comparing the loss of the Quant-Noise model against the loss of the adaptive (variant 2) model on the "hardest" (highest uncompressed loss) training examples. Our scatterplot indicates the opposite of what we expected; loss is higher for the hardest examples on our adaptive variant 2 model as compared to Quant-Noise.

5.2 SLP Analysis

A significant portion of the speech evaluations are aimed at testing our group's custom implementation of the CTC model with quantization noise. Results displayed in Appendix under-perform relative to the scores reported in the original HVB study (47.8% accuracy) [12]. Meeting with one of its authors confirmed that this was expected since there were additional steps not included in the released code. Incorporating data augmentation as in Figure 4 closes this gap by attaining 45% accuracy.

We then move to understand the impact of established quantization methods on model behavior for our problem setting since Quant-Noise is untested on both speech data and recurrent models. First, we explain the counter-intuitive positive correlation of compression level and accuracy in our baseline. Diagnostics on the train and validation loss curves reveal that models at all compression levels are overfitting to the small dataset. This points an explanation based on quantization's regularizing effect: perhaps gains with higher compression ratios was due to lowering variance? Replicating the baselines with stronger regularization in the form of data augmentation confirmed this hypothesis by removing the positive correlation. Notable approaches, such as one done by Wu and Flierl [20] and another by Hirose et al. [21], explicitly use quantization as regularization mechanisms. Our results in this study indicate that such methods are likely to help CTC-based models generalize as well.

Attaining explainable results after data augmentation allowed us to reliably test *random jitter*, the best method second-order noise method that, surprisingly, came out of the ablations from the NLP trials. *Random jitter* outperforming *Quant-Noise* on two compression schemes provides additional evidence suggesting the former may produce gains. This further motivates future investigations into *random jitter*, particularly since speech is a different domain that operates on waveforms that are quite different from text-based sequences. In addition, HarperValleyBank is the smallest dataset we test, continuing support for *random jitter* across dataset sizes as well.

Likely reasons for the small performance gap between *Post-training Quantization* and the baselines are that (1) intent classification is straight-forward enough of a task such that even aggressively quantized networks can effectively learn it and (2) quantization noise is not much more effective than post-training quantization in low-resource regimes. The first reason is tested by mirroring the experiment from Figure 4 and evaluating ASR word error rate instead of intent classification. All performance trends persisted, thus disproving the first explanation. The second, on the other hand, is supported by the fact that quantization noise brings about significant gains with experiments involving Wikitext-103 (100 million tokens), ImageNet (1.2 million images), and RTE/MRPC (small datasets, but used with pretraining) while it does not on the smaller HarperValleyBank dataset. Insights from our NLP runs also supports this hypothesis: Quant-Noise is a high-variance method. One of the well-studied properties of high-variance methods is that they require more data to outperform less expressive alternatives [22]. Though this is a likely explanation with a theoretical base, future work can provide empirical proof by using our framework to evaluate ASR with a larger dataset such as LibriSpeech [23].

6 Conclusion

In this study, we explored methods for varying the quantization noise rate during training, and observe that, surprisingly, adding random jitter to Quant-Noise seems to benefit performance. Our work has limitations: first, for our NLP tasks, we perform both model selection (i.e., selecting the best model checkpoint from training) and evaluation on the development set. This is standard practice for GLUE tasks, but if we had more time, it would be better to evaluate our best models on the GLUE test set. Similarly, for HarperValleyBank, we perform both model selection and evaluation on the development set; again, however, this is consistent with what is done in the original paper [12].

In the future, we will explore whether our surprising random jitter results generalize to other, larger datasets. Also, we will further investigate why our adaptive method led to increased loss on harder examples as compared to Quant-Noise. If we can find a reliable method for decreasing loss on harder examples, this may be a promising avenue for further performance gains in quantization. Additionally, we hope to measure speedups in inference time rather than just compression ratios. Finally, rather than hand-crafting functions for the noise rate, we could search over a large space of functions or learn the noise rate as a parameterized policy network with model-free RL methods like deep Q-learning or policy gradient optimization. However, the relative merits of model speedups at inference time obtained by compression need to be considered within the context of model compression training time. For instance, neural methods like iterative Product Quantization (iPQ) or searching for a quantization noise rate using RL might incur larger environmental costs that outweigh the improved latency of the compressed model.

References

[1] T Brown and B et al. Mann. Language models are few-shot learners, 2020.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018.

[3] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression, 2020.

[4] R Ding and Z et al. Liu. Quantized deep neural networks for energy efficient hardware-based inference, 2018.

[5] P Yin and J et al. Lyu. Understanding straight-through estimator in training activation quantized neural nets, 2019.

[6] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[8] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[9] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[10] V Plagianakos and M Magoulas. Learning rate adaptation in stochastic gradient descent, 2016.

[11] Yinhan Liu and Myle Ott et al. Roberta: A robustly optimized bert pretraining approach, 2019.

[12] Mike Wu, Jonathan Nafziger, Anthony Scodary, and Andrew Maas. Harpervalleybank: A domain-specific spoken dialog corpus, 2020.

[13] Wu et al. Harpervalleybank github. *GitHub. Note: https://github.com/cricketclub/gridspace-stanford-harper-valley*, 2020.

[14] Daehwan Nam. Pytorch rnn util. *GitHub. Note: https://github.com/daehwannam/pytorch-rnn-util*, 2019.

[15] ACL Team. Recognizing textual entailment. "https://aclweb.org/aclwiki/Recognizing_Textual_Entailment", 2013. [Online; Accessed 26-Feb-2021].

[16] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[17] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[18] Fan et al. Fairseq github. *GitHub. Note: https://github.com/pytorch/fairseq*, 2020.

[19] Zach Caceres. Specaugment with pytorch. *GitHub. Note: https://github.com/zcaceres/spec*$_augment$, 2019.

[20] Zach Caceres. Quantization-based regularization for autoencoders. 2019.

[21] K Hirose and K et al. Ando. Quantization error-based regularization in neural networks. 2017.

[22] Jason Brownlee. How to reduce variance in a final machine learning model. 2018.

[23] Panayotov et al. Librispeech: an asr corpus based on public domain audio books. 2015.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

A Appendix

A.1 Scalar Quantization

Scalar quantization uses lower-bit representations of floating point weights. According to [3], for fixed-point `intn` quantization, we quantize weights by applying the following element-wise transform:

$$\boldsymbol{Q} = (\text{round}(\boldsymbol{W}/s + z) - z) \times s \tag{4}$$

where the *scale* is $s = \frac{\max \boldsymbol{W} - \min \boldsymbol{W}}{2^n - 1}$ and the *bias* is $z = \text{round}(\min \boldsymbol{W}/s)$. Note, the activations are also quantized at inference time. Following this compression scheme, the corresponding compression ratio is $32/n$.

A.2 Implementation

Apache License, Version 2.0

We built off of the original codebase for Quant-Noise, available at [18]. FairSeq uses the MIT license. We also rely on PyTorch more broadly [24]. For our HarperValleyBank dataset, which uses Creative Commons Public Licenses. Each of the variants for $\pi(k, e)$ were custom implementations. This involved functionality to vary the quantization noise rate per example, per epoch, and per schedule using modifications to PyTorch modules, the trainer that coordinates them, and other functions called by the trainer. Two of our compression regimes (`int1` and `int4`) also involved additions to FAIR's quantization operators and corresponding low-level PyTorch code. A pull request (found at https://github.com/pytorch/fairseq/pull/3370) has been opened to merge a small subset of our additions to FairSeq [18].

A.3 ACL Ethical Considerations

For papers presenting new datasets AND papers presenting experiments on existing datasets:

1. Does the paper describe the characteristics of the dataset in enough detail for a reader to understand which speaker populations the technology could be expected to work for? **Yes.**

2. Do the claims in the paper match the experimental results, in terms of how far the results can be expected to generalize? **Yes.**

3. Does the paper describe the steps taken to evaluate the quality of the dataset? **Yes.**

Table 2: Fine-tuning RoBERTa on RTE and MRPC using Quant-Noise versus Random Jitter with `int1`, `int4`, and `int8` quantization schemes.

Quantization	**Scheme/compression rate on RTE**			**Scheme/compression rate on MRPC**		
	`int8`/× 4	`int4`/× 8	`int1`/× 32	`int8`/× 4	`int4`/× 8	`int1`/× 32
Uncompressed	77.26	77.26	77.26	88.73	88.73	88.73
Post-training Quantization	27.80	**52.71**	52.71	86.03	68.14	39.71
Quant-Noise	67.87	51.14	50.54	**89.46**	62.50	61.85
Adaptive noise variant 1	60.40	52.22	51.26	87.99	68.79	54.33
Adaptive noise variant 2	60.40	52.22	51.26	87.99	68.79	54.33
Adaptive noise variant 3	67.99	48.49	**60.46**	88.65	69.36	60.46
Random jitter	**78.46**	49.82	51.02	86.93	**69.53**	**62.83**

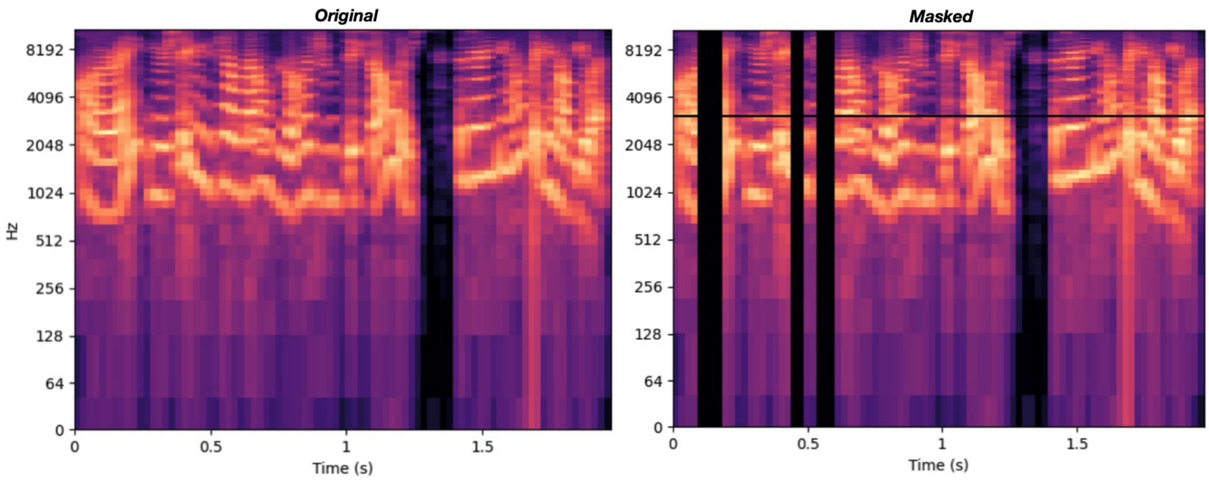

Figure 6: A sample spectrogram with time and frequency masks applied.

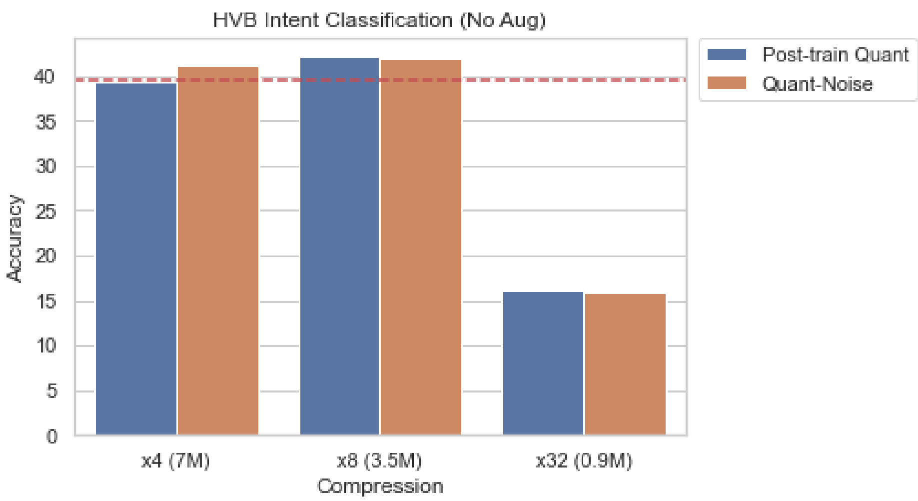

Figure 7: Establish baselines with *Post-training Quantization* to *Quant-Noise* at four levels of compression.

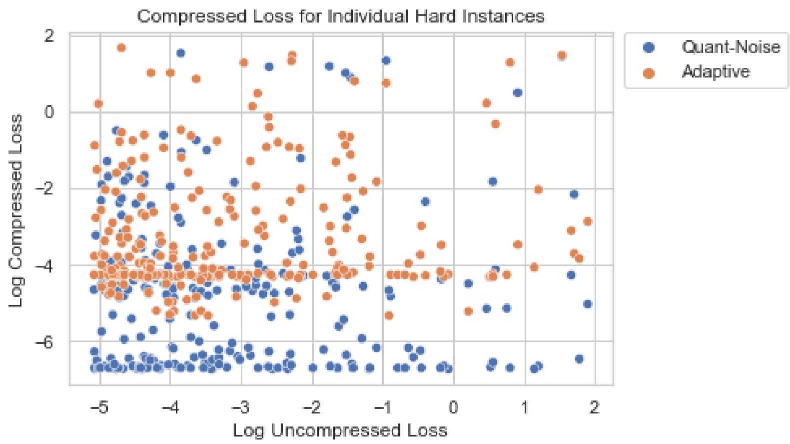

Figure 8: We plot loss on training examples for the uncompressed model against loss on these same examples for the Quant-Noise model and adaptive model (variant 2).