transformers.zip: Compressing Transformers with Pruning and Quantization

Robin Cheong
Department of Computer Science
Stanford University
Stanford, CA 94305
robinc20@stanford.edu

Robel Daniel
Department of Computer Science
Stanford University
Stanford, CA 94305
robeld@stanford.edu

Abstract

The Transformer forms the basis for almost all state-of-the-art pre-trained models in natural language processing but is composed of hundreds of millions of parameters, making the memory costs of their deployment inhibitory. To confront this, we apply a variety of compression techniques to the Transformer architecture. We explore two different quantization methods – a k-means approach as derived in Han et. al [8] and our own modified version of binarization based on Lam [23] – and implement iterative magnitude pruning. We are, to the best of our knowledge, the first to apply quantization methods to the Transformer architecture and the first to compare quantization and pruning on the Transformer architecture. We evaluate on the WMT English to German dataset, and, using solely K-means quantization, we are able to compress the Transformer by a factor of 5.85 while retaining 98.43% of the performance. If we allow performance to degrade to 90%, we show we can compress the Transformer by a factor of 10.02. In addition, we find our proposed quantization method is both significantly faster and gives equal or better performance at the same compression level. Finally, we apply both K-means 1-bit quantization and our proposed binarization scheme to only the self-attention portions of the model in order to better understand the effects of compression on self-attention. We show self-attention layers are highly resistant to quantization, producing almost identical distributions even when constraining the weights to be one of 16 values (4 bit compression).

1 Introduction

Historically, computer vision has seen great success in using pre-trained feature extractors like ResNets [9] or DenseNets [12] to improve performance and generalizability. We now face a similar point in natural language processing in which pre-trained language models can provide significant performance benefits on almost every NLP inference task. Unfortunately, these models are incredibly large, greatly prohibiting their wide usage – it is actually impossible to fine-tune the best performing pre-trained model, BERT-large, using the original settings, on a GPU with 12-16 gigabytes of memory[20]. This poses a large barrier of entry for communities without the resources to purchase either several large GPUs or time on Google's TPUs.

At the same time, however, work in quantization has shown overly parameterized models can be significantly compressed without loss of performance. Han et al. showed in 2016 state of the art vision models like AlexNet and VGG-Net could be reduced to below 5% of their size without loss of accuracy, illustrating a way to significantly reduce computation and memory costs when using these large models as a pre-trained feature extractor [7].

Thus, we aim to apply these model compression techniques to the relatively new Transformer architecture. We choose to focus on the Transformer architecture because it has, to the best of our knowledge, only received limited attention in the compression domain and almost every large pre-trained network relies on components of the Transformer architecture [20, 28].

Though the Transformer is composed almost entirely of linear layers which have been shown to be amiable to deep compression [7], we note Transformers rely on a self-attention mechanism that may be disrupted by model compression. In particular, as far as we are aware, no work has yet studied how quantization affects these layers and the Transformer architecture as a whole. Thus, our contributions are as follows:

- We implement the k-means algorithm derived in [7] and show weights in the linear layers of the Transformer can be compressed into 4 bit representations, achieving an overall 5.85x compression ratio while maintaining 98.43% of performance.
- We also implement a modified version of the binarization algorithm used in [11, 22] and show our method does just as well as or better than k-means compression in producing binarized weights while being significantly faster.
- We implement iterative magnitude pruning as derived in [8] and are able to remove 50% of the weights and retain 93.98% of the BLEU score. In addition, we compare the performances of quantization and pruning.
- We give visualizations of the self-attention distributions post-compression and show compressed models produce almost exactly the same distributions. In addition, we show that, as compression increases, the model produces sharper, more-defined distributions – brighter areas, suggesting a distillation of the representations.

In all, we cover and apply several methods of model compression and show the potential for highly compressed Transformers.

2 Related Works

2.1 Quantization

The quantization literature is extensive, though only in recent years has there been application to neural machine translation. Gong et. al reviewed a broad array of techniques for quantizing models post-training, including matrix factorization, vector factorization, scalar quantization via k-means, and structured quantization using product or residual quantization. They found simple k-means scalar quantization outperformed vector and matrix factorization methods, though improvements could be made using product quantization as well [4].

In addition to quantizing models post-training, many methods exist to train pre-quantized models from scratch. Courbariaux et. al and Rastegari et. al introduced methodologies of training neural networks using only binary weights matching the performance of several top models on MNIST, CIFAR-10, and SVHN [6, 11, 14]. Li et. al extended the approach to ternary networks which use one of three values, -1, 0, 1, and attempt to minimize the Euclidean distance between the original and ternary weights [13].

Recent work on quantizing neural machine translation models has focused on increasing the speed of the models to the point where they can be run on CPUs, primarily using 8 and 16 bit quantization [24, 16]. Though some work exists in compressing other NMT models using quantization [21, 25], as far as we are aware, no form of quantization has been directly applied to the Transformer architecture.

2.2 Pruning

The literature of pruning over-parameterized models is also extensively well studied. Le Cun et. al introduced the concept of removing connections based on some measure of model complexity [1]. Hassabi et. al later showed retraining the model can lead to worse generalization performance since, as the retraining process continues, important weights can incorrectly be removed [2]. Unfortunately, these methods are computationally prohibitive as second derivative computations are expensive.

Thus, Han et. al introduced a more computationally feasible method for pruning connections and relearning weights based solely on the magnitude of the original weights, compressing VGG-16 to less than a tenth of its original size and without loss of accuracy on the ImageNet dataset [8, 3, 5]. Later, Han et. al combined iterative magnitude pruning and k-means quantization and showed the two are actually complementary to each other, producing compression rates far larger than what either can do individually [7]. Likewise, Zafrir et. al applied pruning and integer quantization to an LSTM model and showed a compression rate of 10x with a drop in BLEU score of 2.3 [21].

In recent years, pruning has also been applied to neural machine translation. See at al. pruned LSTMs and experimented with pruning specific layers differently, finding that, surprisingly, class-blind pruning performed best, removing 80% of the connections and suffering no loss in BLEU score [15]. Recently, Gale et. al applied iterative magnitude pruning to the Transformer architecture and found 90% of the weights could be removed while only suffering a drop of 2 in BLEU score, performing at least as well as more complex methods of pruning such as l_0 regularization and variational dropout [27].

2.3 Knowledge Distillation

Finally, other means of compression such as knowledge distillation show impressive performances as well. Hinton et. al use the logits of a larger pre-trained model to efficiently train a much smaller model to the same performance [10]. In comparison to quantization and pruning, knowledge distillation significantly changes the model architecture and allows the model to be represented much more compactly. Specific to neural machine translation, Chia et. al are able to compress the Transformer architecture into a small convolution architecture, achieving a 39x reduction in parameter count [19]. Likewise, Senellart et. al are able to distill the Transformer architecture into a small RNN with a 10x reduction in parameter count and a loss in BLEU score of 2 [26].

3 Approach / Methods

We begin by giving a brief description of the compression task. Given a model M and a task T, we'd like to derive a model M' which has fewer parameters than M but achieves the same performance on task T. In this paper, the task T is English to German sentence translation and our model M is the Transformer architecture.

3.1 Quantization

3.1.1 K - Means Quantization

Algorithm 1 K-Means Quantization

procedure QUANTIZE(w, c) ▷ The original weight and the number of centroids
  Initialize $centroids$ to be c linearly spaced values between $max(w), min(w)$
  $centroids = K - Means(w, centroids)$
  Initialize q_w to be the same shape as w ▷ The quantized representation of w
  for w_i in w **do**
    $q_i = closest(centroids, w_i)$ ▷ Assigns q_i to the index of the closest centroid
  return $q, centroids$

We use the same implementation of k-means quantization as in Han et. al [7]. Specifically, we replace each weight with a corresponding centroid index and keep a table mapping centroid index to centroid value. After performing algorithm 1, we then fine-tune the centroid locations by re-training the model and backpropagating through the centroids. For the forward pass, the index stored for each connection is mapped to a centroid which is then used as the weight. For back-propagation, the gradient with respect to a centroid is defined as the sum of the gradients of each of the weights in the cluster. Thus a quantized layer contains only integer indices and a look up table for the centroid values. We refer readers to [8] for visual illustrations. As another point of detail, we use linear initialization, which was found in Han et al. [7] to outperform other methods. Linear initialization produces centroids that fit well to large outliers which are arguably more important to the model's

performance in comparison to other methods. It does so by equally spacing the initial centroids between the minimum and maximum weights.

3.1.2 Modified Binarization

We first introduce the binarization scheme used in [23] that was motivated by [11]. For clarity, we'll call this Binary Scheme Fixed (BS-Fixed) . As an alternative to k-means quantization, BS-Fixed stores the original weights and *during the forward pass* replaces the values with a masked value of c_1 or c_2, where c_1 and c_2 are fixed and chosen with hyperparameter tuning. Specifically, if w represents our weight matrix, then we replace w with w_{bin} where

$$w_{bin,ij} = \begin{cases} c_1 \text{ if } w_{ij} > 0 \\ c_2 \text{ if } w_{ij} <= 0 \end{cases} \tag{1}$$

While this requires us to keep the full precision weights during training, at the end of training, we can replace the weights with the index of its masked value, thus creating binary representations of the weights. Furthermore, contrary to the k-means approach, this allows weights to be assigned to different centroids in each iteration, giving the model more flexibility.

We note, however, that choosing the values of c_1 and c_2 can be difficult and time-consuming. Thus, we extend this approach by initializing the values of c_1 and c_2 in the same way as in Algorithm 1. Specifically, we initialize c_1 and c_2 by running k-means with two centroids over the weights, and then update c_1 and c_2 using back-propagation, where the gradient is as defined as in k-means quantization. In addition, rather than setting the values based on the sign of the weight, we instead set the values based on whether the weight is greater than the average of c_1, c_2:

$$w_{bin,ij} = \begin{cases} c_1 \text{ if } w_{ij} > \frac{c_1+c_2}{2} \\ c_2 \text{ if } w_{ij} \leq \frac{c_1+c_2}{2} \end{cases} \tag{2}$$

These changes eliminate the need for hyperparameter tuning and allow the model to adjust the locations of c_1, c_2 as necessary. We refer to this as Binary Scheme Flexible or BS-Flexible for short.

3.2 Pruning

We also implement iterative magnitude as described in Han et. al [8]. Iterative magnitude pruning is the process of masking out weights with less than a certain magnitude and retraining the remaining weights. We give a more formal definition in Algorithm 2. We then select the best pruned model based on the performance on a held-out validation set or until a certain percentage of weights have been pruned. Magnitude-based pruning tends to be extremely effective in compressing models because weights with small magnitudes are intuitively less important in performing the target task. Thus, removing these weights by setting them to zero should not affect performance greatly.

Algorithm 2 Iterative Magnitude Pruning

```
procedure PRUNE(M, threshold, dataset, max_iters)
    M = learn(M, dataset)                          ▷ Learn the model on the dataset
    for 1 : max_iter do
        for layer in M do
            for weight in layer do
                if |weight| < threshold then
                    weight = 0
        M = learn(M, dataset)                      ▷ The model now has zero-ed weights.
    return M
```

4 Experiments

4.1 Dataset

We train and evaluate on the WMT English - German translation task. Specifically, we train on all of Europarl, Common Crawl, and News Commentary, validate on the WMT2014 test set, and test on the WMT2017 test set. In total, we have approximately 4.5 million sentences to train on and approximately 3000 sentences to validate and test on. We use a vocabulary size of 32,538. The dataset is available for download here: `http://www.statmt.org/wmt14/translation-task.html`, though the OpenNMT system also gives a pre-processed version here: `https://github.com/OpenNMT/OpenNMT-tf/blob/master/scripts/wmt/prepare_data.sh` which we use in this paper.

4.2 Evaluation Method

To evaluated our compressed Transformers, we compare the compression ratio with the resulting BLEU score, which is the standard metric for evaluation of machine translations.

We can calculate the compression ratio as follows. Given n weights, b bits to represent a float, and c clusters, the compression rate r for a layer is given by the equation from Han et al. [7]:

$$r = \frac{nb}{n\log(c) + c * b} \tag{3}$$

The numerator represents the initial cost of storing the weights (n connections times b bits per connection for the weights), and the denominator represents the quantized cost (n connections times log(c) bits to store each cluster index, and k clusters times b bits per centroid weight).

To calculate compression for quantization, we modify the equation (3) slightly; we multiply the $c * b$ term in the denominator by the layer count l, as we store a lookup table of size $c * b$ for each layer l that we compress. This gives us an accurate estimation of the compression on the layers that we quantized. We then multiply this by the proportion of the parameters that were actually quantized, which is all of the linear layer parameters except the generator layer.

To calculate compression for pruning, we simply take the inverse of the proportion of remaining parameters. For example, for 80% pruning on a layer, there are 20% parameters remaining, yielding $\frac{1}{0.2} = 5$x compression. This is then multiplied by the proportion of the parameters pruned. We pruned all of the linear layer weights, including the generator layer.

We note that we also factor into our compression ratio layer normalization, which was not compressed in either pruning or quantization but is an imperceptibly small portion of the model. We do not factor in word embeddings, whose compression is outside of the scope of this paper but has been explored [23], and whose compression would not illuminate anything about the Transformer model.

4.3 Experimental Details

We evaluate our quantization technique on the Transformer architecture described in Vaswani et al. [18] and implemented in OpenNMT. We train each experiment on either a GeForce GTX 1080 Ti or a NVIDIA Tesla M60. We use a batch size of 256 and aggregate gradients across two batches. We use linear initialization of centroids for the quantized layers. We quantize the layer weights based on the number of bits of the experiment, but we fix the quantization of the bias at 8-bit, which we factor in to our compression calculations. We re-trained the models with quantized layers for 10k iterations which, depending on the method, can take anywhere between 2 to 48 hours. We used a learning rate scheduler that conducted a linear warm-up to 1 over 100 iterations. The remainder of the hyperparameters are kept the same as the ones enumerated by OpenNMT to emulate the Vaswani et al. [18] Transformer. These hyperparameters were chosen based on the initial loss curves, but were not tuned for each individual model.

In addition, we quantize every linear layer except the generator module of the Transformer, which in the OpenNMT implementation is a linear layer with in_features $= 512$ and out_features $= 31538$. We found quantizing this layer resulted in dramatic decreases in performance, likely because the ratio

of parameters to centroids is much smaller than in other layers for a fixed centroid size. To keep the ratio fixed, we'd have to manually adjust the number of centroids for the generator. In addition, k-means over 15,000,000 weights takes an extraordinarily long time. Thus, for simplicity, in this work, we chose not to quantize the generator.

For our pruning experiments, we use the same hardware and hyperparameters. For starting proportion X% and ending proportion Y%, our iterative magnitude pruning procedure pruned X% of each of the pre-trained Transformer layers, began re-training, and pruned $\frac{(Y-X)}{9}\%$ of each of the layers every 1001 iterations. Thus, by the 10,000th iteration, we reached Y% pruning of the model iteratively.

4.4 Results

Table 1: WMT 2017 Test set BLEU and Compression Ratio

Model	BLEU	% Performance	Compression
Baseline model	28.09	100%	1x
K-Means 8-bit quantization	27.38	97.47%	2.92x
K-Means 4-bit quantization	27.65	98.43%	5.85x
K-Means 2-bit quantization	24.94	88.78%	11.69x
K-Means 1-bit quantization	12.07	42.96%	23.37x
BS-Fixed	11.61	41.33%	23.37x
BS-Flexible	12.11	43.11%	23.37x
K-Means 1-bit att-quantization	24.96	88.85%	10.02x
BS-Flexible 1-bit att-quantization	25.54	90.92%	10.02x
Pruned 30% $\longrightarrow$ 50%	26.40	93.98%	2x
Pruned 50% $\longrightarrow$ 80%	25.02	89.07%	5x
Pruned 50% $\longrightarrow$ 90%	9.21	32.79%	10x

We give the BLEU score versus compression results on the 2017 WMT English to German test set for our compression methods in Table 1. Note that the BS-Fixed and BS-Flexible methods are equal in compression to 1-bit quantization. 1-bit att-quantization refers to quantizing solely the self-attention portions of the Transformer.

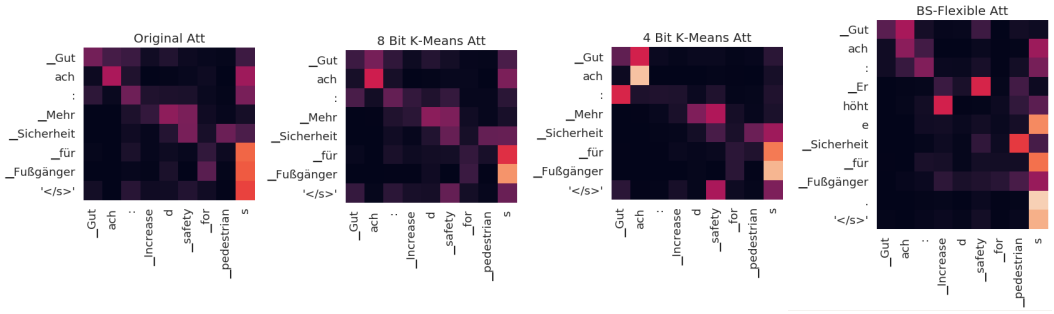

Figure 1: Attention distributions from the last layer of the decoder over the encoder hidden states. From left to right, the distribution of: the original model, the 8-bit k-means model, the 4-bit k-means model, and the model after Binary Scheme Flexible was applied to only the attention layers.

4.4.1 Quantitative Analysis

We're happily surprised by our results – we find that the Transformer architecture is highly resistant to quantization, and are, in essence, able to match the original model up to a 4-bit representation. We were also surprised quantization worked well with our relatively arbitrary choice of hyperparameters, indicating that this method may be robust to hyperparameter initialization. We also saw Binary Scheme Flexible outperform 1-bit k-means in both pure 1-bit compression and quantizing only the

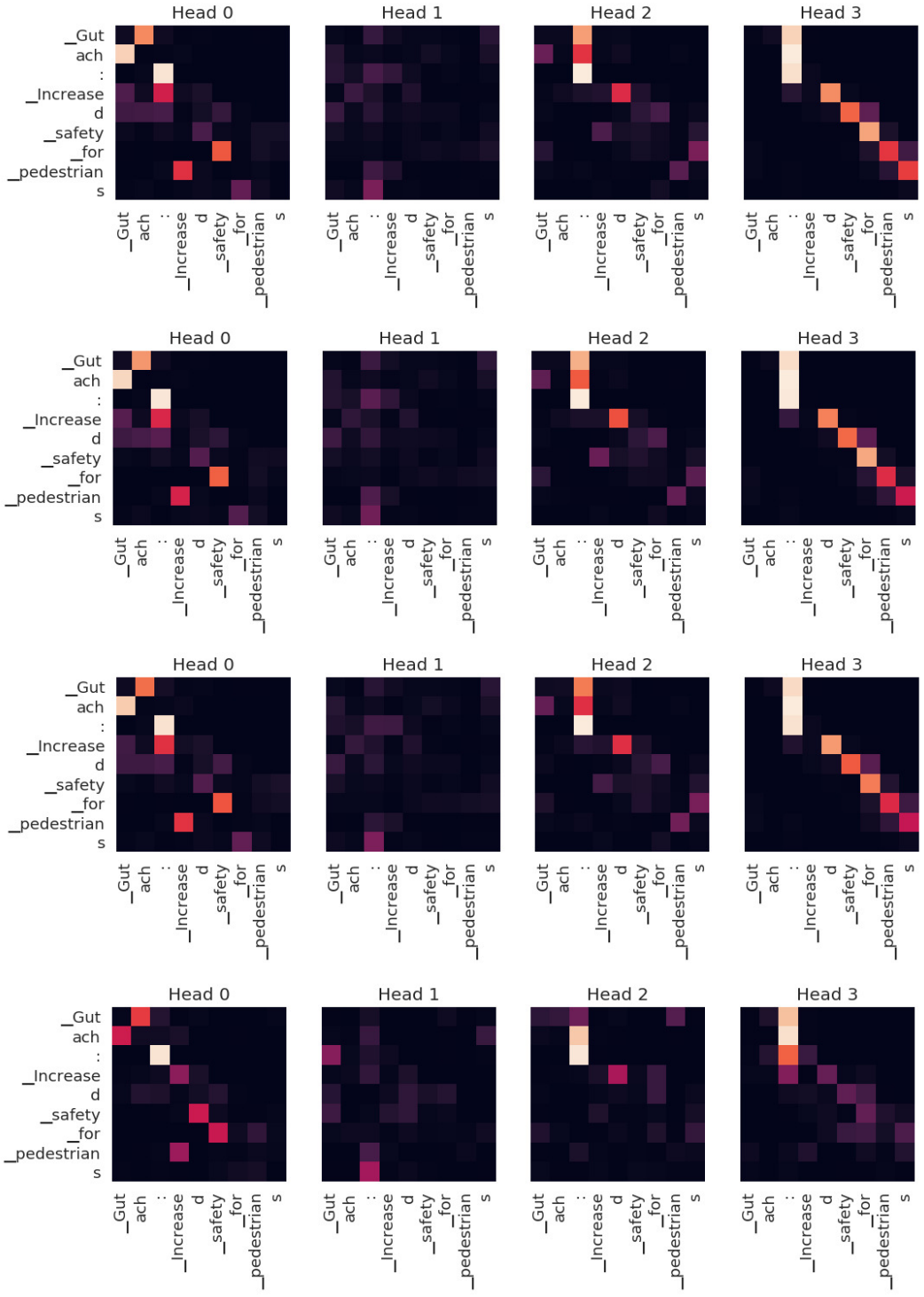

Figure 2: Self-attention distributions over the first layer in the encoder portion of the Transformer. From top to bottom, the distribution of: the original model, the 8-bit k-means model, the 4-bit k-means model, and the model after Binary Scheme Flexible was applied to only the attention layers layers.

attention layers, suggesting not tying the weights to particular centroids improves performance, and outperform Binary Scheme Fixed, indicating learning the values to be a superior method. Furthermore, due to PyTorch indexing procedures, we found our Binary Scheme Flexible to be more than 10x faster the 1-bit k-means procedure, which requires iterating through and replacing all weights.

We were surprised pruning did so much worse than quantization and were unable to replicate the results in Gale et al. in which 90% of the weights were pruned while maintaining approximately 90% of the performance [27]. We believe the discrepancy is due to a lack of hyperparameter tuning. Pruning has two more adjustable components – number of iterations before repruning and starting percentage pruned – that seem likely to heavily affect results.

We are also quite surprised to find that, after binarizing only the attention layers, we are still able to recover 90% of the model's performance. We give a deeper qualitative insight below.

4.4.2 Qualitative Analysis

To better understand the effects of quantization on the model, we show the self-attention distributions before and after compression for three different quantization levels: 8 Bit K-Means, 4 Bit K-Means, and BS-Flexible applied to only the attention layers. We note some interesting trends.

In the last attention layer of the decoder over the encoder hidden states, the attention distribution of the original, 8-bit, and 4-bit model are highly similar (see Figure 1), indicating 4 bit weights, i.e weights that take on one of 16 values, is enough to get the full effects of attention. Even the BS-Flexible attention model produces reasonable attention distributions. In addition, the 4-bit model arguably produces a slightly sharper version of the attention distributions from the 8-bit and original model – there are less faded squares and more squares with bright colors. We interpret this behavior as a kind of purification process in which the representations are forced to contain only the most relevant signals when highly compressed.

We also analyzed the attention distributions in the encoder layers of the Transformer (see Figure 2 and also Figures 3, 4 in the Appendix), and were shocked to find the original, 8-bit, and 4-bit models are almost indistinguishable from one another. This again highlights the idea that self-attention is highly resistant to quantization and could be heavily compressed. In addition, we see the BS-Flexible model is reasonably close to the original as well.

5 Conclusion

In this work, we apply and evaluate a variety of techniques for the compression of the Transformer. We are the first to utilize quantization techniques on the Transformer, and we compare the results from both the K-means quantization established in the literature and our modified binarization schemes to iterative magnitude pruning. We find that significant compression of the Transformer can be achieved with minimal performance loss. Additionally, we find that the application of our binarization scheme to every self-attention portion of the encoder and the decoder achieves almost 91% of the performance of the uncompressed Transformer, despite limiting values in the self-attention matrices to only two options. We produce self-attention visualizations that illustrate that even 4-bit quantization maintains nearly identical self-attention distributions despite the constraints on the values, though we were unable to conduct human comparisons on the translation quality as we do not speak German.

We note that unlike Han et al. [7], we did not combine pruning and quantization for greater gains in compression. However, the fairly steep performance drop for pruning suggest the combination of the two might not have been as effective for our Transformer experiments. On that note, we were not able to conduct significant hyperparameter tuning due to time and computational resources; this most likely explains the worse performance of pruning in our experiments than in [27]. Additionally, we do not apply quantization to the larger generator layer due to performance challenges; pruning the generator and then quantizing may address the large number of weights in this layer. Finally, we do not extend our flexible or fixed binarization schemes to more than 1-bit quantization, but we believe our method is likely to outperform k-means quantization based on our results.

Our contributions demonstrate the potential for significant compression of state-of-the-art NLP models, which would allow for more accessibility in their deployment; in the pursuit of this goal, we open-source our PyTorch code on a repository forked from OpenNMT at `https://github.com/robeld/ERNIE`.

6 Acknowledgements

We'd like to thank Chris Manning for his mentorship throughout the project and Chris Chute for helping to inspire and motivate the idea.

In addition, we'd like to thank the OpenNMT authors for providing a library that we used as the basis for our code [17]. We used the Transformer architecture, the preprocessing, training, and data loading scripts and contributed a compression class `ernie.py` which includes all code for quantization and pruning. In addition, we made several adjustments to get visualizations working. The self-attention visualizations were based on the Annotated Transformer guide released by the OpenNMT team. We also wrote code to do tests for our K-means quantized layers, binarized layers, and pruned layers.

References

[1] Yann Le Cun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1990, pp. 598–605.

[2] B. Hassibi, D. G. Stork, and G. J. Wolff. "Optimal Brain Surgeon and general network pruning". In: *IEEE International Conference on Neural Networks*. Mar. 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572.

[3] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[4] Yunchao Gong et al. "Compressing Deep Convolutional Networks using Vector Quantization". In: *CoRR* abs/1412.6115 (2014). arXiv: 1412.6115. URL: http://arxiv.org/abs/1412.6115.

[5] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *CoRR* abs/1511.00363 (2015). arXiv: 1511.00363. URL: http://arxiv.org/abs/1511.00363.

[7] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *CoRR* abs/1510.00149 (2015). arXiv: 1510.00149. URL: http://arxiv.org/abs/1510.00149.

[8] Song Han et al. "Learning both Weights and Connections for Efficient Neural Networks". In: *CoRR* abs/1506.02626 (2015). arXiv: 1506.02626. URL: http://arxiv.org/abs/1506.02626.

[9] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *arXiv e-prints*, arXiv:1503.02531 (Mar. 2015), arXiv:1503.02531. arXiv: 1503.02531 [stat.ML].

[11] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830 (2016). arXiv: 1602.02830. URL: http://arxiv.org/abs/1602.02830.

[12] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[13] Fengfu Li and Bin Liu. "Ternary Weight Networks". In: *CoRR* abs/1605.04711 (2016). arXiv: 1605.04711. URL: http://arxiv.org/abs/1605.04711.

[14] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). arXiv: 1603.05279. URL: http://arxiv.org/abs/1603.05279.

[15] Abigail See, Minh-Thang Luong, and Christopher D. Manning. "Compression of Neural Machine Translation Models via Pruning". In: *CoRR* abs/1606.09274 (2016). arXiv: 1606.09274. URL: http://arxiv.org/abs/1606.09274.

[16] Jacob Devlin. "Sharp Models on Dull Hardware: Fast and Accurate Neural Machine Translation Decoding on the CPU". In: *CoRR* abs/1705.01991 (2017). arXiv: 1705.01991. URL: http://arxiv.org/abs/1705.01991.

[17] Guillaume Klein et al. "OpenNMT: Open-Source Toolkit for Neural Machine Translation". In: *Proc. ACL*. 2017. DOI: 10.18653/v1/P17-4012. URL: https://doi.org/10.18653/v1/P17-4012.

[18] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[19] Yew Ken Chia and Sam Witteveen. "Transformer to CNN: Label-scarce distillation for efficient text classification". In: 2018.

[20] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[21] Martin Fenner. *Compressing GNMT Models*. Blog. 2018.

[22] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Training Pruned Neural Networks”. In: *CoRR* abs/1803.03635 (2018). arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635.

[23] Maximilian Lam. “Word2Bits - Quantized Word Vectors”. In: *CoRR* abs/1803.05651 (2018). arXiv: 1803.05651. URL: http://arxiv.org/abs/1803.05651.

[24] Jerry Quinn and Miguel Ballesteros. “Pieces of Eight: 8-bit Neural Machine Translation”. In: *CoRR* abs/1804.05038 (2018). arXiv: 1804.05038. URL: http://arxiv.org/abs/1804.05038.

[25] Jean Senellart et al. “OpenNMT System Description for WNMT 2018: 800 words/sec on a single-core CPU”. In: *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 122–128. URL: http://aclweb.org/anthology/W18-2715.

[26] Jean Senellart et al. “OpenNMT System Description for WNMT 2018: 800 words/sec on a single-core CPU”. In: *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 122–128. URL: http://aclweb.org/anthology/W18-2715.

[27] Trevor Gale, Erich Elsen, and Sara Hooker. *The State of Sparsity in Deep Neural Networks*. Feb. 2019.

[28] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2019).

Our mentor is Professor Chris Manning. We have no external collaborators, nor are we sharing our project with another class.

7 Appendix

7.1 Additional Self-Attention Visualizations

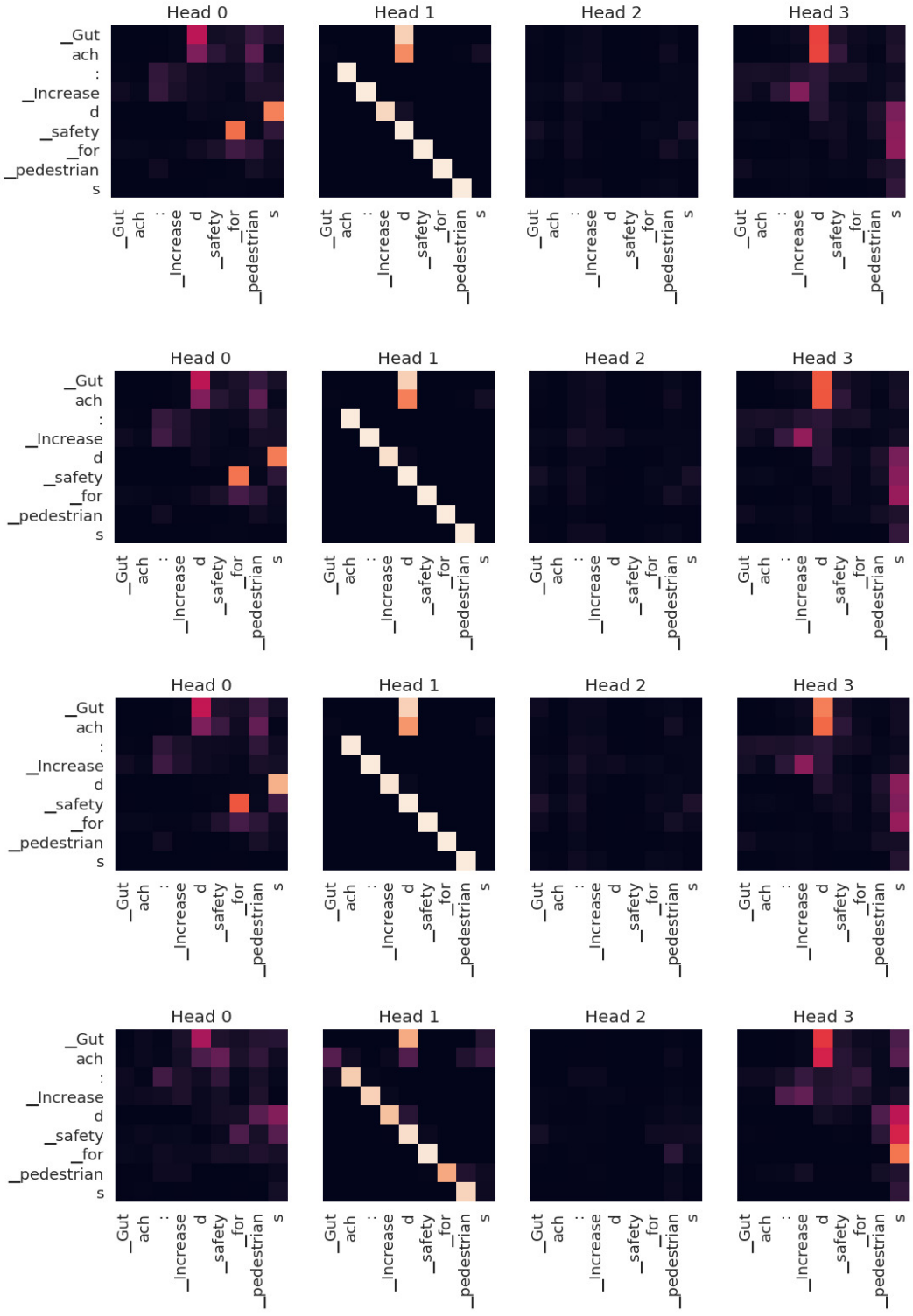

Figure 3: Self-attention distributions over the third layer in the encoder portion of the Transformer. From top to bottom, the distribution of: the original model, the 8-bit k-means model, the 4-bit k-means model, and the model after Binary Scheme Flexible was applied to only the attention layers layers. Again we see the distributions are almost identical even for 4-bit quantization.

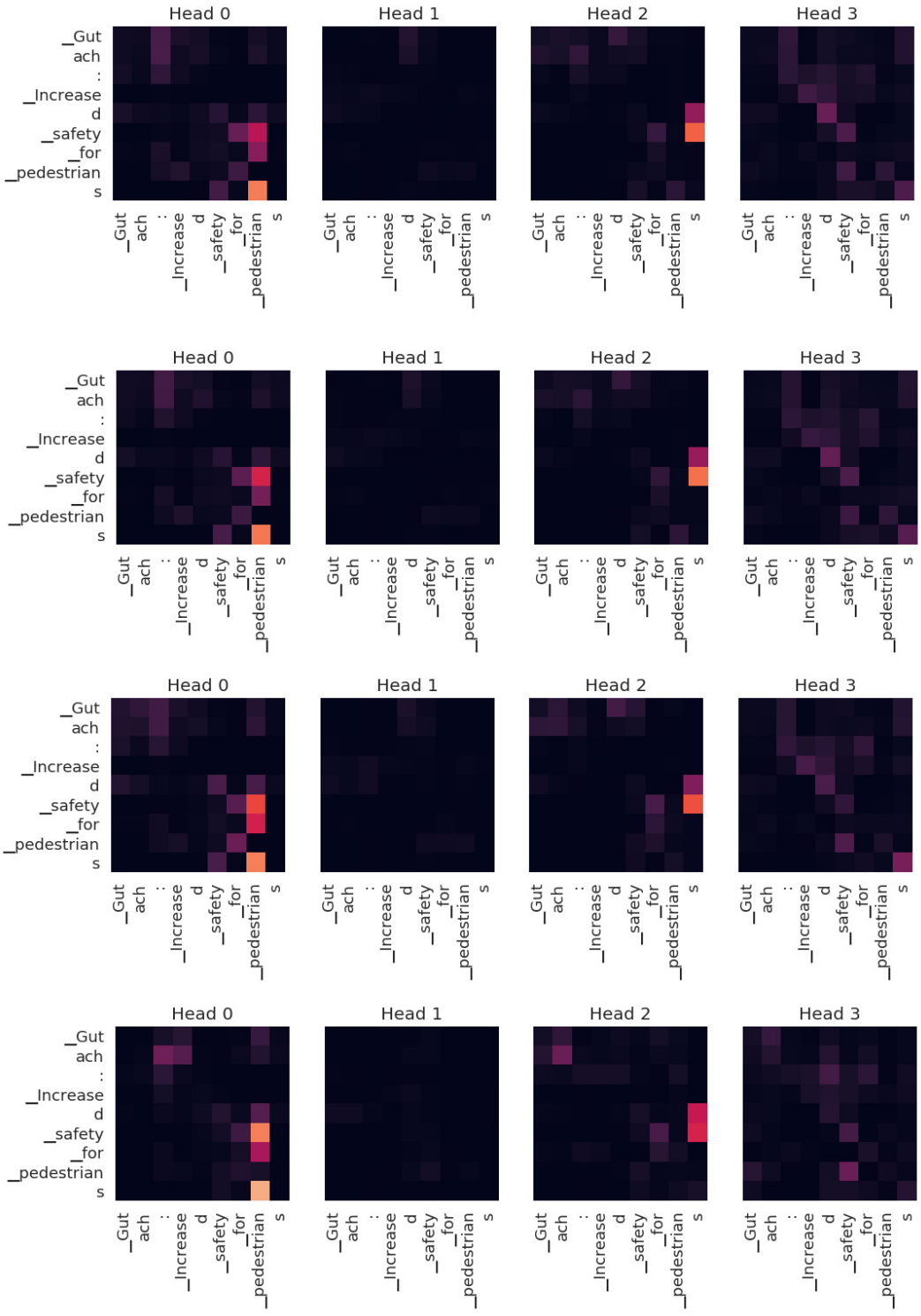

Figure 4: Self-attention distributions over the fifth layer in the encoder portion of the Transformer. From top to bottom, the distribution of: the original model, the 8-bit k-means model, the 4-bit k-means model, and the model after Binary Scheme Flexible was applied to only the attention layers layers.