Fully Quantized Transformer for Machine Translation

Gabriele Prato
Mila, Université de Montréal
`pratogab@mila.quebec`

Ella Charlaix
Huawei Noah's Ark Lab
`ella.charlaix@huawei.com`

Mehdi Rezagholizadeh
Huawei Noah's Ark Lab
`mehdi.rezagholizadeh@huawei.com`

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 this end, we propose FullyQT: an all-inclusive quantization strategy for the Transformer. To the best of our knowledge, we are the first to show that it is possible to avoid any loss in translation quality with a fully quantized Transformer. Indeed, compared to full-precision, our 8-bit models score greater or equal BLEU on most tasks. 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 only recently proposed (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). Nonetheless, the approach became the state-of-the-art in the field (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 have been proposed, such as multiplicative 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 results in 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 use 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 networks is to lower numerical precision. Consequently, numerical values can be represented using fewer bits (Tang and 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 quantization-aware training strategy for the entire Transformer architecture. 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. In comparison with full-precision, our quantized models score equal or higher BLEU on most tasks. We are, to the best of our knowledge, the first to show that the Transformer architecture can be fully quantized without impairing translation quality. We also perform an ablation study and show that quantizing specific components of the Transformer improves BLEU score.

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 (Jacob et al., 2017) and learned (Zhang et al., 2018) quantization. These methods can be universally applied to any type of neural network. To maintain performance though, specific architectures usually require custom tailored quantization schemes.

Several recent work explore recurrent neural network (Jordan, 1990) quantization. Ott et al. (2016) propose an exponential quantization method for RNN weights. They find ternary and exponential quantization to work well on language modeling and speech recognition, while binary weights seemed ineffective. Hubara et al. (2016) quantize weights and activations of both RNNs and LSTMs (Hochreiter and Schmidhuber, 1997) to 2, 4 and 6-bit. Meanwhile, He et al. (2016) propose modifications to the gates and interlinks of quantized LSTM and GRU (Cho et al., 2014) cells, as well as a balanced quantization method for weights. Wu et al. (2016) successfully quantize a stacked sequence-to-sequence LSTM to 8-bit without any loss in translation quality. Most recently, Wang et al. (2018) propose applying different quantization methods for different RNN components.

With regards to CNNs (LeCun et al., 1989), various works have also explored quantizing these models. Gong et al. (2014) compare matrix factorization, binarization, k-means clustering, product quantization and residual quantization of CNNs. Wu et al. (2015) apply quantization to both kernels and fully connected layers of convolutional neural networks. Rastegari et al. (2016) propose using binary weighted filters on AlexNet (Krizhevsky et al., 2012). Testing their method on ImageNet, they show classification accuracy to be on par with full-precision. For faster inference and training, Zhou et al. (2016) use low bitwidth weights, activations and gradients on CNNs.

Quantization has been applied in tandem with other compression methods. Han et al. (2015) combine pruning, quantization, weight sharing and Huffman coding. In another line of work, Polino et al. (2018) employ quantization with knowledge distillation (Hinton et al., 2015) for higher compression rates. Moreover, Chen et al. (2018) blend quantization with block based low-rank matrix approximation of embeddings.

2.2 Pruning

The pruning of neural networks for model compression has also been largely explored. LeCun et al. (1990) were the first to propose a Hessian based method to prune neural net weights. Hassibi et al. (1994) later improved 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 and Gupta, 2017). To avoid sparse matrices, Liu et al. (2017) prune nodes instead of weights. They apply a penalty in the loss on the γ parameters of batch normalization layers. With a similar objective, 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.

Similarly to quantization, pruning methods have also been adapted to specific architectures. Liu et al. (2015) propose an efficient sparse matrix multiplication algorithm for CNNs. As 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. Lastly, Park et al. (2018) introduce simple recurrent units (SRUs) for easy pruning of RNNs.

3 FullyQT

3.1 Quantization Methodology

Our quantization scheme 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. A brief overview of uniform quantization is given in this section. For more details, we refer the reader to Jacob et al. (2017).

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

$$\mathcal{Q}(x) = \left\lfloor \frac{\text{clamp}(x; x_{min}, x_{max}) - x_{min}}{s} \right\rceil * s + x_{min} \quad (1)$$

$$s = \frac{x_{max} - x_{min}}{2^k - 1} \quad (2)$$

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 clamp function associates all values outside of the

$[x_{min}, x_{max}]$ range to the closest endpoint and $\lfloor \cdot \rceil$ represents rounding to the nearest integer. The value k is simply the bit precision. For example, in the context of 8-bit quantization, $k = 8$.

During backpropagation, we use the straight-through estimator (Hinton, 2012) and set the gradients of clamped values to zero. 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, these are fixed and can thus be quantized once prior to training. The γ weights of Layer-Norms 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. At inference, the softmax does not need to be computed in full-precision. Indeed, the exponential function can instead be replaced with a step function. For the position-wise feed-forward networks, we quantize the output of the ReLUs and of the feed-forwards 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. A visual guide is provided in appendix A.

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 all values being mapped to a single low numerical precision domain.

We use this bucketing method for all weight matrices, with a 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, quotient and output.

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 Softmax numerator and output, 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 and the Softmax's numerator and output, we fix their x_{min} 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 and 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 the parameters' memory usage is reduced, these constantly have to be converted back to full-precision. Achieving quantization of both weights and activations is much more beneficial. The first attempt at doing so for the

Transformer applies 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 resulted in a 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 poorer results in accuracy. Aside from translation, the concurrent work by Zafrir et al. (2019) quantizes the embedding and fully connected layers of BERT (Devlin et al., 2018). The Softmax and LayerNorm operations are kept in full-precision. On the GLUE benchmark, their loss in accuracy is minimal compared to the original model.

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. Then we 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 and experiment with node pruning.

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. Reported perplexity is per token and BLEU was measured with `multi-bleu.pl`[1] on the `newstest2014`[2] test set. We used beam

---

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

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

search with a beam size of 4 and a length penalty of 0.6. Unlike Vaswani et al. (2017), 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 over 2 million training steps. Validation was computed every training epoch. Models were trained once. Our objective 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 and 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 for details).

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. We also evaluate 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). 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. Baselines and FullyQT 8-bit results were averaged over 5 trials. Standard deviation of the BLEU scores did not seem higher for any method and ranged between 0.09 and 0.51. Training with quantization was about twice as slow as with the baselines. As for post-training quantization, the BLEU score was computed on the test set using the best validation performance out of 20 trials. The default approach's *nan* in the EN-FR task is due to numerical instability. By quantizing every operation, zeros in the LayerNorm's denominator are more frequent.

Method	Fully Quantized	Size (Gb) [EN-DE, EN-FR]	Compr.	BLEU EN-DE (2014)	EN-FR	EN-DE (2017)
Vaswani et al. (2017)		[2.02, 1.94]	1x	27.3	38.1	-
Cheong and Daniel (2019)		0.69	2.92x	-	-	27.38
Bhandare et al. (2019)		≥ 0.96	≤ 2.1x	27.33	-	-
Fan (2019)		≥ 0.51	≤ 3.99x	26.94	-	-
FullyQT	✓	[0.52, 0.50]	3.91x	**27.60**	**39.91**	**27.60**

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.

Model	Method	Precision	EN-DE				EN-FR			
			PPL	BLEU	Size (Gb)	Compr.	PPL	BLEU	Size (Gb)	Compr.
Base	Baseline	32-bit	4.95	26.46	2.02	1x	3.21	38.34	1.94	1x
	Default Approach	8-bit	74.04	0.21	0.52	3.91x	*nan*	0	0.50	3.91x
	Post-Quantization	8-bit	4.97	26.44	0.52	3.91x	3.26	38.30	0.50	3.91x
	FullyQT	8-bit	4.94	26.38	0.52	3.91x	3.23	**38.41**	0.50	3.91x
	Post-Quantization	6-bit	6.00	24.84	0.39	5.18x	3.98	35.02	0.37	5.17x
	FullyQT	6-bit	5.09	**26.98**	0.39	5.18x	3.38	37.07	0.37	5.17x
	FullyQT	4-bit	11.96	18.32	0.26	7.66x	48.21	1.59	0.25	7.64x
Big	Baseline	32-bit	4.38	**27.13**	6.85	1x	2.77	**40.54**	6.69	1x
	Post-Quantization	8-bit	4.27	26.55	1.74	3.95x	2.78	39.78	1.69	3.95x
	FullyQT	8-bit	4.57	26.96	1.74	3.95x	2.80	40.25	1.69	3.95x
	Post-Quantization	6-bit	5.12	24.86	1.31	5.24x	3.08	37.92	1.28	5.24x
	FullyQT	6-bit	4.78	26.76	1.31	5.24x	2.87	39.59	1.28	5.24x
	FullyQT	4-bit	33.11	10.22	0.88	7.79x	42.42	2.81	0.86	7.79x

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-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**
	FullyQT	8-bit	6.81	**23.06**	3.53	**33.08**	5.60	29.88
Big	Baseline	32-bit	7.41	22.22	3.57	**32.22**	5.32	30.06
	FullyQT	8-bit	7.17	**22.49**	3.66	31.74	5.35	**30.15**

Table 3: Evaluation of our quantization method on the WMT14 EN-CS, WMT14 RU-EN and WMT14 ES-EN translation datasets.

Results on additional translation datasets can be found in Table 3. All models were trained following the same setup as WMT14 EN-FR and WMT14 EN-DE. Vocabulary size is set to 32k 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.

Looking at all conducted experiments, including section 5.3, translation quality of the 8-bit FullyQT models seems to be on par with full-precision. Most of the time, the highest BLEU was scored by the quantized model. For example in the case of WMT14 EN-DE, the maximum BLEU FullyQT base 8-bit obtained was 26.98, while the baseline's

highest was 26.64. As for the big models, the max FullyQT scored was 27.95, whereas the baseline's was 27.43. We looked at training and validation curves to see if quantization had any effect, but saw no discernible difference.

All models use full-precision biases, s and x_{min}. This amounts to 11.61 Mb in the base models and 23.15 Mb in the big models. In the case of 8-bit, these represent less than 2.35% of the total size. Without bucketing, this would amount to 2.18 Mb and 4.35 Mb respectively. We believe the small increase in model size to be worth it. Indeed, in section 5.2, we show that training without bucketing 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, 8-bit seems like the best choice for most hardware.

5.2 Ablation Studies

To better understand which operations are more sensitive to quantization, we evaluate such effect on single operations of the Transformer. By this, we mean quantizing the operation of a module for all Transformer layers. Table 4 shows results on the WMT14 EN-FR translation task for 8-bit precision. The effect of bucketing was also evaluated. BLEU was computed on the test set after 100k steps of training. In 24 out of 27 experiments, performance was better than our full-precision baseline of 38.34 BLEU. Solely quantizing the LayerNorm's denominator with no bucketing results in poor performance. The latter also cannot be bucketed since all dimensions of the variance tensor vary per batch. To successfully quantize this element without causing any loss in performance, 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 on the WMT14 EN-FR task are presented in Table 5. All models are 8-bit quantized base Transformers. Training procedure is the same as in section 5.1.

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 throughout the whole training, this is not a necessity. Quantization could also be applied later during training. Results for different starting points are compared in Table 6. The earliest we start quantizing is at 100 steps, since we need at least a few steps to assess the running estimates. 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.

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. By the latter, we mean keeping all weights fixed and compute the s and x_{min} over a few hundred steps. This is another advantage, since many trials can be performed in search of the best performing candidate. We found post-quantization BLEU scores to vary by about 0.2 BLEU.

5.4 Language Modeling

To evaluate if our quantization scheme generalizes well to other tasks, we evaluate it on two language modeling datasets: WikiText-2 and WikiText-103. As the setup, we use PyTorch's language modeling toy example[3]. 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, each with different precision. All models consist of two Transformer encoder layers, with the embedding and hidden size set to 200. Multi-Head Attention has two heads with key and value 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. Results are presented in Table 7. Validation 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. See footnote 3 for any extra details.

6 Pruning Useless Nodes

We experiment with node pruning our Transformer models. 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. The latter method usually requires special hardware or software to

---

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

Module	Quantized Activation	No Bucketing		Bucketing	
		PPL	BLEU	PPL	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: Effect of quantizing single activations of the Transformer. Results are on the WMT14 EN-FR test set.

Method	PPL	BLEU
No Bucketing	3.49	37.14
No Gradient Clipping	2549.30	0
No LayerNorm Denominator Quantization	3.22	38.29
8-bit Quantized Weights, Full-precision Activations	3.20	38.36

Table 5: Variations to our quantization scheme evaluated on the WMT14 EN-FR translation task.

Quantization Start (training step)	EN-DE		EN-FR	
	PPL	BLEU	PPL	BLEU
Never quantized	4.95	26.46	3.21	38.34
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: Impact of delaying quantization. Results are on the WMT14 EN-DE and WMT14 EN-FR test set.

leverage sparse weight matrices. 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.

To evaluate which nodes can be safely pruned without affecting translation quality, we estimate 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 3 in the appendix 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. This does not happen in 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 layer's x_{max}. 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 pruning nodes.

Using this method, we can further compress the Transformer without affecting BLEU scores. Our approach has the advantage of being adaptive,

Precision	Size (Mb)	Compression	WikiText-2		WikiText-103	
			Loss	PPL	Loss	PPL
32-bit	243.04	1x	5.65	284.15	5.91	**369.20**
8-bit	61.93	3.92x	5.64	282.67	5.94	377.79
6-bit	46.75	5.20x	5.64	**281.48**	5.93	376.44
4-bit	31.57	7.70x	5.65	284.26	5.94	378.67

Table 7: Evaluation of our quantization method on the WikiText-2 and WikiText-103 language modeling tasks.

Model	Precision	Method	EN-DE				EN-FR			
			PPL	BLEU	Nodes Pruned in Encoder FF	Total Compr.	PPL	BLEU	Nodes Pruned in Encoder FF	Total Compr.
Base	8-bit	No pruning	4.39	27.60	0%	3.95x	2.90	39.91	0%	3.95x
		L1-norm fixed	5.57	23.99	13.57%	4.02x	4.38	29.01	9.47%	3.99x
		x_{max} fixed	4.57	27.33	13.57%	4.02x	3.18	39.40	9.47%	3.99x
		x_{max} adaptive	4.40	**27.60**	13.57%	4.02x	2.90	**39.91**	9.47%	3.99x
	6-bit	No pruning	5.09	26.98	0%	5.25x	3.38	37.07	0%	5.24x
		L1-norm fixed	6.97	20.81	12.06%	5.31x	4.19	31.64	9.62%	5.28x
		x_{max} fixed	5.41	26.20	12.06%	5.31x	3.68	36.91	9.62%	5.28x
		x_{max} adaptive	5.09	**26.98**	12.06%	5.31x	3.38	**37.07**	9.62%	5.28x
Big	8-bit	No pruning	4.24	27.95	0%	3.97x	2.80	40.17	0%	3.97x
		L1-norm fixed	5.80	22.65	26.39%	4.21x	4.16	28.85	28.41%	4.24x
		x_{max} fixed	4.47	27.43	26.39%	4.21x	2.91	39.40	28.41%	4.24x
		x_{max} adaptive	4.25	**27.95**	26.39%	4.21x	2.80	**40.17**	28.41%	4.24x
	6-bit	No pruning	4.78	26.76	0%	5.28x	2.87	39.59	0%	5.28x
		L1-norm fixed	7.73	17.32	29.96%	5.64x	7.88	15.09	22.66%	5.54x
		x_{max} fixed	4.92	**26.86**	29.96%	5.64x	2.91	39.25	22.66%	5.54x
		x_{max} adaptive	4.78	26.76	29.96%	5.64x	2.87	**39.59**	22.66%	5.54x

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

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. Results are presented in Table 8. Reported results are averaged on the test set over a few trials. BLEU varied by about $0.01 - 0.02$.

Other approaches usually decide the ratio first and then prune. We compared with two such methods. For each task, we fix their ratio to the average percentage of nodes pruned by our method and only prune nodes in the encoder. The first fixed pruning method uses L1-norm to sort nodes in ascending weight order, while the second sorts the x_{max}, also in ascending order.

7 Conclusion

We proposed a full quantization strategy for the Transformer architecture. Our objective was to ex-

ploit hardware resources as efficiently as possible, quantizing all operations which could provide a computational speed gain.

With FullyQT, 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 35 experiments, 8-bit quantization performed better than full-precision in 21 cases.

If instead of minimizing inference time, one wants to maximize translation accuracy, then applying quantization to only certain components of the Transformer seems to be the best option. Indeed, our ablation study showed than BLEU score could increase even more when only specific elements of the Transformer were quantized. Further gains might be possible, but supplementary experiments would be necessary to determine the best combination.

We plan on extending 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. 2017. Weighted Transformer Network for Machine Translation. *arXiv e-prints*, page arXiv:1711.02132.

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

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

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

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. 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, editors, *Advances in Neural Information Processing Systems 31*, pages 10988–10998. Curran Associates, Inc.

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

Robin Cheong and Robel Daniel. 2019. transformers.zip: Compressing Transformers with Pruning and Quantization. Technical report, Stanford University, Stanford, California.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. 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*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

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

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

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

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

Chaofei Fan. 2019. Quantized Transformer. Technical report, Stanford University, Stanford, California.

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

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

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

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

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

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

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

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv e-prints*, page arXiv:1712.05877.

Michael I. Jordan. 1990. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pages 112–127. IEEE Press, Piscataway, NJ, USA.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of*

the *2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics.

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

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

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

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

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

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

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

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

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

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

M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. 1993. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, 4(1):53–62.

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

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

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

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

Jinhwan Park, Yoonho Boo, Iksoo Choi, Sungho Shin, and Wonyong Sung. 2018. 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, editors, *Advances in Neural Information Processing Systems 31*, pages 10620–10630. Curran Associates, Inc.

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

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

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

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

C. Z. Tang and H. K. Kwan. 1993. Multilayer feed-forward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727.

Andrew Tierno. 2019. Quantized Transformer. Technical report, Stanford University, Stanford, California.

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

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

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

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

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. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, page arXiv:1609.08144.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8Bit BERT. *arXiv e-prints*, page arXiv:1910.06188.

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

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

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

A FullyQT Visual Guide

Figure 1 and 2.
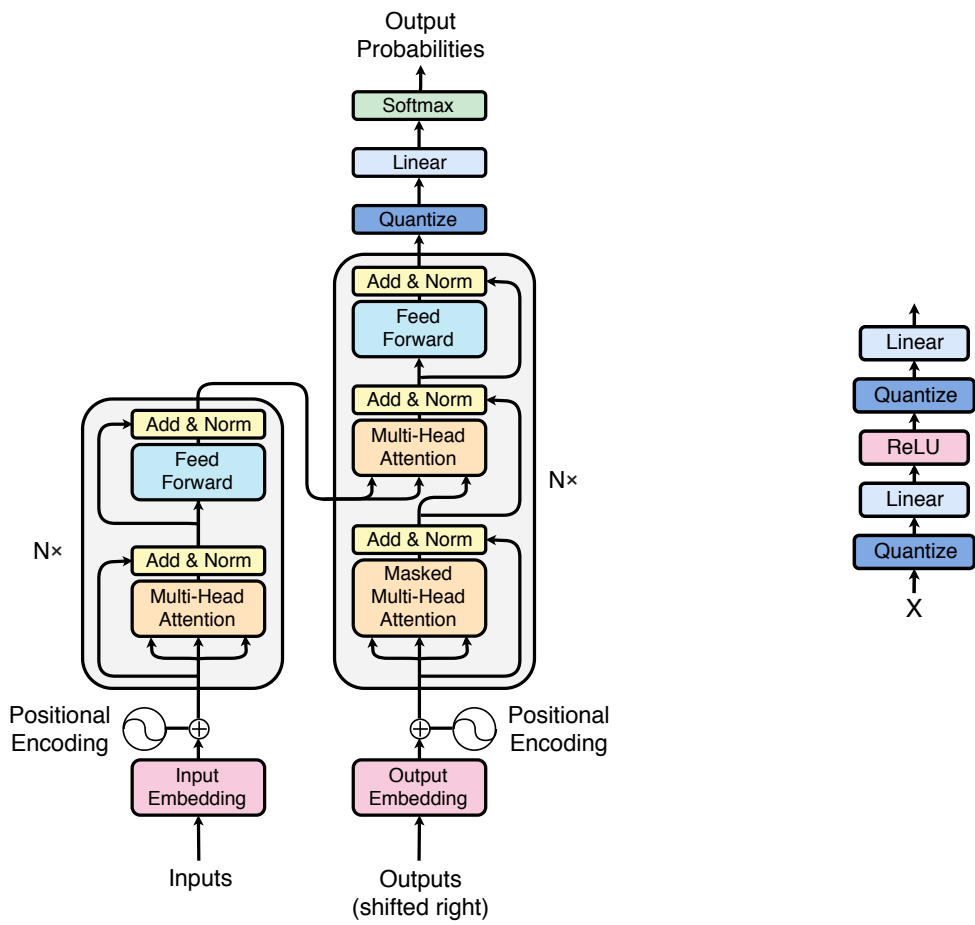
B Node Pruning Running Estimate

Figure 3.

Figure 1: (left) Fully Quantized Transformer, (right) Feed-forward.
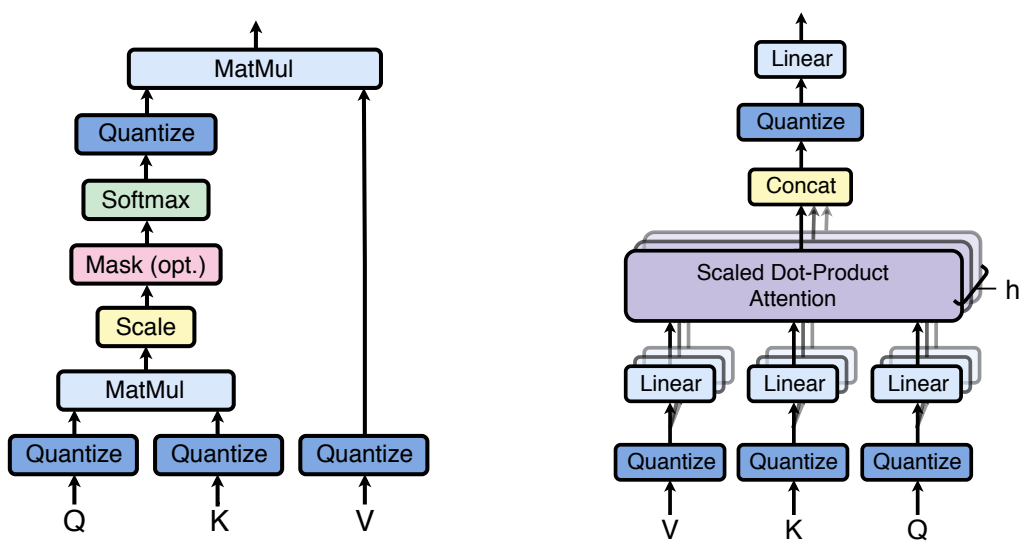

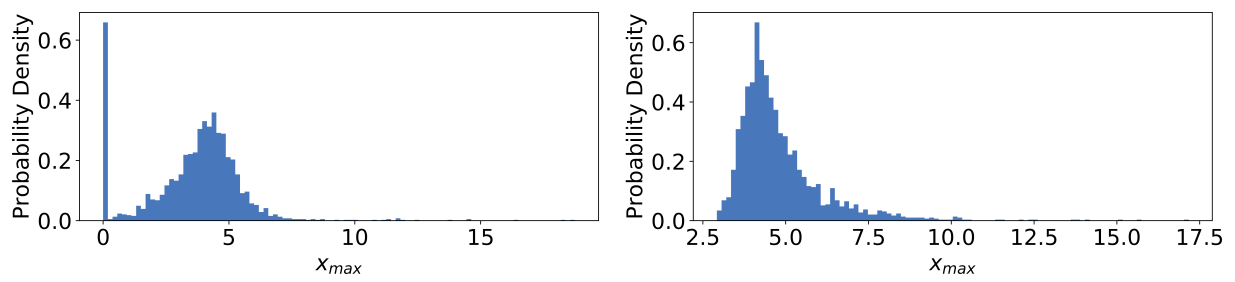Figure 2: (left) Scaled Dot-Product Attention, (right) Multi-Head Attention.

Figure 3: x_{max} histogram of a ReLU layer in the encoder (left) and decoder (right), one x_{max} per output node.