FULLY QUANTIZED TRANSFORMER FOR IMPROVED TRANSLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

State-of-the-art neural machine translation methods employ massive amounts of parameters. Drastically reducing computational costs of such methods without affecting performance has been up to this point unsuccessful. To the best of our knowledge, we are the first to propose a quantization strategy inclusive of all components of the Transformer (Vaswani et al., 2017). We are also the first to show that it is possible to avoid any loss in translation quality with a fully quantized network. Indeed, our 8-bit models consistently score equal or higher BLEU than the full-precision variant on multiple translation datasets. Comparing ourselves to all previously proposed methods, we achieve state-of-the-art quantization results.

1 INTRODUCTION

The idea of using neural networks for machine translation was proposed only recently (Kalchbrenner & Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). Nonetheless, the approach has reached impressive levels of translation. (Ahmed et al., 2017; Ott et al., 2018; Edunov et al., 2018). A key element of this success was to allow the decoder to attend to all hidden states of the encoder (Bahdanau et al., 2014). A few variations to this additive attention mechanism were proposed, such as multiplicative attention and self-attention (Luong et al., 2015; Cheng et al., 2016; Lin et al., 2017). The latter formed the basis of the Transformer network (Vaswani et al., 2017), which achieved state-of-the-art machine translation. Inspiring a new wave of work, numerous natural language processing tasks reached new heights (Devlin et al., 2018; Liu et al., 2019). Unfortunately, these models make use of an enormous amount of parameters. Inference on resource-limited hardware such as edge-devices is thus impractical.

A solution to reduce the computational burden of these neural networks is to lower numerical precision. Consequently, numerical values can be represented using fewer bits (Tang & Kwan, 1993; Marchesi et al., 1993). This method called quantization has the advantage of providing good compression rates with minimal loss in accuracy. It is also conveniently supported by most hardware. Properly quantizing the Transformer would allow computational speed gains at inference, as well as deployment on more constrained devices.

In this work, we propose a custom quantization strategy of the entire Transformer architecture, where quantization is applied during the training process. Our method is easy to implement and results are consistent with the full-precision Transformer. We test our approach on multiple translation tasks such as WMT14 EN-FR and WMT14 EN-DE and obtain state-of-the-art quantization results. On most tasks, our quantized models score equal or higher BLEU compared to full-precision. We are, to the best of our knowledge, the first to fully quantize the Transformer architecture without impairing translation quality.

2 BACKGROUND

In this section, we review a broad spectrum of quantization and pruning methods for neural network compression.

2.1 QUANTIZATION

Over the years, a large range of methods have been proposed to quantize neural networks. These include, among many others, binary (Courbariaux et al., 2016), ternary (Lin et al., 2015; Li et al., 2016), uniform affine (Jacob et al., 2017) and learned (Zhang et al., 2018) quantization.

Quantization has been applied to RNNs (Jordan, 1990), LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014). Ott et al. (2016) propose an exponential quantization method for RNN weights. They compare their method with binary and ternary quantization on language modeling and speech recognition. They reach the conclusion that binary weights are ineffective for RNNs, while ternary and exponential quantization perform well. Hubara et al. (2016) quantize weights and activations of RNNs and LSTMs to 2, 4 and 6-bit. He et al. (2016) propose modifications to the gates and interlinks of quantized LSTM and GRU cells, as well as a balanced quantization method for weights. Wu et al. (2016) successfully quantize the stacked LSTM neural machine translation model to 8-bit without any loss in translation quality. Wang et al. (2018) propose to use different quantization methods for different RNN components.

With regards to CNNs (LeCun et al., 1989), various works have explored quantizing the architecture. Gong et al. (2014) compare matrix factorization, binarization, k-means clustering, product quantization and residual quantization on CNNs. Wu et al. (2015) apply quantization to both kernels and fully connected layers of convolutional neural networks. Rastegari et al. (2016) evaluate the use of binary weights on AlexNet (Krizhevsky et al., 2012). Zhou et al. (2016) use low bitwidth weights, activations and gradients on CNNs.

Quantization has also been jointly used with other compression methods. Han et al. (2015) combine pruning, quantization, weight sharing and Huffman coding, while Polino et al. (2018) use quantization with knowledge distillation (Hinton et al., 2015) for higher compression rates.

2.2 PRUNING

LeCun et al. (1990) were the first to propose a Hessian based method to prune neural net weights, with Hassibi et al. (1994) later improving the method. More recently, See et al. (2016) show that pruning a fully trained model and then retraining it can increase performance over the original non-pruned model. Gradually pruning in tandem with training has also been shown to increase performance (Zhu & Gupta, 2017). Liu et al. (2017) prune nodes instead of weights by applying a penalty in the loss on the γ parameters of batch normalization layers. Narang et al. (2017b) make better use of hardware by applying pruning and weight decay in blocks to minimize the number of loaded weight matrix chunks. Chen et al. (2018) combine quantization with block based low-rank matrix approximation of embeddings.

Pruning methods have also been adapted to specific architectures. Liu et al. (2015) propose an efficient sparse matrix multiplication algorithm for CNNs. For RNNs, Narang et al. (2017a) show sparse pruning to work well on the architecture. In order to maintain dimension consistency, Wen et al. (2017) propose to prune all basic LSTM structures concurrently. Park et al. (2018) introduce simple recurrent units (SRUs) for easy pruning of RNNs.

3 QUANTIZATION STRATEGY

3.1 QUANTIZATION METHOD

Our quantization methodology was chosen to be uniform, meaning that the step size between two quantized values is constant. This choice, which is an additional constraint, was made for practical reasons. It indeed simplifies all computations required during inference, enabling the exploitation of hardware resources more efficiently. If the performance with uniform quantization is already on par with full-precision, then more weighty methods are unnecessary. The uniform quantization scheme employed is further described by Jacob et al. (2017).

Given an element x of a tensor $\mathbf{X}$, we apply the quantization function $\mathcal{Q}$:

$$\mathcal{Q}(x) = \frac{x - x_{min}}{s} \tag{1}$$

$$s = \frac{x_{max} - x_{min}}{2^k - 1} \tag{2}$$

where x_{min} and x_{max} defines the endpoints of the quantization interval. When quantization is applied to weights, these values are respectively $\min(\mathbf{X})$ and $\max(\mathbf{X})$. However, when quantization is applied to activations, those values are running estimates. The latter are computed during training, where for every forward pass, the x_{min} and x_{max} variables are updated via an exponential moving average with a momentum of 0.9. The value k is the bit precision. For example, in the context of 8-bit quantization, $k = 8$.

At training time, we simulate quantization by first quantizing and then rescaling to the original domain:

$$\left\lfloor \frac{\mathrm{clamp}(x; x_{min}, x_{max}) - x_{min}}{s} \right\rceil * s + x_{min} \tag{3}$$

where the clamp function associates all the values outside the $[x_{min}, x_{max}]$ range to the closest endpoint and $\lfloor\cdot\rceil$ represents rounding to the nearest integer. During backpropagation, we use the straight-through estimator (Hinton, 2012) and set the gradients of clamped values to zero. The only exception is for the LayerNorm's denominator, where values can still be clamped, but gradients are never zeroed. Once training is finished, s and x_{min} are frozen along with the weights.

3.2 What to Quantize

We choose to quantize all operations which can provide a computational speed gain at inference. In this regard, we quantize all matrix multiplications, meaning that the inputs and weights of MatMuls will both be k-bit quantized. The other operations we quantize are divisions, but only if both the numerator and denominator are second or higher rank tensors. For all other operations, such as sums, the computational cost added by the quantization operation outweighs the benefit of performing the operation with reduced precision. Hence, we do not quantize such operations.

More precisely, we quantize all weights of the Transformer, excluding biases. The latter are summed with the INT32 output of matrix multiplications and thus provide no additional computational efficiency from being quantized. Furthermore, the memory space of biases is insignificant in comparison to the weight matrices, representing less than 0.1% of total weights. For positional embeddings, memory gain is also minimal, but since these will be summed with the quantized input embeddings, we likewise quantize them. These embeddings also stay fixed, we can thus quantize them once prior to training. The γ weights of LayerNorms are also quantized. As for activations, we quantize the sum of the input embeddings with the positional encodings in both the encoder and decoder. In the Multi-Head Attention, we quantize the (Q, K, V) input, the softmax's numerator, the softmax's denominator, the softmax's output and the Scaled Dot-Product Attention's output. For the position-wise feed-forward networks, we quantize the output of the ReLUs and of the feed-forward networks themselves. Finally, for all LayerNorms, we quantize the numerator $x - \mu$, the denominator $\sqrt{\sigma^2 + \epsilon}$, their quotient and the output of the LayerNorm.

3.3 Bucketing

Instead of using a single set of (s, x_{min}) per quantized tensor, we can quantize subsets of the latter with each its own set of (s, x_{min}) (Alistarh et al., 2016). Even though this adds more scalars, the memory cost is insignificant overall. Furthermore, the added flexibility can greatly alleviate the precision loss resulting from trying to fit all values of a tensor into a single domain with lower numerical precision.

We use this bucketing method for all weight matrices, with the number of subset equal to the output dimension. For activations, we use bucketing when quantizing: the sum of input embeddings with the positional encoding, the Q, K, V inputs, the Scaled Dot-Product Attention's output, the feed-forward's output, the LayerNorm's numerator, the LayerNorm's quotient and the LayerNorm's output.

Table 1: Our quantization strategy achieves better BLEU scores than all other quantization methods for the Transformer on the WMT14 EN-DE, WMT14 EN-FR and WMT17 EN-DE test set.

Method	Fully Quantized	Size (Gb) [EN-DE, EN-FR]	Compr.	BLEU EN-DE (2014)	BLEU EN-FR	BLEU EN-DE (2017)
Vaswani et al. (2017)		[2.02, 1.94]	1x	27.3	38.1	-
Cheong & Daniel (2019)		0.69	2.92x	-	-	27.38
Bhandare et al. (2019)		$\geq$ 0.96	$\leq$ 2.1x	27.33	-	-
Fan (2019)		$\geq$ 0.51	$\leq$ 3.99x	26.94	-	-
Our method	✓	[0.51, 0.49]	3.95x	**27.60**	**39.91**	**27.60**

3.4 Dealing with Zeros

Unlike Jacob et al. (2017), we do not nudge the domain so that the zero value gets perfectly mapped. The only zero values which we have to deal with are the padding, the output of ReLU layers and dropouts. Since padding has no effect on the final output, we completely ignore these values when quantizing. For ReLUs, we fix the x_{min} estimate of those quantization layers to 0, which guarantees the perfect mapping of the value. Finally, quantization is applied before any dropout operation. Indeed, even though the zeros added to the output of the quantization layer might not be part of the domain, this only happens during training.

4 Related Work

Recently, simple quantization solutions have been applied to the Transformer. Cheong & Daniel (2019) apply k-means quantization and binarization with two centroids over the weights of the network. For both methods, a look up table associated with each quantized layer is used to map indices to their corresponding centroids. Similarly, Fan (2019) compares binary, 4 and 8-bit uniform quantization of the Transformer weights. A big disadvantage with quantizing only the weights of a network is that operations must still be performed in full-precision. Even though there is a reduced memory usage of parameters, these constantly have to be casted back to full-precision. Achieving quantization of both weights and activations is thus much more beneficial. The first attempt at doing so for the Transformer applied 8-bit quantization on weights and inputs of feed forward layers and binarizes the (Q, K) input of the Multi-Head Attention (Tierno, 2019). The scaling factor $\sqrt{d_k}$ is approximated by a constant which can be computed as a right bitshift. The method though results in huge drop in translation accuracy. Achieving better performance, Bhandare et al. (2019) quantize certain MatMul operations and use the KL divergence to estimate the most suited parameters for each quantization range. They restrain from quantizing all MatMuls, reporting that this resulted in poor accuracy.

All of these methods omit quantizing the whole Transformer architecture, resulting in suboptimal computational efficiency. Furthermore, these solutions all fail to avoid impairing translation quality. Our method achieves both.

5 Experiments

In this section, we present the results of our full quantization scheme on various tasks. We first compare our method on a machine translation setup. We then present the results of numerous ablation studies. We also compare the impact of delaying quantization on translation quality. Finally, we evaluate our method on two language model tasks.

5.1 Full Quantization

We apply our quantization strategy on both the base and big Transformer (Vaswani et al., 2017). The training setup of all presented models is the same as in the original paper, with the exception that the dropout ratio is set to 0.1 in all cases. We refer readers to the original paper for experimental details. Our models were first evaluated on the WMT 2014 / 2017 English-to-German and WMT 2014 English-to-French translation tasks. Section A contains results for additional languages. Reported

Table 2: Performance of our quantization method on the WMT14 EN-DE and WMT14 EN-FR test set for a fixed number of training steps.

Model	Method	Precision	EN-DE				EN-FR			
			PPL	BLEU	Size (Gb)	Compr.	PPL	BLEU	Size (Gb)	Compr.
Base	Baseline	32-bit	4.91	26.42	2.02	1x	3.23	38.36	1.94	1x
	Default Approach	8-bit	74.04	0.21	0.51	3.95x	*nan*	0	0.49	3.95x
	Post-Quantization	8-bit	4.97	26.44	0.51	3.95x	3.26	38.30	0.49	3.95x
	Our method	8-bit	4.67	**26.98**	0.51	3.95x	3.23	**38.55**	0.49	3.95x
	Post-Quantization	6-bit	6.00	24.84	0.38	5.25x	3.98	35.02	0.37	5.24x
	Our method	6-bit	5.09	**26.98**	0.38	5.25x	3.38	37.07	0.37	5.24x
	Our method	4-bit	11.96	18.32	0.26	7.81x	48.21	1.59	0.25	7.8x
Big	Baseline	32-bit	4.03	26.85	6.85	1x	2.72	40.17	6.69	1x
	Post-Quantization	8-bit	4.27	26.55	1.73	3.97x	2.78	39.78	1.68	3.97x
	Our method	8-bit	4.24	**27.95**	1.73	3.97x	2.80	**40.17**	1.68	3.97x
	Post-Quantization	6-bit	5.12	24.86	1.30	5.28x	3.08	37.92	1.27	5.28x
	Our method	6-bit	4.78	26.76	1.30	5.28x	2.87	39.59	1.27	5.28x
	Our method	4-bit	33.11	10.22	0.87	7.88x	42.42	2.81	0.85	7.88x

perplexity is per token and BLEU was measured with `multi-bleu.pl`[1] on the `newstest2014`[2] test set. We used beam search with a beam size of 4 and a length penalty of 0.6, as in (Vaswani et al., 2017). Another difference is that no checkpoint averaging was performed.

We compare our results with the original Transformer and other 8-bit quantization methods in Table 1. All models are base Transformers. Original uncompressed size is the same in all cases. Most work do not report their compressed model size. For those, we give lower bounds based on their reports. Our BLEU score was computed on the test set using the checkpoint with the highest validation accuracy of 2 million training steps. Validation was computed every training epoch. Our objective here was to train quantized models up to convergence. Very similar BLEU scores can be obtained with much fewer training (see below). As for other methods, Cheong & Daniel (2019) retrain for 10k steps a 200k steps pretrained Transformer. Fan (2019) also does the same but does not mention the number of retraining steps. Bhandare et al. (2019) and the original Transformer paper both do not mention the number of training steps. Out of all methods, we are the only one quantizing every component of the model (see section 4).

In Table 2, we show performance of our method on the WMT14 EN-DE and WMT14 EN-FR for a fixed amount of training steps. We compare our results with two full-precision Transformers: base and big variants. Training the quantized models was about twice as slow as training the baselines. We also compare with two other quantization approaches. The first one is the ”default” approach, which is to naively quantize every possible operation. The second approach applies our quantization strategy post-training (see section 5.3 for details). In all cases except for post-quantization, BLEU was computed on the test set using the checkpoint which scored the highest accuracy on the validation set. Towards the end of training, we ran one validation epoch for every 100 training steps. As for post-training quantization, the BLEU score was computed on the test set using the best scoring BLEU score on the validation set out of 20 trials. The latter varied by about 0.2 BLEU. For the big Transformer variants, best results were obtained when not bucketing the Scaled Dot-Product Attention’s output and the sum of the decoder’s input embeddings with the positional encoding. The reason for the default approach’s *nan* in the EN-FR task is because quantizing every operation causes numerical instability in the LayerNorm’s denominator, normally provided by the ϵ.

Generally, fully quantizing the Transformer seems to result in no loss in translation accuracy. We believe the reason for this might be the lower numerical precision acting as a regularization effect. Looking for such an effect in the training and validation curves, differences were too subtle for any conclusions to be made.

All models use full-precision biases, s and x_{min}. This amounts to 6.52 Mb in the base models and 13.04 Mb in the big models. Without bucketing, this would amount to 2.17 Mb and 4.33 Mb respectively. All in all, these represent less than 2% of the total size of our quantized models. We

[1]`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

[2]`https://www.statmt.org/wmt14/translation-task.html`

Table 3: Effect of quantizing single activations of the Transformer on the translation quality. Results are on the WMT14 EN-FR test set.

Module	Quantized Activation	No Bucketing PPL	No Bucketing BLEU	Bucketing PPL	Bucketing BLEU
Encoder	(Input Embedding + Positional Encoding)	3.20	38.61	3.20	39.08
Decoder	(Input Embedding + Positional Encoding)	3.20	39.35	3.20	39.36
Multi-Head Attention	Input (Q, K, V)	3.21	39.06	3.21	39.29
	LayerNorm Output	3.21	39.09	3.20	38.78
Scaled Dot-Product Attention	Softmax Numerator	3.20	39.32	3.21	39.01
	Softmax Denominator	3.21	39.35	3.21	39.11
	Softmax Output	3.22	39.41	3.22	38.87
	Output	3.21	38.73	3.21	39.02
Feed-forward	ReLU Output	3.21	39.43	3.22	38.93
	Feed-forward Output	3.54	38.03	3.20	39.27
	LayerNorm Output	3.21	38.67	3.21	39.04
LayerNorm	Numerator	3.53	37.75	3.21	38.86
	Denominator	1748	0	-	-
	Quotient	3.22	38.97	3.21	39.02

Table 4: Variations to our quantization scheme evaluated on the WMT14 EN-FR translation task.

Method	PPL	BLEU
No Bucketing	3.58	36.49
No Gradient Clipping	2549.30	0
No LayerNorm Denominator Quantization	3.20	37.97
8-bit Quantized Weights, Full-precision Activations	3.19	37.94

believe the small increase in model size is worth it in the case of bucketing. We show in section 5.2 that training without leads to poorer translation.

Although 6-bit quantization seems to perform well, the compression advantage over 8-bit is usually lost. Most hardware store INT6 using either 8 or 32 bits. Dedicated hardware is needed to get the full compression advantage. Unless 6-bit quantization results in better models, sticking to 8-bit seems like the best choice for most hardware.

5.2 Ablation Studies

To compare the effect of bucketing and better understand which operation is more sensitive to quantization, we evaluate the effect of quantizing to 8-bit single operations of the Transformer, with and without bucketing. By single operation, we mean quantizing the operation of a module for all Transformer layers. Table 3 shows results on the WMT14 EN-FR translation task. BLEU was computed on the test set after 100k steps of training. The only operations underperforming our full-precision baseline of 38.36 BLEU are the LayerNorm's numerator when not bucketed and the denominator. The latter cannot be bucketed because all dimensions of the variance tensor vary per batch. Solely quantizing the LayerNorm's denominator with no bucketing works, but results are poor. To successfully quantize this element without causing performance loss, we suspect quantizing other elements in the network helps.

To further validate our quantization scheme, we evaluated four models trained with alterations to our design choices. Results are presented in Table 4). All models are 8-bit quantized base Transformers, trained in the same fashion as in section 5.1 on the WMT14 EN-FR task.

Table 5: Impact of delaying quantization on translation quality. Results are on the WMT14 EN-DE and WMT14 EN-FR test set.

Quantization Start	EN-DE		EN-FR	
(training step)	PPL	BLEU	PPL	BLEU
Base (no quantization)	4.91	26.42	3.23	38.36
100	4.67	**26.98**	3.23	38.55
10000	4.99	26.63	3.21	**38.62**
50000	4.98	26.84	3.21	38.50
80000	5.03	26.41	3.21	38.43
Post-Quantization	4.45	25.50	3.22	37.96

Table 6: Evaluation of our quantization method on the WikiText-2 and WikiText-103 language modeling tasks.

Precision	Size (Gb)	Compression	WikiText-2		WikiText-103	
			Loss	PPL	Loss	PPL
32-bit	243.04	1x	5.65	284.15	5.91	**369.20**
8-bit	61.22	3.97x	5.64	282.67	5.94	377.79
6-bit	46.04	5.28x	5.64	**281.48**	5.93	376.44
4-bit	30.86	7.87x	5.65	284.26	5.94	378.67

5.3 Delaying Quantization

Our method's goal is to increase computational efficiency when inferring with the Transformer. To this end, our quantization scheme only requires us to learn s and x_{min}. Although we do so with our quantization scheme throughout the whole training, this is not a necessity. Quantization could also be applied later on while training. Results for different starting points are compared in Table 5. The earliest we start quantizing is at 100 steps, since we need at least a few steps to assess the x_{min} and x_{max} running estimates. We consider this to be training with quantization "from scratch". Post-training quantization is also an option, where once the model is fully trained, we keep the weights fixed, but compute the s, x_{min} and x_{max} over a few hundred steps. All models were evaluated on the WMT14 EN-DE and WMT14 EN-FR translation tasks. BLEU was measured on the test set using the checkpoint which scored the highest accuracy on the validation set during training. Validation was computed every 100 training steps towards the end of training. From our observed results, quantizing the model early on seems preferable. This reinforces our belief that quantization helps training via regularization.

Learning quantization parameters adds a significant computational cost during training. A major advantage to delaying quantization is to perform more training steps in the same given amount of time. Therefore, when training time is a constraint, a possible strategy is to train a model without quantization, perform more training steps and finally post-quantize the model. Another advantage of post-quantization is that the method is quick to perform. This makes it easy to run many iterations and search for the best performing candidate.

5.4 Language Modeling

To evaluate if our quantization scheme generalizes well to other tasks, we evaluate it on a language modeling benchmark. As the setup, we use PyTorch's language modeling toy example[3] on the WikiText-2 and WikiText-103 corpus. The task consists of predicting the sequence $\{x_{t+1}, \cdots, x_{t+n+1}\}$ from the input sequence $\{x_t, \cdots, x_{t+n}\}$. We trained four Transformer models, one full precision and three with our quantization method. In each case, the model consists of two Transformer encoder layers, with the embedding and hidden size set to 200. Multi-Head Atten-

[3] https://github.com/pytorch/examples/tree/master/word_language_model

tion has two heads with keys and values of size 64. The final word projection layer's weights are shared with the embedding layer. Models were trained for 10 epochs with a batch size of 20 and sequence length of 35. Learning rate is set to 5, dropout to 0.2 and gradient clipping to 0.25. Loss is computed on every element of the output sequence. We refer readers to the PyTorch example (see 3) for any extra details. Results are presented in Table 6. Validation loss was computed every epoch to determine the best candidate. Loss and perplexity are computed on the test set and averaged over 10 trials for WikiText-2 and 3 trials for WikiText-3.

6 Conclusion

We proposed a quantization strategy for the Transformer, quantizing all operations which could provide a computational speed gain, for a fully quantized architecture. All of our design decisions were aimed at maximizing computational efficiency while making sure our method would be compatible with as many different types of hardware as possible.

With our method, we achieve higher BLEU scores than all other quantization methods for the Transformer on multiple translation tasks and avoid any loss in BLEU compared to full-precision. Specifically, out of 41 experiments, 8-bit quantization performed equal or better to full-precision in 36 cases.

We are very excited about the possibilities this work opens and plan on applying our method to other tasks. We also intend to extend our work to variations of the Transformer, as well as further exploring the compression of these networks.

References

Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted Transformer Network for Machine Translation. *arXiv e-prints*, art. arXiv:1711.02132, Nov 2017.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. *arXiv e-prints*, art. arXiv:1610.02132, Oct 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, art. arXiv:1409.0473, Sep 2014.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model. *arXiv e-prints*, art. arXiv:1906.00532, Jun 2019.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10988–10998. Curran Associates, Inc., 2018.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. *arXiv e-prints*, art. arXiv:1601.06733, Jan 2016.

Robin Cheong and Robel Daniel. transformers.zip: Compressing Transformers with Pruning and Quantization. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15763707.pdf`.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv e-prints*, art. arXiv:1406.1078, Jun 2014.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv e-prints*, art. arXiv:1602.02830, Feb 2016.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, art. arXiv:1810.04805, Oct 2018.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding Back-Translation at Scale. *arXiv e-prints*, art. arXiv:1808.09381, Aug 2018.

Chaofei Fan. Quantized Transformer. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15742249.pdf`.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv e-prints*, art. arXiv:1412.6115, Dec 2014.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, art. arXiv:1510.00149, Oct 2015.

Babak Hassibi, David G. Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. In J. D. Cowan, G. Tesauro, and J. Alspector (eds.), *Advances in Neural Information Processing Systems 6*, pp. 263–270. Morgan-Kaufmann, 1994.

Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou. Effective Quantization Methods for Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1611.10176, Nov 2016.

Geoffrey Hinton. Neural networks for machine learning, 2012. Coursera, video lectures.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv e-prints*, art. arXiv:1503.02531, Mar 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *arXiv e-prints*, art. arXiv:1609.07061, Sep 2016.

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. *arXiv e-prints*, art. arXiv:1712.05877, Dec 2017.

Michael I. Jordan. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pp. 112–127. IEEE Press, Piscataway, NJ, USA, 1990. ISBN 0-8186-2015-3.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary Weight Networks. *arXiv e-prints*, art. arXiv:1605.04711, May 2016.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural Networks with Few Multiplications. *arXiv e-prints*, art. arXiv:1510.03009, Oct 2015.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A Structured Self-attentive Sentence Embedding. *arXiv e-prints*, art. arXiv:1703.03130, Mar 2017.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-Task Deep Neural Networks for Natural Language Understanding. *arXiv e-prints*, art. arXiv:1901.11504, Jan 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. *arXiv e-prints*, art. arXiv:1708.06519, Aug 2017.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints*, art. arXiv:1508.04025, Aug 2015.

M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, 4(1):53–62, Jan 1993. ISSN 1045-9227. doi: 10.1109/72.182695.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring Sparsity in Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1704.05119, Apr 2017a.

Sharan Narang, Eric Undersander, and Gregory Diamos. Block-Sparse Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1711.02782, Nov 2017b.

Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. Recurrent Neural Networks With Limited Numerical Precision. *arXiv e-prints*, art. arXiv:1608.06902, Aug 2016.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling Neural Machine Translation. *arXiv e-prints*, art. arXiv:1806.00187, Jun 2018.

Jinhwan Park, Yoonho Boo, Iksoo Choi, Sungho Shin, and Wonyong Sung. Fully neural network based speech recognition on mobile and embedded devices. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10620–10630. Curran Associates, Inc., 2018.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv e-prints*, art. arXiv:1802.05668, Feb 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv e-prints*, art. arXiv:1603.05279, Mar 2016.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of Neural Machine Translation Models via Pruning. *arXiv e-prints*, art. arXiv:1606.09274, Jun 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc., 2014.

C. Z. Tang and H. K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727, Aug 1993. ISSN 1053-587X. doi: 10.1109/78.229903.

Andrew Tierno. Quantized Transformer. Technical report, Stanford University, Stanford, California, 2019. URL `https://web.stanford.edu/class/cs224n/reports/custom/15848474.pdf`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, art. arXiv:1706.03762, Jun 2017.

Peiqi Wang, Xinfeng Xie, Lei Deng, Guoqi Li, Dongsheng Wang, and Yuan Xie. Hitnet: Hybrid ternary recurrent neural network. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 604–614. Curran Associates, Inc., 2018.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning Intrinsic Sparse Structures within Long Short-Term Memory. *arXiv e-prints*, art. arXiv:1709.05027, Sep 2017.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. *arXiv e-prints*, art. arXiv:1512.06473, Dec 2015.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, art. arXiv:1609.08144, Sep 2016.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *arXiv e-prints*, art. arXiv:1807.10029, Jul 2018.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv e-prints*, art. arXiv:1606.06160, Jun 2016.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv e-prints*, art. arXiv:1710.01878, Oct 2017.

Table 7: Evaluation of our quantization method on the WMT14 EN-CS, WMT14 RU-EN and WMT14 ES-EN translation datasets.

Model	Method	Precision	EN-CS		RU-EN		ES-EN	
			PPL	BLEU	PPL	BLEU	PPL	BLEU
Base	Baseline	32-bit	6.90	22.71	3.56	32.62	5.59	**29.99**
	Our method	8-bit	6.81	**23.06**	3.53	**33.08**	5.60	29.88
Big	Baseline	32-bit	-	-	-	-	5.32	30.06
	Our method	8-bit	-	-	-	-	5.35	**30.15**

A Additional Datasets

We evaluated our quantization method on additional translation datasets (see Table 7). All models are trained following the same setup as in section 5.1, except the big model was only trained for one epoch. Vocabulary size is set to 30k for all models. Since there is no test set for WMT14 ES-EN, we used the validation set as a test set and omitted computing any validation epochs during training.

B Pruning Useless Nodes

We propose an additional compression method for the Transformer, which is independent of our quantization method. Both though can be used conjointly to further compress the Transformer.

Once the model is fully trained and quantized, we can further compress it by removing useless nodes. By useless, we mean nodes which do not cause any loss in translation quality when removed. We choose to prune nodes instead of independently pruning weights, to avoid the need of any special hardware or software, which is usually the case when trying to leverage sparse weight matrices obtained by the latter method. Pruning nodes results in concretely shrunken models. When getting rid of a node, we remove its corresponding set of weights from the layer outputting it and the following layer receiving the node as input.

The only nodes of the Transformer which can be removed without causing alterations to other components of the network are the nodes in between the two layers of each feed-forward network. Fortunately, these consist of a substantial portion of the model's weights. In the case of the base Transformer, for a respective vocabulary of size 37000 and 32000, 39.96% and 41.65% of the total weights are owned by the feed-foward networks. This number grows to 47.03% and 48.18% in the big Transformer, for again, a respective vocabulary of size 37000 and 32000.

To evaluate which nodes can be safely pruned without affecting translation quality, we estimate the maximum value x_{max} for each node of the ReLU output over a few hundred steps. This is done on the training set, using the fully trained model and keeping all other weights frozen. These x_{max} are computed before quantizing the ReLU output and do not replace the ones used by the quantization process. Figure 1 shows the histogram of these running estimates for one ReLU layer in the encoder and one in the decoder. All other ReLU layers share the same pattern, where in the encoder there are always multiple x_{max} close to 0, while this not being the case for the decoder.

Once the running estimates are computed, we prune its corresponding node if $x_{max} < z\sigma$ where z is a hyperparameter and σ the standard deviation of the x_{max} of the layer. We empirically found $z = 0.025$ to work well, with higher thresholds causing BLEU to quickly decay. No retraining of the model is performed after nodes have been pruned.

Using this pruning method, we can further compress the Transformer without affecting BLEU scores. Table 8 shows results of our pruning method. Our approach has the advantage of being adaptive, meaning the number of nodes pruned per layer will differ as opposed to a fixed pruning ratio method. For example, in the case of the big Transformer trained on WMT14 EN-FR, 169 nodes were pruned in the first ReLU of the encoder, while in the second, 1226 were pruned. Nodes in the decoder rarely got pruned, at most 4 in the whole decoder.

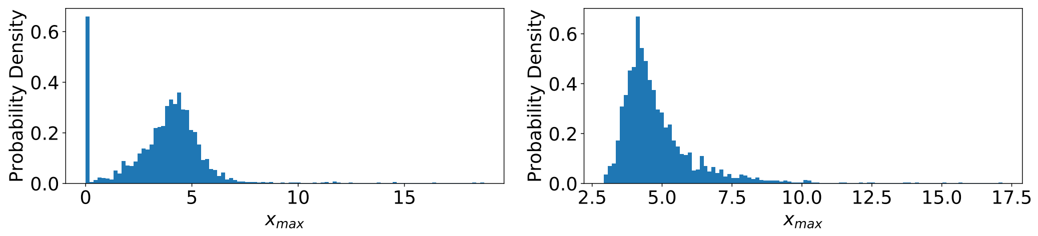

Figure 1: Histograms of the running estimates x_{max} of a ReLU layer in the encoder (left) and decoder (right), one x_{max} per output node of the layer.

The threshold allows us to find the right ratio of nodes to prune per layer instead of the usual: decide the ratio first and then prune. We compared with two such methods, where for each task, we fix the ratio to the global percentage of nodes pruned in the encoder by our method. The first fixed pruning method uses L1-norm to sort nodes to prune in ascending order, while the second sorts nodes using the x_{max}, also in ascending order.

Since x_{max} is a running estimate, results varied per trial. The method only takes a few hundred training steps to perform though, so running many trials is not an issue. We evaluated our method on the validation set a few times until accuracy increased over the non-pruned model. Perplexity and BLEU were then computed on the test set. Reported results are averaged over a few trials. When accuracy was not improving, BLEU would either be the same or decreased by about $0.01-0.02$.

Table 8: Comparison of our adaptive pruning scheme versus fixed rate pruning methods for equal pruning proportions. Total compression is of quantization combined with pruning.

Model	Task	Precision	Method	PPL	BLEU	Nodes Pruned in Encoder FF
Base	EN-DE	8-bit	No pruning	4.39	27.60	0%
			L1-norm, fixed	5.57	23.99	13.57%
			x_{max}, fixed	4.57	27.33	13.57%
			x_{max}, adaptive	4.40	**27.60**	13.57%
		6-bit	No pruning	5.09	26.98	0%
			L1-norm, fixed	6.97	20.81	12.06%
			x_{max}, fixed	5.41	26.20	12.06%
			x_{max}, adaptive	5.09	**26.98**	12.06%
	EN-FR	8-bit	No pruning	2.90	39.91	0%
			L1-norm, fixed	4.38	29.01	9.47%
			x_{max}, fixed	3.18	39.40	9.47%
			x_{max}, adaptive	2.90	**39.91**	9.47%
		6-bit	No pruning	3.38	37.07	0%
			L1-norm, fixed	4.19	31.64	9.62%
			x_{max}, fixed	3.68	36.91	9.62%
			x_{max}, adaptive	3.38	**37.07**	9.62%
Big	EN-DE	8-bit	No pruning	4.24	27.95	0%
			L1-norm, fixed	5.80	22.65	26.39%
			x_{max}, fixed	4.47	27.43	26.39%
			x_{max}, adaptive	4.25	**27.95**	26.39%
		6-bit	No pruning	4.78	26.76	0%
			L1-norm, fixed	7.73	17.32	29.96%
			x_{max}, fixed	4.92	**26.86**	29.96%
			x_{max}, adaptive	4.78	26.76	29.96%
	EN-FR	8-bit	No pruning	2.80	40.17	0%
			L1-norm, fixed	4.16	28.85	28.41%
			x_{max}, fixed	2.91	39.40	28.41%
			x_{max}, adaptive	2.80	**40.17**	28.41%
		6-bit	No pruning	2.87	39.59	0%
			L1-norm, fixed	7.88	15.09	22.66%
			x_{max}, fixed	2.91	39.25	22.66%
			x_{max}, adaptive	2.87	**39.59**	22.66%