Applying Neural Network Compression to the Transformer

Abhi Mayur Dubal
University of Utah

UUCS-20-012

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

28 August 2020

Abstract

The Transformer is a popular deep neural network model specialized for natural language processing. Like many deep neural networks, the Transformer is composed of hundreds of millions of parameters that make it favorable to undergo neural network compression techniques. Recent research has shown success with using quantization-aware training as a compression strategy for the Transformer and have delved into understanding which layers are sensitive to quantization. Moreover, existing research has used other compression strategies such as pruning but has failed to explain proper parameter tuning and the effects of these strategies on a per layer basis for the Transformer. This thesis aims to provide an in-depth analysis after applying post-training quantization, automated gradual pruning, and quantization-aware training and understanding their effects on the Transformer in hopes of improving uncompressed model accuracy while achieving high compression rates for the task of machine translation.

APPLYING NEURAL NETWORK COMPRESSION TO THE TRANSFORMER

by

Abhi Mayur Dubal

A Senior Thesis Submitted to the Faculty of
The University of Utah
In Partial Fulfillment of the Requirements for the Degree

Bachelor of Computer Science

School of Computing
The University of Utah
August 2020

Approved:

[illegible]
Rajeev Balasubramonian
Supervisor

H. James de St. Germain
Director of Undergraduate Studies
School of Computing

Mary Hall
Director
School of Computing

ABSTRACT

The Transformer is a popular deep neural network model specialized for natural language processing. Like many deep neural networks, the Transformer is composed of hundreds of millions of parameters that makes it favorable to undergo neural network compression techniques. Recent research has shown success with using quantization-aware training as a compression strategy for the Transformer and have delved into understanding which layers are sensitive to quantization. Moreover, existing research has used other compression strategies such as pruning but has failed to explain proper parameter tuning and the effects of these strategies on a per layer basis for the Transformer. This thesis aims to provide an in-depth analysis after applying post-training quantization, automated gradual pruning, and quantization-aware training and understanding their effects on the Transformer in hopes of improving uncompressed model accuracy while achieving high compression rates for the task of machine translation. [1]

[1]We make our code available at: https://github.com/abhid1/annotated-transformer

For my parents, Mayur and Rita, and my advisor, Rajeev

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Deep neural networks (DNNs) have helped achieve state-of-the-art results on many artificial intelligence tasks such as computer vision, robotics, and natural language processing [28]. Deploying these networks on edge devices has been an attractive goal as more and more of these devices are being built with resources that can handle DNN workloads. Most state-of-the-art DNNs, however, consist of a large number of parameters which make them non-trivial for storing and running on edge devices as they require large memory resources and in turn, would consume a lot of energy due to memory accesses [15]. Therefore, it is crucial to deploy a compressed network that will retain as much accuracy as possible of its non-compressed counterpart and will not exhaust a memory and battery-constrained edge device. In addition to this importance of compressing a DNN, significant efforts have been made into developing hardware accelerators to efficiently process DNNs including compressed DNNs. These accelerators aim to achieve low energy consumption by using the fewest chips possible and/or exploiting data reuse while achieving high throughput for DNN operations [28].

DNN compression strategies aim to lessen the storage demand and energy consumption of neural networks by manipulating their weight parameters. This can be accomplished by removing non-essential weights and lowering the numerical precision of these weights via quantization and pruning, respectively. As the storage space of the network decreases, the energy efficiency increases as there will be fewer memory accesses when weight fetching and performing operations such as dot products. The goal is that after applying DNN compression strategies to a network, the original accuracy of the network is retained as much as possible [15]. This will require thorough analysis on the trade-offs between compression ratio and model accuracy.

The Transformer is a popular, highly parameterized natural language representation

DNN that has heavily influenced many state-of-the-art pre-trained models [9,21,31]. This model is built upon attention mechanisms, deviating from recurrent and convolutional neural networks commonly used by previous DNNs specialized for natural language processing (NLP). Like many DNNs, the Transformer puts a large demand on computational and memory resources via weight fetching and dot products, which makes it difficult to store on mobile systems [15]. With its success in many NLP tasks, the Transformer is one such DNN that is attractive to undergo compression.

Recent research has shown success in compressing the Transformer and retaining its accuracy through applying different quantization-aware training strategies, but has not provided an in-depth analysis on compressing specific layers like [27], who applied their own quantization strategy to the Transformer. Additionally, recent research has not shown much success in terms of retaining accuracy after applying pruning to the network, which we aim to show that pruning works very well for the Transformer. In this thesis, we apply post-training quantization, automated gradual pruning, and quantization-aware training to the Transformer network focused on achieving high compression rates and high model accuracy. We also apply these strategies to different layers of the Transformer to study their effects on the entire performance of the model and study the sensitivity of these layers when they are compressed. We consider compression on the tasks of English to German and German to English translation. We summarize our contributions as follows:

- We give an analysis of post-training quantization, automated gradual pruning, and quantization-aware training on a per layer basis of the Transformer. We also combine certain layers to study the effects in BLEU score and compression rates.
- We provide results from different parameter tuning of the compression schemes we have applied.
- We achieve considerable performance improvement and compression rates for the Transformer when we apply automated gradual pruning and quantization-aware training.

We present background information on the Transformer network, compression, and existing research on compressing the Transformer in the following chapter. We then discuss our methodologies in compressing the Transformer in Chapter 3. We share our experiments and results in Chapter 4 and conclude our work along with giving suggestions for future work in Chapter 5.

CHAPTER 2

BACKGROUND

2.1 The Transformer

The Transformer is a DNN model that formed the basis of many DNN models that have achieved state-of-the-art results for many NLP tasks (e.g., [9], [21], [31]). NLP models prior to the Transformer were built upon recurrent neural networks and convolutional layers; however, the Transformer dispenses those layers and uses attention mechanisms as a means to promote parallelization and computational efficiency [30]. The idea behind attention is to weigh relevant parts of an input being fed into a model and then take such a weight into account while performing a task such as machine translation [12]. In the case of the Transformer, the attention mechanism is slightly extended to compute self-attention which allows the model, when encoding a word in a sequence, to attend to other words in the respective sequence (see **Figure 2.1** on the next page). Self-attention aims to generate effective encodings for sequences as they are passed through each layer of the Transformer and is formally calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where Q, K, V are the query, key, value matrices that are created during the training process of the Transformer and $\sqrt{d_k}$ is the square root of the length of the key matrix.

The standard Transformer architecture, however, computes self-attention many times in each of its encoders and decoders. This idea known as Multi-Head Attention allows the model to encode an effective representation of the current word sequence by expanding its ability to focus on different positions of the entire sequence. The Multi-Head Attention mechanism in the standard Transformer architecture is constructed with eight heads, where each head is responsible for computing matrix multiplications to calculate a final weighted sum (**Figure 2.2** on page 6).

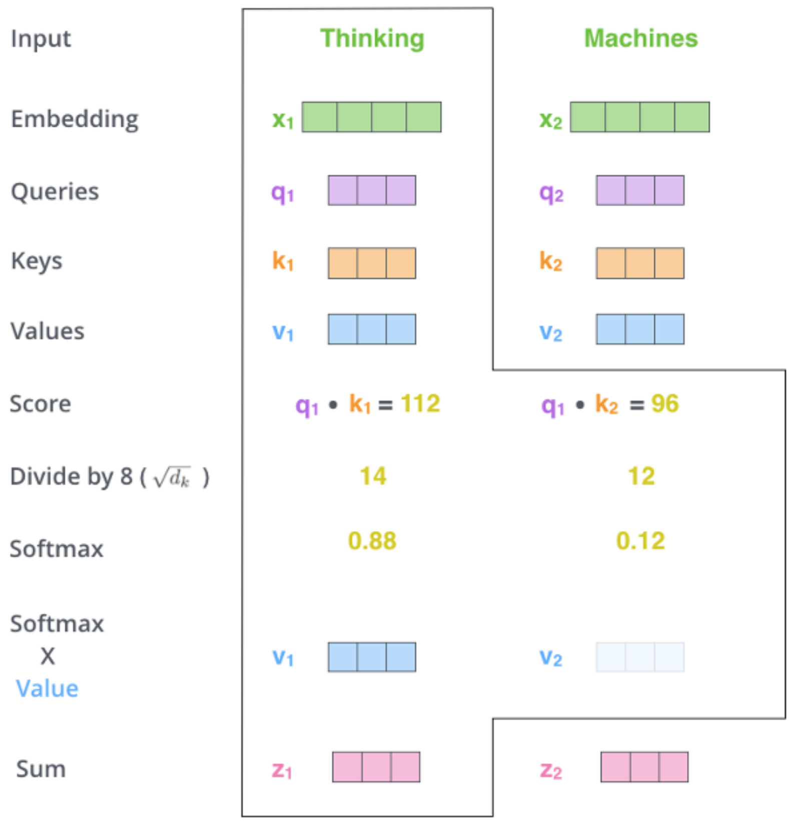

Figure 2.1: Self-attention calculations depicted. An input is first embedded at the bottom-most encoder of the Transformer using a standard embedding algorithm [22]. The embeddings are then multiplied with a query, key, and value matrix that are learned during the training process. The query and key vectors contribute to a score that is computing through a dot product and the score is divided by 8 (square root of 512, the length of the key vector). SoftMax serves as a normalization step to convert the scores and ensure they add up to one. Finally, each value vector is multiplied by the SoftMax score to compute a final weighted sum. This figure was taken from [2].

Before providing input to the first layer of the Transformer, inputs are converted into word embeddings with positional encodings to provide meaningful distances between each word embedding. The Transformer is composed of six identical encoder stacks and six identical decoder stacks. Each encoder first begins with a Multi-Head Attention layer followed by a fully connected feed-forward layer. A residual connection is used between the two layers and each layer's output is normalized using LayerNorm [3]. Each decoder is built identically as the encoder module but employs an additional layer that executes Multi-Head Attention over the output of the last encoder stack. The sole purpose of this layer is to allow the decoder to focus on important positions of the input sequence. In

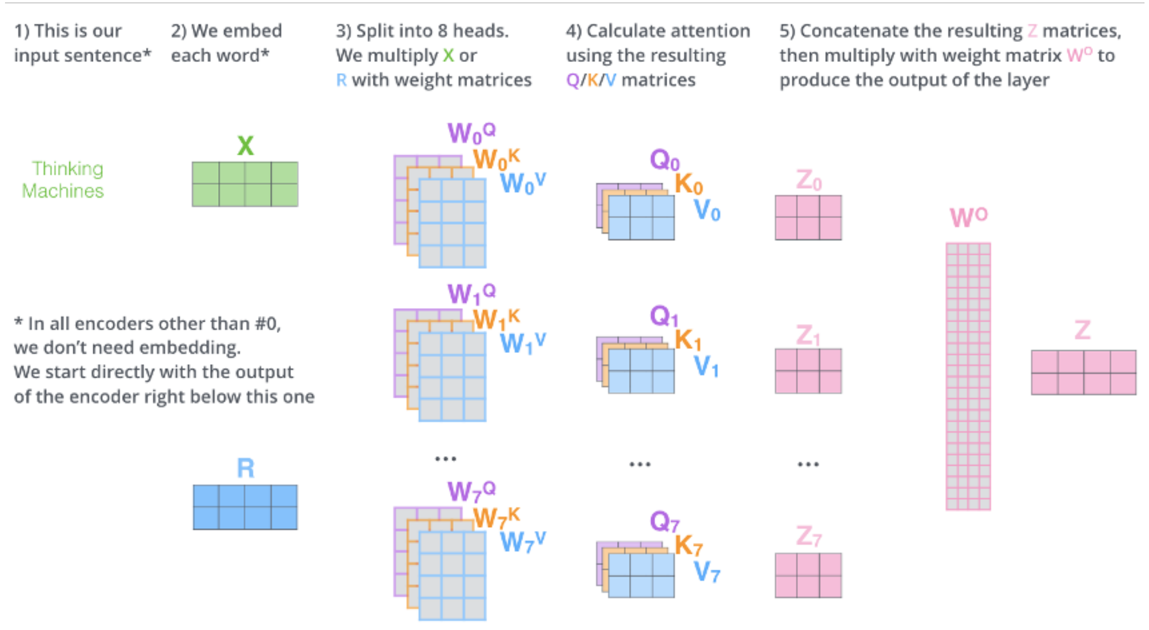
1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
R
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
W^O
Z

Figure 2.2: A depiction of Multi-Head Attention and its calculations. The steps are very similar to **Figure 2.1** on the preceding page, but instead, self-attention is calculated eight times to make up the eight heads. Finally, the eight generated weighted sums are concatenated to insert into the next layer. This figure was taken from [2].

addition, the decoder's self-attention layer is only allowed to attend to earlier positions of the input sequence as subsequent positions are effectively masked out. The architecture of the overall Transformer is depicted by **Figure 2.3** on the next page.

2.2 Compressing DNNs

Running DNN workloads on mobile devices have become appealing as these devices are being equipped with appropriate hardware to run neural networks. Running DNN workloads on mobile devices can improve latency, inference speeds, and privacy as personal data would not need to be sent to servers as much [19]. Due to the large storage demands many DNNs place, effective compression is necessary to ensure these networks do not exhaust the storage and energy components of a mobile device.

Compressing DNNs is an active area of research as more and more networks are achieving high accuracies with non-trivial memory demand. There are many compression strategies that can be applied to neural networks and figuring out the most effective strategies that achieve high compression rates and retain original model accuracy requires extensive experimentation. The most typical compression strategies include, but are not limited to,

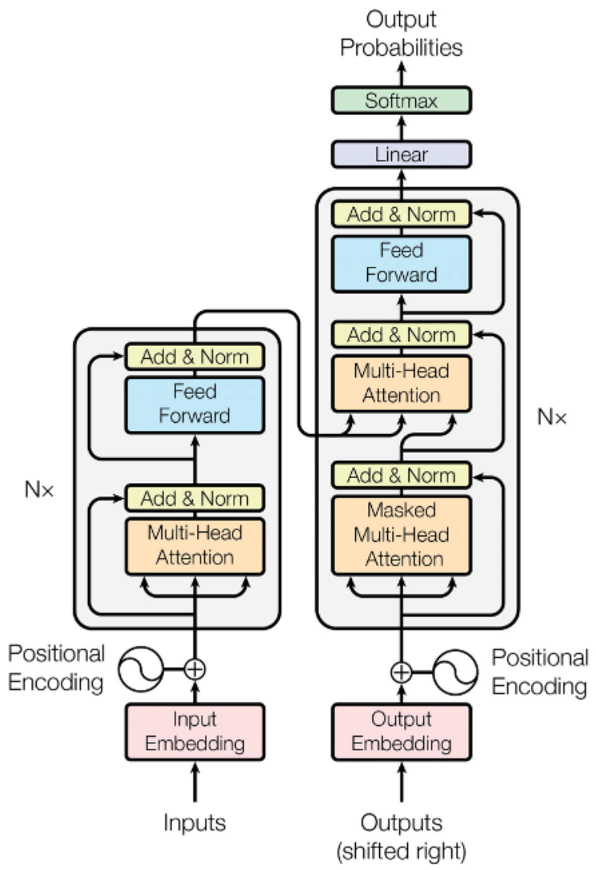

Figure 2.3: The overall Transformer architecture depicted. The base Transformer consists of six identical stacked encoders (left half of figure) and six identical stacked decoders (right half of figure). This figure was taken from [30].

quantization and pruning.

Quantization focuses on reducing the number of bits that represent each weight in a tensor to promote weight sharing where multiple connections in a network share the same weight. The two main methods of quantization include post-training quantization and quantization-aware training. Quantization and weight sharing are typically implemented using a codebook structure that stores shared weights [15]. Post-training quantization performs quantization on a trained model that seeks to reduce CPU and hardware accelerator latency but with some cost to model accuracy. Quantization-aware training, on the other hand, performs quantization during training time that will ultimately produce a quantized model [1]. In quantization research, it is typical to implement quantization while training over post-training as high accuracy is a very crucial goal to retain during compression. There are many strategies of quantization that can be used on a network (e.g., [8], [23], [32], [34]) as literature pertaining to quantization is extensive.

Pruning aims to strategically remove non-essential weights of an overly parameterized DNN. Like quantization, there are many pruning strategies that can be applied to a network (e.g., [14], [20], [24], [33]). Pruning can be applied after training a model; however, most research involving this form of compression has suggested it to be applied during training. Aside from inducing sparsity to promote high compression, pruning can also help a model better generalize to ultimately improve its accuracy [18].

2.3 Compressing the Transformer

Relevant research on compressing the Transformer has seen success applying quantization-aware training. FullyQT is a quantized Transformer developed by Prato et al. that has achieved impressive compression rates using a quantization-aware training approach. Their approach involves using a uniform clamping scheme where elements of a tensor are either the max or min value of the tensor as shown below:

$$Q(x) = [\frac{clamp(x; x_{min}, x_{max}) - x_{min}}{s}] * s + x_{min}$$

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

FullyQT quantizes most operations of the Transformer including matrix multiplications, divisions, and all the activations and weights of the Transformer excluding biases. They also apply post-training quantization using the same approach and found that there was only a slight variation in BLEU (see methodology section) [27].

[11] applied range-based linear and binary quantization during training of the Transformer. Range-based linear quantization scales a full floating point 32-bit value to a specified bit integer form:

$$X_q = round[(x_f - minx_f) * \frac{2^n - 1}{max_{x_f} - minx_f}]$$

Binary quantization simply maps weights to a value of negative one or positive one and can be formalized as follows:

$$binarize(x_f) = sign(x_f)$$

Due to the non-differentiability of range-based linear and binary quantization, [11] applied the straight-through estimator (STE) during backpropagation using an identity function [4]. 8-bit quantization of the Transformer saw a small degradation in BLEU score and binary quantization of weights performed slightly better than 4-bit quantization of weights and activations. Fan concluded that the Transformer is sensitive to quantization for its activations.

This idea is supported by [29], who applied a linear and fixed-point quantization scheme from [16], shown below to the feed-forward network during training of the Transformer:

$$LinearQuant(x, bitwidth) = clip(round(\frac{x}{bitwidth})xbitwidth, minV, maxV)$$

In addition, [29] binarized the inputs of the query and key vectors of the Multi-Head Attention layer. Linear and fixed-point quantization performed poorly as the compressed models' BLEU scores were significantly impacted compared to their uncompressed counterpart.

Alongside quantization, Cheong and Daniel apply iterative magnitude pruning to the Transformer. They performed k-means quantization (see **Figure 2.4** on the following page) where they created linearly-spaced centroids [15] and mapped weights to the closest centroid. They then retrained the Transformer model and backpropagated using the centroid values. They also performed two binarized forms of quantization. Their first method, also shown below, sets weight tensors to one of two chosen values prior training based on their sign bit and their second methods sets weight tensors to one of two chosen values prior training based on comparison to the average of the two values. They also performed two binarized forms of quantization. Their first method, shown below sets weight tensors to one of two chosen values prior training based on their sign bit:

$$w_{bin,ij} = \begin{cases} c_1 \, if \, w_{ij} > 0 \\ c_2 \, if \, w_{ij} \leq 0 \end{cases}$$

and their second methods sets weight tensors to one of two chosen values prior training based on comparison to the average of the two values:

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

For their pruning experiment, they applied the algorithm in **Figure 2.5** on the next page and selected the pruned Transformer model that performed well on the validation set or

Algorithm 1 K-Means Quantization

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

Figure 2.4: K-Means Quantization applied by Cheong and Daniel. Centroids are first initialized and linearly spaced and then weight tensors are mapped to their respected centroids following the K-Means clustering approach [7].

Algorithm 2 Iterative Magnitude Pruning

```
1: procedure PRUNE($M, threshold, dataset, max_iters$)
2:     $M = learn(M, dataset)$                    ▷ Learn the model on the dataset
3:     for 1 : $max\_iter$ do
4:         for $layer$ in $M$ do
5:             for $weight$ in $layer$ do
6:                 if $|weight| < threshold$ then
7:                     $weight = 0$
8:         $M = learn(M, dataset)$              ▷ The model now has zero-ed weights.
9:     return $M$
```

Figure 2.5: The pruning algorithm applied by Cheong and Daniel. For specific layers of the Transformer model, if the absolute value of a weight element does not meet the specified threshold, it is set to 0 [7].

until a certain number of weights have been removed. They found that 4-bit k-means quantization performed the best with a 1.57% accuracy loss and a compression ratio of 5.85x. Their best pruned model removed 50% of weights less than a certain threshold and achieved a 6.02% accuracy loss with a compression ratio of 2. Their pruned model, however, was not able to replicate the results of Gale et al. where they achieved 90% sparsity through iterative magnitude pruning with an estimate of 7% accuracy loss [7,13].

CHAPTER 3

METHOD

In this chapter, we begin by explaining our problem statement, the implementation we followed, the measurements we used for analyzing the performance of our implementation, the Distiller compression library, and compression strategies we applied to the Transformer.

3.1 Problem Statement

As mentioned in the introduction, our work focuses on applying compression algorithms to the Transformer architecture to achieve high compression rates and retain high model accuracy. More formally, given a Transformer model M and a task T, our goal is to derive a new Transformer model M′ that has fewer parameters and performs identically on task T [7]. We chose T to be the tasks of English to German and German to English translation. To understand the sensitivity of the Transformer from certain compression algorithms, we apply such algorithms on a per layer basis followed with compressing the entire model.

3.2 Harvard NLP - The Annotated Transformer

The Annotated Transformer is a popular and efficient implementation of the original Transformer developed by [17] from the Harvard NLP research group. The implementation is around 400 lines of PyTorch code and is based on the OpenNMT implementation of the Transformer. The main purpose of this implementation is to provide a comprehensive view of the Transformer architecture and a starting place for researchers to adapt the implementation. The code can process around 27,000 tokens on 4 GPUs.

3.3 SacreBLEU

SacreBLEU is a Python library developed by [26] that aids in calculating BLEU scores. A BLEU score is calculated by comparing n-grams of the candidate translation to the n-grams of a reference translation and is position-independent. Research work that report BLEU scores often do not report the parameters used to calculate such scores and makes it difficult to replicate results for intrigued researchers. SacreBLEU differs from other metrics such as Moses' scoring scripts as it does not require the user to handle reference tokens and applies its own preprocessing. This prevents ambiguity in reference pre-processing which can affect the way a BLEU score is calculated. In addition, the library makes it easy to report parameters used when calculating BLEU scores to aid in replicating results in the research pipeline.

3.4 Distiller

Distiller is an open-source Python package developed by Intel AI Labs that aids in performing DNN compression research. The goal of Distiller is to improve DNN design and optimizations to promote fast and energy-efficient neural networks. Distiller's algorithms are primarily related to quantization and sparsification through pruning and regularization. It allows control over applying compression algorithms using a scheduler mechanism and YAML file configuration. Categories of compression include, but are not limited to, weight regularization, pruning, post-training quantization, quantization-aware training, knowledge distillation, and more [34]. We now begin discussing the different compression techniques we chose to apply to the Transformer using Distiller.

3.4.1 Post-Training Quantization

We chose to apply unsigned asymmetric and symmetric range-based linear, post-training quantization to the Transformer. Range-based linear quantization (see section 2.3) multiplies a float value with a scale factor where the scale factor is computed from the min and max values of a tensor. There are two modes of range-based linear quantization offered by Distiller: asymmetric and symmetric. In asymmetric mode, the min and max of the float range are mapped to the min and max of the integer range using an offset in addition to the scale factor. In symmetric mode, an offset is not used and the maximum absolute

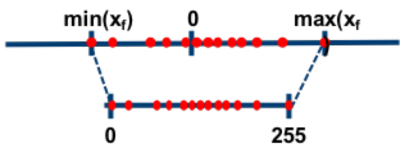

Figure 3.1: The range of unsigned asymmetric quantization. The min and max float are mapped to the min and max of the integer range. This figure was taken from [34].

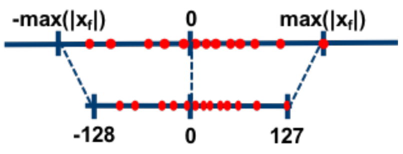

Figure 3.2: The range of symmetric quantization. The absolute value of the min and max float are mapped to the min and max of the integer range. This figure was taken from [34].

value between the float range is mapped to the integer range. The tradeoff between these two modes is simplicity and utilization. Asymmetric quantization uses the full quantized range where symmetric quantization does not if float range gravitates to a specific range. Although this is the case, symmetric quantization is simpler in terms of implementation than asymmetric quantization. These two modes are portrayed in **Figure 3.1** and **Figure 3.2** [34].

3.4.2 Automated Gradual Pruning

We also apply automated gradual pruning formulated by [33] to our implementation. Automated gradual pruning does not require much hyperparameter tuning and does not make assumptions of a DNN which makes it a widely applicable compression technique. Automated gradual pruning is applied during the training process of a network where a given initial sparsity value, s_i, is gradually increased to a final sparsity value, s_f, over a given number of training steps, n. The pruning frequency, Δt, is preset before training. A binary mask variable is added for each layer that is the same shape as the weight tensor and determines the weight tensors that are involved in the forward pass. These masks are updated every Δt steps. The weight tensors that were masked and involved in the forward pass do not get updated during back-propagation. The compression algorithm is shown below:

$$s_t = s_f + (s_i - s_f)(1 - \frac{t - t_0}{n\Delta t})^3 \, for \, t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\}$$

One of the objectives of automated gradual pruning is to prune weights intensely at the beginning of training to remove redundant connections and gradually reduce pruning intensity as connections lessen. Zhu and Gupta saw that after applying automated gradual pruning, sparse models performed better than dense models which suggests that pruning can act as a regularization technique.

3.4.3 Quantization-Aware Training

Next, we consider quantization-aware training for our implementation. Quantization-aware training is a much favorable approach than post-training quantization as it focuses on minimizing loss from "aggressive" quantization. In Distiller, a full-precision copy of weights are maintained during the training process; however, only the quantized weights are using during inference time. Like post-training quantization, we apply an asymmetric unsigned and symmetric range-based linear formulation of quantization-aware training to our implementation (see **Figure 3.1** and **Figure 3.2**).

CHAPTER 4

EXPERIMENTS AND RESULTS

We focus this chapter on detailing the experiments we conducted and the results we collected from applying compression techniques to the Transformer. As recap, our goal is to achieve high compression rates for the Transformer network and retain or improve model accuracy for machine translation. In order to suggest an optimal compression technique, we find it necessary to conduct a per layer analysis, perform compression parameter tuning, and repeat our experiments with a larger parameterized Transformer. We begin this chapter by discussing our experiment setup.

4.1 Setup

Our experiments were conducted using the Multi30k [10] and IWSLT English and German data sets [6]. The Multi30k data set consists of 29,000 training data, 1,014 development data, and 1,000 test examples with the task of translating English sentences into German. The IWSLT data set we use consists of 196,546 training examples, 1,305 test examples, and 992 validation examples with the task of German to English translation. For the IWSLT data set, we limit the vocabulary set to contain words that appear at least twice. We experiment with three compression strategies, namely, post-training quantization, automated gradual pruning, and quantization-aware training. Each strategy was tested on each data set and we analyze performance and compression rates on a per layer basis as well as the overall model. We apply compression to the entire Transformer architecture, but chose not to compress the sublayers as the sublayers make up a very small portion of the entire Transformer [7]. We use the Harvard NLP's PyTorch [25] implementation of the Transformer and did not use beam search which was used by [30]. We trained the Transformer implementation using a single 16 GB Nvidia Titan X GPU with a batch size of 3000 for both data sets. We used the Adam optimizer as expressed by [30] and used Distiller as our resource for applying the three compression strategies.

We measure accuracy using SacreBLEU as detailed by [26]. We measure compression for our quantization experiments as follows:

$$r = \frac{n_m * b}{n_c * b_c + (n_m - n_c) * b}$$

Where n_m is the total number of parameters of the Transformer model, b is the number of uncompressed bits (32 bits for floating point tensors), n_c is the number of parameters being compressed, and b_c is the desired bit compression. For our pruning experiments, we measure compression using:

$$r = \frac{n_m}{NNZ}$$

Where NNZ is the number of non-zero parameters in the model. Because Distiller only simulates compression without considering hardware resources, we did not consider improvements in training and inference times after applying compression for all of our experiments.

Our Transformer baseline for the Multi30k data set was trained with 20 epochs and around 484 steps per epoch which achieved a BLEU score of **34.713**. This model contains **63,736,903** parameters and was trained using a source vocabulary set of 18,657 words and target vocabulary set of 9,799 words. The baseline for the IWSLT data set was trained with 50 epochs with 1,966 steps per epoch and achieved a BLEU score of **17.637**. This model contains **106,139,542** parameters as we restricted the the vocabulary set to a frequency of 2 for frequent words. The vocabulary set for this baseline had a source vocabulary set of 55,704 words and a target vocabulary set of 32,662 words.

4.2 Experiment One - Post-Training Quantization

We apply asymmetric and symmetric post-training range-based linear quantization to the Transformer architecture. To accomplish this, we simply take the trained Transformer baselines that performed the best on the validation sets out of 20 epochs and use Distiller to convert specified layers with quantization wrappers. By default, the wrappers are tuned to quantize with 32 bit precision for activations, weights, bias, and accumulators. We change the default precision and experiment with 16, 8, 4, 3 bits of precision for activations, weights, and biases. We report our results in Tables 4.1 and 4.2:

	FF	ATT	EMB	GEN	FF,EMB,GEN	FULL
16-bit Asym.	34.505	34.541	34.505	34.702	34.706	34.631
	1.22x	1.159x	1.118x	1.038x	1.482x	1.860x
16-bit Sym.	34.505	34.541	34.505	34.55	34.55	34.505
	1.22x	1.159x	1.118x	1.038x	1.482x	1.860x
8-bit Asym.	34.431	34.528	34.734	34.467	**34.752**	34.540
	1.421x	1.286x	1.207x	1.063x	**2.115x**	4x
8-bit Sym.	34.510	34.566	34.474	34.608	34.687	**34.575**
	1.421x	1.286x	1.207x	1.063x	2.115x	**4x**
4-bit Asym.	34.322	33.401	34.374	34.037	33.622	32.086
	1.529x	1.351x	1.25x	1.074x	2.597x	8x
4-bit Sym.	33.887	32.722	34.573	34.037	27.276	24.849
	1.529x	1.351x	1.25x	1.074x	2.597x	8x
3-bit Asym.	34.362	27.185	34.522	29.188	27.924	18.938
	1.588x	1.368x	1.261x	1.077x	2.754x	10.614x
3-bit Sym	33.285	22.045	34.431	8.673	16.334	4.803
	1.588x	1.368x	1.261x	1.077x	2.754x	10.614x

Table 4.1: The Multi30k post-training quantization results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. FF = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. We saw 8-bit asymmetric quantization of the feed forward, embedding, and generator layers to perform the best in retaining model accuracy. We also show that we can achieve 4x compression and incur slight BLEU score loss when compressing the entire model using 8-bit symmetric quantization.

	FF	ATT	EMB	GEN	FF,EMB,GEN	EMB,GEN	FULL
16-bit Asym.	0.216 1.135x	0.011 1.098x	17.620 1.271x	17.641 1.086x	0.216 1.697x	17.639 1.413x	0 2x
16-bit Sym.	2.487 1.135x	0.025 1.098x	17.627 1.271x	17.633 1.086x	2.483 1.697x	17.617 1.413x	.005 2x
8-bit Asym.	17.425 1.217x	17.653 1.154x	17.615 1.47x	17.219 1.134x	17.3 2.61x	16.89 1.78x	**17.287** **4x**
8-bit Sym.	17.553 1.217x	17.613 1.154x	**17.747** **1.47x**	16.722 1.134x	16.657 2.61x	16.96 1.78x	16.853 4x
4-bit Asym.	10.839 1.26x	8.056 1.185x	17.296 1.595x	0.001 1.16x	**17.3** **3.557x**	.001 2.045x	0.035 8x
4-bit Sym.	6.422 1.26x	3.813 1.185x	16.849 1.595x	0.005 1.16x	0.008 3.557x	.004 2.045x	0.027 8x
3-bit Asym.	1.820 1.274x	0.780 1.193x	15.651 1.629x	0 1.167x	0 3.914x	0 2.125x	0 10.635x
3-bit Sym	1.26 1.274x	0.19 1.193x	15.339 1.629x	0 1.167x	0 3.914x	0 2.125x	0 10.635x

Table 4.2: IWSLT Post-Training Quantization Results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. F = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. We note slight BLEU score improvement when the embedding layers of the Transformer are compressed using 8-bit symmetric quantizaiton. We also achieve good compression rates when the feed-forward, embedding, and generator are compressed using 4-bit asymmetric quantization. We achieve 4x compression with a 2% accuracy loss using 8-bit asymmetric quantization.

From Table 4.1, we see that applying 8-bit post-training quantization did considerably better than other precisions in retaining model accuracy and achieving good compression rates for the Multi30k data set. We show that we can achieve 4x compression and lose only .4% in BLEU score for the Multi30k data set. We also show a slight increase in BLEU score over our baseline while achieving 2.115x compression. Finally, we note significant BLEU score drop as precision lowers when solely quantizing the attention and generator layers of the Transformer architecture.

This idea is also supported by Table 4.2, where we saw near total accuracy drop for the attention and generator layers as precision for quantization lowered. Unlike Table 4.1, we also saw significant BLEU score drop for the feed-forward layers as precision for quantization lowered, but found 8-bit post-training quantization to be the best for the layers. We suspect the attention and generator layers are sensitive to post-training quantization and that the feed-forward layers become more sensitive to quantization as model parameters increase. We also show that applying 8-bit, symmetric range-based linear post-training quantization slightly increased BLEU score over our baseline for our IWSLT experiment. Finally, we show that we can compress the model by 4x and incur a 2% accuracy loss in Table 4.2. Like in Table 4.1, we demonstrate that 8-bit post-training quantization retained model accuracy the best across all the layers of the Transformer while achieving considerable compression rates. We did not see a distinction in terms of performance between asymmetric and symmetric post-training quantization as BLEU scores varied for the two modes. We now consider automated gradual pruning for the Transformer network.

4.3 Experiment Two - Automated Gradual Pruning

We apply automated gradual pruning through Distiller to our implementation with an initial sparsity of 10% and a final sparsity of 70% and a pruning frequency of 1, an initial sparsity of 45% and a final sparsity of 70% with a pruning frequency of 3, and an initial sparsity of 90% and a final sparsity of 100% with a pruning frequency of 1. We use Distiller's YAML file support to configure the initial and final sparsity and pruning frequency of the compression scheme. To our knowledge, we are the first to apply automated gradual pruning to the Transformer architecture in evaluating how it affects BLEU score. For all experiments, we begin the pruning schedule at epoch 2 and end at epoch 20. We use the compressed Transformer model that performed the best on the validation set through 20 epochs. We use Distiller's tool for calculating model sparsity to collect our results. We present the results of this experiment in Tables 4.3 and 4.4:

Initial / Final / Freq	FF	ATT	EMB	GEN	FF,ATT	FULL
10% / 70% / 1	33.077 1.384x	34.743 1.262x	32.268 1.111x	36.171 1.052x	**36.292** **1.791x**	**35.915** **2.37x**
45% / 70% / 3	35.423 1.467x	35.348 1.363x	31.194 1.152x	35.969 1.06x	34.577 1.81x	34.208 3.312x
90% / 100% / 1	35.423 1.656x	25.198 1.414x	31.128 1.262x	21.743 1.082x	33.090 3.175x	9.741 28.094x

Table 4.3: The Multi30k Automated Gradual Pruning results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. The first column represents the initial sparsity, final sparsity, and pruning frequency levels. FF = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. We show that pruning the feed-forward and attention layers with an initial sparsity of 10% and final sparsity of 70% achieves a BLEU score over our baseline with a 1.791x compression rate. Similarly, we show we can use the same scheme and also achieve a BLEU score over our baseline for full model compression with a rate of 2.37x.

We saw varying results between the experiments with the Multi30k and IWSLT data sets. When automated gradual pruning was applied to the feed-forward layers for the Multi30k data set, we notice BLEU score improvement the more sparse the layers are. The opposite was the case when pruning the same layers for the IWSLT data set. Similar to post-training quantization, we also suspect the feed-forward layers are sensitive to automated gradual pruning as model parameters increase.

We were amazed to see the BLEU score increase over baseline for both data sets when automated gradual pruning was applied with a 45% initial sparsity, 70% final sparsity, and pruning frequency of 3, to the self-attention network. We achieved a BLEU score of 35.348 with a compression rate of 1.363x and a BLEU score of 25.315 with a compression rate of 1.114x for the Multi30k and IWSLT data sets respectively. We suspect that the self-attention

Initial / Final / Freq	FF	ATT	EMB	GEN	FF,ATT	FULL
10% / 70% / 1	17.637 0x	22.296 1.134x	11.327 1.328x	11.981 1.113x	23.728 1.341x	24.679 2.37x
45% / 70% / 3	11.518 1.195x	**25.315** **1.114x**	16.197 1.382x	12.003 1.077x	24.805 1.313x	**25.032** **2.338x**
90% / 100% / 1	4.783 1.311x	21.416 1.186x	20.996 1.743x	2.789 1.082x	24.293 1.616x	11.842 14.514x

Table 4.4: The IWSLT Automated Gradual Pruning results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. FF = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. Pruning the attention layers with a 45% and 70% initial and final sparsity with a pruning frequency of 3 achieved the largest BLEU score improvement. We also show that pruning the full architecture using the same scheme also shows large BLEU score improvement with a compression rate of 2.338x.

network is resistant to automated gradual pruning. We also note varying results when automated gradual pruning is applied to the embedding layers. We see a drop in BLEU score accuracy for both data sets as the layer became more sparse for the Multi30k, but see improvement for the IWSLT data set as the embedding layers become sparser.

For the Multi30k data set, we saw the biggest improvement in BLEU score when the feed-forward and self-attention layers are 63.74% sparse, but as sparsity increased, the BLEU score would drop. We were surprised to see that we can sparsify the feed-forward and self-attention layers by up to 91.66% and still see better results than our uncompressed baseline. This idea completely deviates from the sensitivity we saw with compressing these layers through post-training quantization.

Finally, we saw that we can compress the Transformer network using automated gradual pruning by a factor of 2.37x and see results better than our baseline for the Multi30k and IWSLT data sets. We were also able to sparsify the entire model by 69.8% and still retain 98.5% model accuracy for the Multi30k dataset. For the IWSLT data set, we show that we can compress the model by a factor of 2.338x and improve BLEU score performance that is over our baseline's.

4.4 Experiment Three - Quantization-Aware Training

Next, we apply asymmetric and symmetric range-based linear quantization-aware training to our implementation. Like for our automated gradual pruning experiment, we again use Distiller's YAML configuration to configure precision for our quantization experiments. Like our post-training quantization experiment, we consider 16, 8, 4, and 3 bits of precision for activations, weights, and biases. Again, we take the compressed models that performed the best on the validation set within 20 epochs. We report our results in Tables 4.5 and 4.6:

Our results show that we achieved the best BLEU score of 36.280 with a compression rate of 2.115x for the Multi30k data set. Also for this data set, we show that we can compress the Transformer by 8x and achieve a BLEU score over our baseline when we apply 4-bit symmetric, range-based linear quantization-aware training. Unlike our post-training quantization experiments for the Multi30k data set, we note fairly considerable retention across all the layers of the model as precision lowered. This idea is supported by [34], where quantization-aware training is argued to do better in retaining accuracy than post-training quantization because the quantized tensors are baked into the training procedure which forces the model to learn better through smaller tensors.

In contrast with the Multi30k data set, for the IWSLT data set, we show that compressing the generator to 3-bits using asymmetric unsigned quantization-aware training achieves the higher BLEU score over our baseline and the highest BLEU score of all our experiments. Similar to our Multi30k experiment for this strategy, we achieve a BLEU score better than baseline applying 4-bit range-based quantization-aware training, but for asymmetric unsigned. Like post-training quantization, we did notice a significant BLEU score drop for the feed-forward and attention layers, again suggesting that these layers are sensitive to quantization. Unlike post-training quantization, however, we saw improvement in BLEU score as we lowered the precision each time we quantized the generator layer.

	FF	ATT	EMB	GEN	FF,EMB,GEN	EMB,GEN	FULL
16-bit Asym.	34.7 1.22x	34.875 1.159x	34.268 1.118x	34.524 1.038x	33.641 1.482x	35.578 1.182x	34.006 1.860x
16-bit Sym.	35.761 1.22x	35.393 1.159x	33.976 1.118x	34.069 1.038x	34.849 1.482x	34.372 1.182x	31.799 1.860x
8-bit Asym.	32.987 1.421x	34.182 1.286x	33.797 1.207x	34.961 1.063x	**36.280** **2.115x**	32.885 1.3x	34.398 4x
8-bit Sym.	32.804 1.421x	32.574 1.286x	34.284 1.207x	34.517 1.063x	34.351 2.115x	33.932 1.3x	34.115 4x
4-bit Asym.	34.324 1.529x	35.135 1.351x	33.128 1.25x	33.457 1.074x	35.883 2.597x	31.459 1.368x	33.057 8x
4-bit Sym.	34.321 1.529x	35.582 1.351x	34.227 1.25x	34.894 1.074x	34.502 2.597x	33.238 1.368x	**35.445** **8x**
3-bit Asym.	34.324 1.588x	34.467 1.368x	34.874 1.261x	33.495 1.077x	34.213 2.754x	35.800 1.368x	33.335 10.614x
3-bit Sym.	33.705 1.588x	34.178 1.368x	32.740 1.261x	34.370 1.077x	35.077 2.754x	34.181 1.368x	34.101 10.614x

Table 4.5: The Multi30k quantization-aware training results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. FF = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. 8-bit asymmetric quantization to the feed-forward, embedding, and generator availed the best model accuracy for the Multi30k quantization-aware training experiment. We also show that we can compress the model by a factor of 8x using 4-bit symmetric quantization and still achieve a BLEU score over our baseline.

	FF	ATT	EMB	GEN	FF/EMB/GEN	EMB/GEN	FULL
16-bit Asym.	17.16	16.073	16.291	15.583	16.872	15.497	14.188
	1.135x	1.098x	1.271x	1.086x	1.697x	1.413x	2x
16-bit Sym.	10.092	14.427	15.719	16.763	15.918	12.494	16.612
	1.135x	1.098x	1.271x	1.086x	1.697x	1.413x	2x
8-bit Asym.	13.230	16.514	12.686	12.729	14.955	16.355	13.804
	1.217x	1.154x	1.47x	1.134x	2.61x	1.78x	4x
8-bit Sym.	15.266	14.706	14.871	15.021	16.486	12.495	16.486
	1.217x	1.154x	1.47x	1.134x	2.61x	1.78x	4x
4-bit Asym.	12.735	14.013	15.464	16.365	20.1	19.524	**19.636**
	1.26x	1.185x	1.595x	1.16x	3.557x	2.045x	**8x**
4-bit Sym.	11.518	12.306	17.072	18.198	**21.611**	21.275	15.847
	1.26x	1.185x	1.595x	1.16x	**3.557x**	2.045x	8x
3-bit Asym.	10.809	11.762	18.154	**22.752**	9.874	19.463	6.415
	1.274x	1.193x	1.629x	**1.167x**	3.914x	2.125x	10.635x
3-bit Sym.	10.987	11.044	17.153	21.979	10.713	14.356	9.478
	1.274x	1.193x	1.629x	1.167x	3.914x	2.125x	10.635x

Table 4.6: The IWSLT quantization-aware training results. For each cell, the BLEU score is displayed at the top and the model compression rate is displayed under it. FF = Feed-Forward Layers, ATT = Attention Layers, EMB = Embedding Layers, GEN = Generator Layer, and FULL = Total Model Compression. Applying 3-bit asymmetric quantization to the generator layer gave the best BLEU score improvement for the IWSLT dataset. We also saw high BLEU score improvement when applying 4-bit symmetric quantization to the feed-forward, embedding, and generator layers of the Transformer while achieving a 3.557x compression rate. We show that we can compress the entire model and still achieve a BLEU score higher than our baseline using 4-bit asymmetric quantization and achieve an 8x compression rate.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we considered compressing the novel Transformer architecture through post-training quantization, automated gradual pruning, and quantization-aware training strategies. We considered tuning different compression parameters and analyzed how they affected performance of the overall architecture. We saw varying results from our ablation studies when compressing individual and multiple layers of the Transformer network which we believe can ultimately be leveraged in achieving higher accuracy and better compression rates. Our results show that automated gradual pruning and quantization-aware training can improve BLEU score when compressing certain layers and compressing the entire model.

In the future, we wish to see how certain compression schemes work in tandem in compressing the Transformer. Specifically, we would like to answer the question: Can pruning and quantization work well together when compressing the Transformer architecture [15]? Because of constraints with experimentation time, we would have also liked to perform these studies on the WMT14 English to German translation task [5] as was done in [27,30]. Additionally, we would like to explore how certain design changes can be made to neuromorphic accelerators in efficiently processing DNNs such as the Transformer network.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] J. Alammar, *The illustrated transformer*, 2018.

[3] J. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, ArXiv, abs/1607.06450 (2016).

[4] Y. Bengio, N. Léonard, and A. C. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, ArXiv, abs/1308.3432 (2013).

[5] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. s. Tamchyna, *Findings of the 2014 workshop on statistical machine translation*, in Proceedings of the Ninth Workshop on Statistical Machine Translation, Baltimore, Maryland, USA, June 2014, Association for Computational Linguistics, pp. 12–58.

[6] M. Cettolo, C. Girardi, and M. Federico, *Wit^3: Web inventory of transcribed and translated talks*, in Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT), Trento, Italy, May 2012, pp. 261–268.

[7] R. Cheong and R. Daniel, *transformers.zip: Compressing transformer with pruning and quantization*, tech. rep., Stanford University, 2019.

[8] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, *Pact: Parameterized clipping activation for quantized neural networks*, ArXiv, abs/1805.06085 (2018).

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, in NAACL-HLT, 2019.

[10] D. Elliott, S. Frank, K. Sima'an, and L. Specia, *Multi30K: Multilingual English-German image descriptions*, in Proceedings of the 5th Workshop on Vision and Language, Berlin, Germany, Aug. 2016, Association for Computational Linguistics, pp. 70–74.

[11] C. Fan, *Quantized transformer*, tech. rep., Stanford University, 2019.

[12] A. Galassi, M. Lippi, and P. Torroni, *Attention in natural language processing.*, arXiv: Computation and Language, (2020).

[13] T. Gale, E. Elsen, and S. Hooker, *The state of sparsity in deep neural networks*, ArXiv, abs/1902.09574 (2019).

[14] Y. Guo, A. Yao, and Y. Chen, *Dynamic network surgery for efficient dnns*, in NIPS, 2016.

[15] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*, CoRR, abs/1510.00149 (2016).

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, J. Mach. Learn. Res., 18 (2017), pp. 187:1–187:30.

[17] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, *Opennmt: Open-source toolkit for neural machine translation*, in Proc. ACL, 2017.

[18] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal brain damage*, in Advances in Neural Information Processing Systems 2, D. S. Touretzky, ed., Morgan-Kaufmann, 1990, pp. 598–605.

[19] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, *On-device neural net inference with mobile gpus*, ArXiv, abs/1907.01989 (2019).

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient convnets*, ArXiv, abs/1608.08710 (2017).

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, ArXiv, abs/1907.11692 (2019).

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, arXiv e-prints, (2013), p. arXiv:1301.3781.

[23] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, *Wrpn: Wide reduced-precision networks*, ArXiv, abs/1709.01134 (2018).

[24] S. Narang, G. Diamos, S. Sengupta, and E. Elsen, *Exploring sparsity in recurrent neural networks*, ArXiv, abs/1704.05119 (2017).

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in pytorch*, in NIPS 2017 Workshop on Autodiff, 2017.

[26] M. Post, *A call for clarity in reporting bleu scores*, ArXiv, abs/1804.08771 (2018).

[27] G. Prato, E. Charlaix, and M. Rezagholizadeh, *Fully quantized transformer for improved translation*, ArXiv, abs/1910.10485 (2019).

[28] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient processing of deep neural networks: A tutorial and survey*, Proceedings of the IEEE, 105 (2017), pp. 2295–2329.

[29] A. P. Tierno, *Quantized transformer*, tech. rep., Stanford University, 2019.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, ArXiv, abs/1706.03762 (2017).

[31] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, *Ernie: Enhanced language representation with informative entities*, ArXiv, abs/1905.07129 (2019).

[32] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients*, ArXiv, abs/1606.06160 (2016).

[33] M. Zhu and S. Gupta, *To prune, or not to prune: exploring the efficacy of pruning for model compression*, ArXiv, abs/1710.01878 (2018).

[34] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, *Neural network distiller: A python package for dnn compression research*, ArXiv, abs/1910.12232 (2019).