Compressing Neural Machine Translation Models with 4-bit Precision

Alham Fikri Aji and **Kenneth Heafield**
School of Informatics, University of Edinburgh
10 Crichton Street
Edinburgh EH8 9AB
Scotland
a.fikri@ed.ac.uk, kheafiel@inf.ed.ac.uk

Abstract

Quantization is one way to compress Neural Machine Translation (NMT) models, especially for edge devices. This paper pushes quantization from 8 bits, seen in current work on machine translation, to 4 bits. Instead of fixed-point quantization, we use logarithmic quantization since parameters are skewed towards zero. We then observe that quantizing the bias terms in this way damages quality, so we leave them uncompressed. Bias terms are a tiny fraction of the model so the impact on compression rate is minimal. Retraining is necessary to preserve quality, for which we propose to use an error-feedback mechanism that treats compression errors like noisy gradients. We empirically show that NMT models based on the Transformer or RNN architectures can be compressed up to 4-bit precision without any noticeable quality degradation. Models can be compressed up to binary precision, albeit with lower quality. The RNN architecture appears more robust towards compression, compared to the Transformer.

1 Introduction

Neural Machine Translation (NMT) is resource-demanding. Current state-of-the-art architectures, such as the Transformer (Vaswani et al., 2017) or deep RNN (Barone et al., 2017) are typically hundreds of megabytes in size. In a client-based translation system, these large models must be deployed locally, thus consuming network bandwidth for distributing the model, and disk space for storing the model.

Model quantisation has been widely studied as a way to compress model size and increase the inference speed. However, most of this work has focused on convolution neural networks for computer vision tasks (Miyashita et al., 2016; Lin et al., 2016; Hubara et al., 2016, 2017; Jacob et al., 2018). As such, research on model quantisation for NMT tasks remains limited.

We find that the model can be compressed at up to 4-bit precision without sacrificing quality. We first explore the use of logarithmic-based quantisation over fixed-point quantisation (Miyashita et al., 2016) based on the empirical findings that parameter distribution is not uniform, but instead concentrated near zero (Lin et al., 2016; See et al., 2016). The magnitude of a parameter also varies across layers; therefore, we propose an improved method of scaling the quantization centres. We also notice that biases do not quantise very well. However, since biases do not consume a noticeable amount of memory, they can be left unquantised. Lastly, we explore the significance of re-training in the model compression scenario. We adopt an error feedback mechanism (Seide et al., 2014) to preserve the quantisation error rather than discarding it at every update during re-training.

2 Related Work

A considerable amount of research on model quantisation has been performed in the area of computer vision with convolutional neural networks; however, research on model quantisation in the field of neural machine translation is far more limited. Therefore, we will also refer to work on neural models for image processing in this section, where appropriate.

Hubara et al. (2016) quantised the model and activation to binary on a CNN network for various image classification tasks. The binary network achieved near state-of-the-art quality on several easier tasks such as MNIST and CIFAR-10 but achieved sub-par performance on the more challenging ImageNet dataset (losing over 20% accuracy with quantised GoogleNet). Hubara et al. (2017) later reported that with 6-bit fixed-point quantisation, GoogleNet "only" lost 5% accuracy. (Lin et al., 2016) used different bit precisions on various CNN layers, achieving over 20% compression on the CIFAR-10 task.

Since the model's parameters are highly concentrated near zero, Miyashita et al. (2016) opted for logarithmic quantisation. They report an improvement in preserving model accuracy over linear quantisation while achieving the same model compression rate. They also reported negligible accuracy degradation when compressing VGG16 with 3-bit logarithmic quantisation, whereas 3-bit fixed-point quantisation suffered a 6% accuracy drop.

Hubara et al. (2017) compress an LSTM-based architecture for language modelling to 4-bits without any quality degradation but had to scale the hidden layer size by 3. See et al. (2016) pruned an NMT model by removing any weight values lower than a certain threshold. They achieve 80% model sparsity without any quality degradation.

A relevant work with respect to our purposes is the submission of Junczys-Dowmunt et al. (2018) to the Shared Task on Efficient Neural Machine Translation in 2018. This submission applied an 8-bit linear quantisation for NMT models without any noticeable deterioration in translation quality. Similarly, Quinn and Ballesteros (2018) proposed the use of 8-bit matrix multiplication to increase the CPU inference speed of an NMT system.

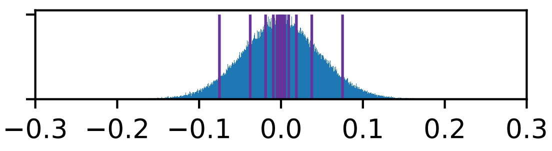
3 Low-precision Neural Machine Translation

3.1 Log-based Compression

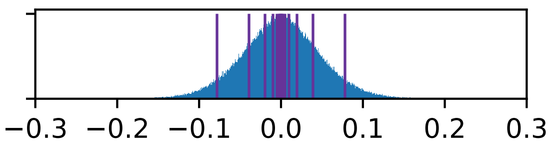
Parameters in deep learning models are normally distributed (Lin et al., 2016; See et al., 2016). Therefore, a uniformly distributed fixed-point quantisation may not fit the parameter distribution. To improve resolution for small values, we adopt logarithmic quantisation following Miyashita et al. (2016) where parameter density is the highest. Figure 1 illustrates the weight distribution and our log-based quantisation.

We use the same quantisation centres for positive and negative values. When compressing to B bits, a single bit represents the sign while the remaining $B-1$ bits represent the log magnitude. The centres are tuned based on the absolute value of the data.

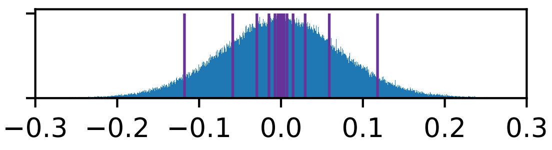
For efficient implementation and because the impact on quality was minimal after re-training, we use log base 2. Log base 2 means that exponentiation amounts to a bit-shift while taking a rounded log (which will be used to quantise a value) amounts to addition followed by finding the leftmost 1 in binary.



(a) First layer encoder self attention weight histogram



(b) First layer decoder self attention weight histogram



(c) First layer decoder context attention weight

Figure 1: Histograms of the first layer's attention key weights. Parameters follow a normal distribution. Vertical lines illustrate the log-based quantisation centres.

We find that tensors might not have the same parameter magnitude. Therefore we also scale the quantisation centres to approximate each tensor better. This approach is different from that of Miyashita et al. (2016), where quantisation centres are not scaled, thus letting every tensor to have the same set of centres. Formally, each quantisation centre takes the form $\pm S2^q$ where S is a scaling factor, and q is an integer in the range $(-2^{B-1}, 0]$. The scaling factor S is selected separately for each tensor in the model.

To minimise the mean squared encoding error, values should be quantised to the nearest centre. Miyashita et al. (2016) find the nearest centre in logarithmic space by taking the log and then rounding to the nearest integer, which is not the same as finding the nearest centre in normal space. For example, their approach will quantise 5.8 to 2^3 instead of 2^2 because $\log_2(5.8) \approx 2.536$, which rounds to 3. In normal space, 5.8 is closer to 2^2 instead of 2^3.

We can implement rounding to the nearest centre in normal space efficiently by multiplying by $\frac{2}{3}$, taking the log and rounding up to the next integer. Let $x \in [2^q, 2^{q+1}]$. Thus:

$$\begin{aligned} x \text{ rounds up to } 2^{q+1} &\iff x > \frac{2^q + 2^{q+1}}{2} \\ &\iff x > \frac{2^q(1+2)}{2} \\ &\iff \frac{2}{3}x > 2^q \\ &\iff \log_2 \frac{2}{3}x > q \end{aligned} \tag{1}$$

Therefore, given a positive x, we can find the quantised magnitude of q with respect to rounding scheme in normal space by:

$$q = \lceil \log_2(\frac{2}{3}t) \rceil \tag{2}$$

Ultimately, given a value v that will be quantised a B-bit logarithmic quantisation. We encode v as $(sign, q)$, where $sign$ represents the sign (1-bit), and q represents the magnitude ($B-1$ bits). Our quantisation functions as follows:

$$\begin{aligned} sign &= sign(v) \\ t &= clip(|v|/S, [1, 2^{1-2^{B-1}}]) \\ q &= \lceil \log_2(\frac{2}{3}t) \rceil \end{aligned} \tag{3}$$

where t is a temporary variable. We first scale the value to the desired range based on scaling factor S. We will discuss more on computing S later. Then, we clip the value into the given range since we have limited quantisation centres. This then decodes to $v' \approx v$ as $v' = \text{sign}S2^q$. In practice, the sign is stored with q.

3.2 Selecting the Scaling Factor

There are a few heuristics to choose a scaling factor of S. Junczys-Dowmunt et al. (2018) and Jacob et al. (2018) scale the model based on its maximum value, which can be very unstable–especially during re-training. Alternatively, Lin et al. (2016) and Hubara et al. (2016) use a pre-defined step size for fixed-point quantization. Our objective is to select a scaling factor S such that the quantised parameters are as close to the original as possible. Therefore, we optimise S such that it minimises the squared error between the original and the compressed parameters.

We propose a method to fit S by minimising the SME. We start with an initial scale S based on the parameters' maximum value. For a given S, we apply our quantisation routine described in Equation 3 to a tensor v, resulting in an approximation of v'. For a given assignment v', we fit a new scale S such that:

$$S = \arg\min_S \sum_i (v'_i - v_i)^2 \tag{4}$$

Substituting v'_i within Eq. 4, we have:

$$S = \arg\min_S \sum_i (sign(v_i) S 2^{q_i} - v_i)^2 \tag{5}$$

To simplify the equation, let a temporary variable a_i to substitute $sign(v_i)2^{q_i}$. Hence we have:

$$S = \arg\min_S \sum_i (a_i S - v_i)^2 \tag{6}$$

To optimise the given objective, we take the first derivative of Equation 6 such that:

$$\begin{aligned} \frac{d}{dS}\sum_i (a_i S - v_i)^2 &= 0 \\ 2\sum_i (a_i(a_i S - v_i)) &= 0 \\ \sum_i (a_i^2 S) - \sum_i (a_i v_i) &= 0 \\ S\sum_i a_i^2 &= \sum_i (a_i v_i) \\ S &= \frac{\sum_i (a_i v_i)}{\sum_i {a_i}^2} \\ S &= \frac{\sum_i (sign(v_i) 2^{q_i} v_i)}{\sum_i (sign(v_i)2^{q_i})^2} \\ S &= \frac{\sum_i (2^{q_i}|v_i|)}{\sum_i 4^{q_i}} \end{aligned} \tag{7}$$

We optimise S for each tensor independently.

3.3 Re-training

We observe later in Section 4.2 that quantisation damages the model. Therefore, we re-train the model after initial quantisation to allow it to recover some of the quality loss. In the re-training phase, we compute the gradients normally with full precision. We then re-quantise the model after

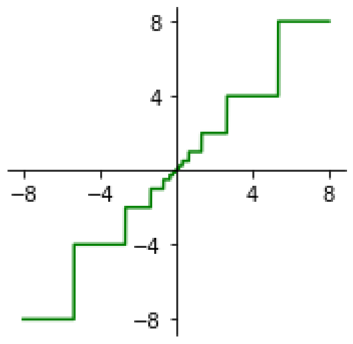


Figure 2: Log-quantization step function.

every update to the parameters, including fitting scaling factors.

Re-quantising the model after every update introduces quantisation errors. The re-quantisation error is preserved in a residual variable and added to the next step's parameter (Seide et al., 2014) before quantisation. We find that re-training fails to work without this mechanism (Section 4.2).

3.4 Handling Biases

We do not quantise bias values in the model. We find that biases are not as highly concentrated near zero when compared to other parameters. Empirically, in our pre-trained Transformer architecture, bias has a higher standard deviation of 0.17 (compared to 0.07 for other parameters). Attempting to log-quantise them used only a fraction of the available quantisation points. In any case, bias values do not consume a lot of memory relative to other parameters. In our Transformer architecture, they account for only ∼0.2% of the parameter values.

3.5 Low-precision Dot Products

To improve the CPU inference speed, we explore training and computing dot products in low precision. Activations coming into a matrix multiplication are quantised on the fly, while intermediate activations (such as tanh) are not quantised.

We use the same log-based quantisation procedure described in Section 3.1 when training the model. However, we only attempt a fixed predetermined scale. Running the slower EM approach to optimise the scale before every dot product would not be fast enough for inference applications.

Training with Quantised Dot Products

Our log-quantised activation is a step function, as illustrated in in Figure 2. Therefore, the derivative of this function is 0 almost everywhere, or undefined in the quantization centres. Thus, we cannot back-propagate through this function normally. Inspired by (Hubara et al., 2017), we utilise a straight-through estimator (Bengio et al., 2013) to set the derivative of the the function to 1, thus enabling training.

Computing Dot Products in Log-space

A dot product operation consists of two sub-operations: element-wise multiplication and sum. In our case, we now have two vectors a and b, both in the form of:

$$a = S_a * [(sign_{j1} \; * \; 2^{j_1}), \dots, (sign_{jn} \; * \; 2^{j_n})]$$
$$b = S_b * [(sign_{k1} \; * \; 2^{k_1}), \dots, (sign_{kn} \; * \; 2^{k_n})]$$

Multiplication is performed by adding the powers. We then add the resulting multiplications together normally, as follows:

$$a \cdot b = S_a * S_b \sum_i (sign_{ji} * sign_{ki} * 2^{j_i + k_i}) \quad (8)$$

Computing power is obtained by using a bit-shift, while computing $sign_{ji} * sign_{ki}$ can be performed using bitwise xor, therefore avoiding expensive multiplication instructions (Miyashita et al., 2016).

4 Experiments

4.1 Experiment Setup

We use systems for the WMT 2017 English-to-German news translation task for our experiment, which differs from the WNGT shared task setting previously reported. We use back-translated monolingual corpora (Sennrich et al., 2016a) and byte pair encoding (Sennrich et al., 2016b) to preprocess the corpus. Quality is measured based on BLEU (Papineni et al., 2002) score using sacreBLEU script (Post, 2018).

We first pre-train baseline models with both Transformer and RNN architectures. Our Transformer model consists of six encoder and six decoder layers with tied embedding. Our deep RNN model consists of eight layers of bidirectional LSTM. Models were trained synchronously with a dynamic batch size of 40 GB per batch using the Marian toolkit (Junczys-Dowmunt et al., 2018). The models are trained until we observe no improvement in 10 consecutive validations. Models are optimised with the Adam optimiser (Kingma and Ba, 2014). The rest of the hyperparameter settings on both models follow the suggested configurations (Vaswani et al., 2017; Sennrich et al., 2017). We use wmt2016 as the test set.

4.2 4-bit Transformer Model

In this experiment subsection, we explore different ways to scale the quantisation centres, the significance of quantising biases and the significance of re-training. We use a pre-trained Transformer model as our baseline and apply our quantisation algorithm on top of that. This experiment focuses solely on the compression ratio. Therefore, models are decompressed back into a 32-bit floating-point value for inference.

Table 1 summarises the results. Using a simple (albeit unstable) max-based scaling has shown to perform better than not using the scale factor. However, fitting the scaling factor to minimise the quantisation squared error produces the best quality. The BLEU score differences between methods of choosing the scaling factor are diminished after re-training.

We can also see improvements by not quantising biases, especially without re-training. Without any re-training involved, we reached the highest BLEU score of 35.47 by using an optimised scale in addition to uncompressed biases. Without bias quantisation, we obtained a $\sim$7.9x compression ratio (instead of 8x) with a 4-bit quantisation. Based on this trade-off, we argue that it is more beneficial to keep the biases in full precision.

Re-training has shown to generally improve quality. After re-training, the quality differences between various scaling and biases quantisation configurations are minimal. These results suggest that re-training helps the model to fine-tune under a new quantised parameter space.

Training Routine

We prepare our 4-bit quantisation model by re-training from a full precision model. We also store the quantisation errors to be considered for the next update. In this subsection, we answer the question of whether it is necessary to perform these steps. We explore the preparation of the 4-bit model if trained from scratch. Similarly, we explore 4-bit model preparation without an error feedback mechanism. For this experiment, we use optimised scaling and 32-bit bias when applying 4-bit log quantisation. Based on the previous result, we left biases unquantised.

The results in Table 2 indicate that fine-tuning from a pre-trained model and error feedback are necessary to produce a high-quality 4-bit model. Removing either of them degrades the quality. BLEU score is dramatically reduced if we train the model from scratch. Likewise, the quantised model is practically unable to learn without the error feedback mechanism. As shown in Table 1, the quantised model achieved a 34.31 BLEU score without re-training. Re-training said model barely improves the BLEU to 34.45 without the error feedback mechanism.

Size Comparison

To demonstrate the improvement of our method, we compare several compression approaches to our 4-bit logarithmic quantisation method with re-training and without bias quantisation. One of the arguably naive methods used to reduce model size is the use of smaller unit size. For Transformer, we set the feed-forward dimension to 512 (from 2048) and the embedding size to 128 (from 512). For RNN, we set the dimension to 320 (from 1024) and the embedding size to 160 (from 512). Using this method, the model size is $\sim$8x smaller and similar to 4-bit quantisation in terms of the model compression rate.

We also introduce the 4-bit fixed-point quantisation approach as a comparison, which is based on Junczys-Dowmunt et al. (2018). However, we made a few modifications to the original approach. Firstly, we apply re-training, which is absent in their implementation. Moreover, we skip bias quantisation. Finally, we optimise the scaling factor instead of the suggested max-based scale.

Table 3 summarises the results, which indicate that reducing the model size by simply reducing the dimension resulted in the worst performance. Our result is in line with (Huang et al., 2019), who show that reducing the model size by using fewer layers degrades quality. Logarithmic-based quantisation has been shown to perform better when compared to fixed-point quantisation using both architectures.

The RNN model seems to be more robust towards the compression. RNN models exhibit reduced quality degradation in all compression scenarios. We hypothesise that the gradients computed with a highly compressed model are very noisy, thus resulting in noisy parameter updates. Our finding is in line with prior research (Chen et al., 2018; Aji and Heafield, 2019), which state Transformer is more sensitive towards noisy training conditions.

Method	Compression	Scaling		
		Unscaled	Max	Optimized
32-bit FP model (Baseline)	-	35.66	-	-
4-bit log model	8x	25.20	28.08	33.33
4-bit log model + 32-bit FP bias	7.88x	34.16	34.29	34.31
4-bit log model + re-training	8x	34.92	34.81	35.26
4-bit log model + 32-bit FP bias + re-training	7.88x	35.09	35.25	35.47

Table 1: 4-bit Transformer quantisation performance for English-to-German translation, measured in BLEU score. We explore different methods of determining the scaling factor as well as skipping bias quantisation and re-training.

Method	FT	EF	Transformer	RNN
Baseline	-	-	35.66	34.28
4-bit	✓	✓	35.47 (-0.19)	34.22 (-0.06)
4-bit	✓	✗	34.45 (-1.21)	33.32 (-0.96)
4-bit	✗	✓	28.54 (-7.12)	28.45 (-5.83)
4-bit	✗	✗	0.05(-35.61)	0.00(-34.48)

Table 2: The model performance (based on BLEU score) of various training scenarios using both Transformer and RNN architectures. FT = Fine-Tuning, EF = Error-Feedback.

Method	Transformer	RNN
Baseline	35.66	34.28
Reduced Dimension	29.03 (-6.63)	30.88 (-3.40)
4-bit fixed point	34.61 (-1.05)	34.05 (-0.23)
4-bit log (Ours)	35.47 (-0.19)	34.22 (-0.06)

Table 3: The model performance (based on BLEU score) of various quantisation approaches using both Transformer and RNN architecture.

4.3 Quantised Dot-Product

Quality Benchmark

We now apply logarithmic quantisation for all matrix multiplication inputs. We use the same quantisation procedure as the parameter. However, we do not fit the scaling factor since it is very inefficient. Hence, we do not scale the quantization centres for the activation. For the parameter quantisation, we use an optimised scale with uncompressed biases based on the previous experiment. Table 4 presents the quality results of the experiment. Generally, we observe quality degradation compared to a full-precision dot product.

Speed Benchmark

Unfortunately, current hardware does not support a 4-bit instruction, thus our dot-product must be emulated using instructions with wider bit widths.[1]

Method	Transformer	RNN
Baseline	35.66	34.28
+ 4-bit model	35.47 (-0.19)	34.22 (-0.06)
+ 4-bit dot-product	35.05 (-0.61)	33.12 (-1.16)

Table 4: Model performance (in BLEU) of model quantisation with dot product quantisation using both Transformer and RNN architecture.

Dot-Product Method	time (ns)
32-bit float	8.45699
8-bit integer	2.08390
4-bit log (16-bit Shift)	3.89595
4-bit log (8-bit Lookup table)	2.51924

Table 5: Time measurement of dot products of 128 elements with different value representations. We use a Cascade Lake processor.

Since there is no 4-bit or 8-bit shift instruction, we emulate 2^q in 16-bit instead. Alternatively, we can choose a lower base, for example $256^{\frac{1}{14}}$ instead of 2 so that the resulting power fits in 8-bit precision. In this case, we can use the 8-bit lookup table instruction `vpshufb` instead.

We benchmark our result with an 8-bit integer dot product based on the `vpdpbusds` instruction (which was introduced in the Cascade Lake to optimise 8-bit matrix multiplication) and a basic 32-bit float dot product using fused multiplication and addition.

Table 5 reports the time required to perform a dot product under different quantisation schemes. 8-bit lookup table is faster than 16-bit. Unfortunately, our 4-bit dot product is inefficient, resulting in it being much slower than an 8-bit dot product. With current hardware, the main advantage over 8-bit quantization is smaller model size, which is

[1] https://github.com/kpu/intgemm/blob/log4-unstable/log4/log4.h

Bit	Transformer		RNN	
	Size (rate)	BLEU(Δ)	Size (rate)	BLEU(Δ)
32	251 MB	35.66	361 MB	34.28
4	32 MB (7.88x)	35.47 (-0.19)	46 MB (7.90x)	34.22 (-0.06)
3	24 MB (10.45x)	34.95 (-0.71)	34 MB (10.49x)	34.11 (-0.17)
2	16 MB (15.50x)	33.40 (-2.26)	23 MB (15.59x)	32.78 (-1.50)
1	8 MB (30.00x)	29.43 (-6.23)	12 MB (30.35x)	31.71 (-2.51)

Table 6: Compression rate and performance of both Transformer and RNN with various bit widths. The compression rate between Transformer and RNN is not equal since they have different biases to parameter size ratio.

of interest for local deployment on mobile devices. Should future hardware also support 4-bit instructions natively, 4-bit models could also improve decoding efficiency.

4.4 Beyond 4-bit precision

With 4-bit quantisation and uncompressed biases, we obtain a 7.9x compression rate. Bit width can be set below 4 bit to achieve an even better compression rate, albeit introducing more compression error. To explore this, we sweep several bit widths. Moreover, we skip bias quantisation and optimise the scaling factor.

Training an NMT system below 4-bit precision remains a challenge. As shown in Table 6, model performance degrades with fewer bits being used. While this result might be acceptable, we argue that the result can be improved. One worthwhile idea would be to increase the unit size in an extremely low-precision setting. We have shown that 4-bit precision performs better compared to the full-precision model with (near) 8x compression rate. Moreover, Han et al. (2015) demonstrated that 2-bit precision image classification can be achieved by scaling the parameter size. An alternative approach is to have different bit widths for each layer (Hwang and Sung, 2014; Anwar et al., 2015).

We also observe the robustness of RNN over Transformer in this experiment since RNN models degrade less compared to the Transformer counterpart. The RNN model outperforms Transformer when compressing at binary precision.

5 Conclusion

We compress the model size in neural machine translation to approximately 7.9x smaller than 32-bit floats by using a 4-bit logarithmic quantisation. Bias terms can be left uncompressed without significantly affecting the compression rate. We also find that re-training after quantisation is necessary to restore the model's performance.

Matrix multiplication can further be quantised, although quality is sacrificed. Unfortunately, 4-bit dot products found in matrix multiplication are slow because current hardware does not natively support the necessary 4-bit instructions.

Acknowledgements

This work was conducted within the scope of the Horizon 2020 Research and Innovation Action *Bergamot*, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825303. Additional support was provided by Intel Corporation. This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (`http://www.csd3.cam.ac.uk/`), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council (`http://www.dirac.ac.uk`).

References

Alham Fikri Aji and Kenneth Heafield. 2019. Making asynchronous stochastic gradient descent work for transformers. *EMNLP-IJCNLP 2019*, page 80.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135. IEEE.

Antonio Valerio Miceli Barone, Jindřich Helcl, Rico Sennrich, Barry Haddow, and Alexandra Birch. 2017. Deep architectures for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 99–107.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, et al. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, pages 103–112.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713.

Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang, Roman Grundkiewicz, and Anthony Aue. 2018. Marian: Cost-effective high-quality neural machine translation in c++. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858.

Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191.

Jerry Quinn and Miguel Ballesteros. 2018. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 114–120.

Abigail See, Minh-Thang Luong, and Christopher D Manning. 2016. Compression of neural machine translation models via pruning. *arXiv preprint arXiv:1606.09274*.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In *Interspeech*.

Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. 2017. The University of Edinburgh's neural mt systems for WMT17. In *Proceedings of the Second Conference on Machine Translation*, pages 389–399.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.