Fast Fourier Transformed Transformers: Circulant Weight Matrices for NMT Compression

Scott Reid
shrcor@stanford.edu

Matthew Mistele
mmistele@stanford.edu

Abstract

Structured weight matrices have been the subject of recent interest due to their potential to compress and accelerate models [1]. Block-circulant matrices are a particularly promising type that has yet to be applied in major NLP publications or compared to traditional pruning methods: they interpolate between highly compressed "circulant" matrices ($O(n)$ parameters and forward-pass runtime of $O(n \log n)$) and traditional unstructured matrices (which have $O(n^2)$ parameters and $O(n^2)$ runtime). We created a block-circulant layer class for PyTorch from scratch with optimized methods we derived for computing the forward pass and gradients. Our two-layer model using this class achieved a test error of 5% on MNIST with 200x compression; by comparison, 200x compression with the conventional pruning approach had a test error of 60%. We then replaced the feedforward layers of the Transformer architecture [3] with block-circulant layers, and the model with largest block size achieved BLEU score of 20.5 on Multi30k Task 1 German-to-English with 100x compression. In comparison, pruning achieved a BLEU score of 28 at 100x compression, and the original achieved 37.4 with no compression.

1 Introduction

Structured weight matrices, such as circulant matrices, have been the subject of recent interest due to their potential to compress and accelerate models [1]. In addition, pruning techniques, wherein sparse weight matrices are learned, have achieved very good success in compressing models in many domains in neural networks [2]. Model compression has typically been used for the deployment of neural networks on resource-constrained devices such as mobile phones and IoT devices. However, with the rapidly accelerating size of deep neural networks within the domain of Natural Language Processing, model compression can be very impactful in reducing computational loads and energy usage in cloud-deployed neural networks. Transformer networks [3] are composed of cascaded chains of positionwise fully-connected feedforward layers and multi-head attention layers with sublayer connections. They have been the subject of much recent excitement for their use in pre-trained networks for context-dependent word encodings [4]. The large scale of these pre-trained Transformer networks makes them computationally expensive and slow to deploy. Thus, we identified the Transformer architecture as a prime candidate within the domain of Natural Language Process to experiment with different types of model compression.

In this project, we identified two deficiencies in the existing literature on compressed neural networks. First of all, existing literature lacks a direct comparison of pruning and structured-weight matrices for model compression. Secondly, we noticed the absence of applications of structured weight matrices into the field of natural language processing. In this project, we directly tackle both questions by comparing the efficacy of both pruning and structured weight matrix techniques on the machine translation task with Transformer networks.

2 Approach

Block-circulant weight matrices have been successfully applied to drastically compress image recognition networks, but no studies to date have applied them to problems in NLP. In addition, no study to date has compared block-circulant model compression to more standard pruning-based model compression. In this project, we compare traditional pruning methods with block-circulant matrices in compressing the Transformer architecture.

In a conventional fully-connected linear layer, the output is given by

$$h(\mathbf{x}) = \phi(\mathbf{R}\mathbf{x})$$

where $\phi(x)$ is a nonlinear activation function and $\mathbf{R} \in \mathbb{R}^{d \times k}$ is an unstructured weight matrix.

Our technique is to replace linear layers with block-circulant layers that instead compute

$$h(\mathbf{x}) = \phi(\mathbf{B}\mathbf{D}\mathbf{x}) \tag{1}$$

where $\mathbf{B}$ is a block-circulant matrix and $\mathbf{D}$ is a fixed diagonal Bernouli matrix introduced to make the projections less correlated. More concretely, $\mathbf{D}$ is a diagonal matrix with fixed diagonal entries that are equally likely to be 1 or -1.

The technique of replacing layers with circulant layers was found to be successful by Cheng et al (2015) [1]. To the best of our knowledge, there have yet to be papers published on applying circulant layers to natural language processing tasks, much less block-circulant layers (a generalization) as we are doing. Meanwhile, there are groups around the world working on ASIC hardware to accelerate FFT-based block-circulant calculations.

2.1 Circulant Weight Matrices

Structured weight matrices allow $\mathbb{R}^{d \times k}$ weight matrices to be defined in fewer than $d \cdot k$ parameters by reusing each parameter in multiple locations within the matrix. A circulant weight matrix $\mathbf{C} \in \mathbb{R}^{d \times d}$ is defined by its weight vector $\mathbf{w} \in \mathbb{R}^d$. The first row of $\mathbf{C}$ is given by $\mathbf{w}$ and each subsequent row is rotated one element to the right.

$$\mathbf{C} = \texttt{circ}(\mathbf{w}) = \begin{bmatrix} w_1 & w_2 & \dots & w_d \\ w_d & w_1 & \dots & w_{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_2 & w_3 & \dots & w_1 \end{bmatrix} \tag{2}$$

Matrix-vector multiplication with circulant matrices can be accelerated by making use of the discrete fourier transform. In diagonalized form, a circulant matrix can be written as:

$$\mathbf{C} = \texttt{circ}(\mathbf{w}) = \mathbf{F}\ \texttt{diag}(\mathbf{F}\mathbf{w})\ \mathbf{F}^\dagger \tag{3}$$

Here, the eigenvector matrix $\mathbf{F} \in \mathbb{R}^{d \times d}$ is the d-dimensional Discrete Fourier Transform (DFT) matrix and its conjugate transpose $\mathbf{F}^\dagger$ represents the Inverse Discrete Fourier Transform matrix. $\mathbf{C}$'s eigenvalues are given by the DFT of its weight vector $\mathbf{w}$.

In this diagonal form, matrix-vector multiplications can be accelerated by making use of the Fast Fourier Transform (FFT) algorithm. This allows matrix-vector products to be computed in $O(d \log d)$ time rather than $O(d^2)$ time.

$$\begin{aligned} \mathbf{C}\mathbf{x} &= \texttt{circ}(\mathbf{w})\mathbf{x} && (4) \\ &= \mathbf{F}\big(\texttt{diag}(\mathbf{F}\mathbf{w})\ (\mathbf{F}^\dagger\mathbf{x})\big) && (5) \\ &= \texttt{FFT}\big(\texttt{FFT}(\mathbf{w}) \circ \texttt{IFFT}(\mathbf{x})\big) && (6) \\ &= \texttt{IRFFT}\big(\texttt{RFFT}(\mathbf{w})^* \circ \texttt{RFFT}(\mathbf{x})\big) && (7) \end{aligned}$$

Here, $\circ$ represents an element-wise product and $\mathbf{y}^*$ represents $\mathbf{y}$'s complex conjugate. The Fast Fourier Transform `FFT` goes through redundant calculations when its input is real-valued (elements form complex-conjugate pairs). Therefore, we reduce computation by using the Real Fast Fourier Transform (`RFFT`) and its inverse (`IRFFT`).

Circulant weight matrices can replace unstructured weight matrices in linear layers of a neural network. A circulant layer's activation is $\mathbf{a} = \texttt{circ}(\mathbf{w})\mathbf{x}$ for input $\mathbf{x}$ and weight vector $\mathbf{w}$. Letting $\mathcal{L}$ be the loss, $\dfrac{\partial \mathcal{L}}{\partial \mathbf{a}}$ is its derivative with respect to the circulant layer's activation. To backpropogate gradients and update the weight vector $\mathbf{w}$, we must compute $\dfrac{\partial \mathcal{L}}{\partial \mathbf{w}}$ and $\dfrac{\partial \mathcal{L}}{\partial \mathbf{x}}$. A beautiful property of circulant weight matrices is that these gradient calculations can also be calculated as circulant products.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \texttt{circ}\Big(\frac{\partial \mathcal{L}}{\partial \mathbf{a}}\Big)\mathbf{x} = \texttt{IRFFT}\left(\texttt{RFFT}\Big(\frac{\partial \mathcal{L}}{\partial \mathbf{a}}\Big)^* \circ \texttt{RFFT}(\mathbf{x})\right) \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \texttt{circ}\Big(\frac{\partial \mathcal{L}}{\partial \mathbf{a}}\Big)\mathbf{w} = \texttt{IRFFT}\left(\texttt{RFFT}\Big(\frac{\partial \mathcal{L}}{\partial \mathbf{a}}\Big)^* \circ \texttt{RFFT}(\mathbf{w})\right) \tag{9}$$

The basic computational unit of circulant layers for inference and learning is the $O(d \log d)$ operation

$$\texttt{IRFFT}\big(\texttt{RFFT}(\ldots)^* \circ \texttt{RFFT}(\ldots)\big).$$

This contrasts with unstructured linear layers, where $O(d^2)$ matrix multiplications are the basic computational unit.

Circulant layers can be extended to the case where $d_{\text{in}} \neq d_{\text{out}}$. If $d_{\text{in}} < d_{out}$, we pad $\mathbf{x}$ to d_{out}. Otherwise, if $d_{\text{out}} < d_{in}$, we slice the first d_{out} elements of $\mathbf{a}$.

2.2 Block-Circulant Weight Matrices

Block-circulant layers interpolate between circulant layers and unstructured linear layers in terms of both compression and computational efficiency. A single block-circulant weight matrix contains multiple circulant submatrices. The number of circulant submatrices is a function of the input dimensionality d_{in}, output dimensionality d_{out}, and circulant block size b. We define $k_{\text{in}} = \lceil d_{\text{in}}/b \rceil$ and $k_{\text{out}} = \lceil d_{\text{out}}/b \rceil$. Our block-circulant weight matrix $\mathbf{B}$ is composed of a set of circulant weight matrices $\Big\{\{\mathbf{C}_{i,j} = \texttt{circ}(\mathbf{W}_{i,j})\}_{i=1}^{k_{\text{out}}}\Big\}_{j=1}^{k_{\text{in}}}$.

$$\mathbf{B} = \begin{bmatrix} \texttt{circ}(\mathbf{W}_{1,1}) & \texttt{circ}(\mathbf{W}_{1,2}) & \ldots & \texttt{circ}(\mathbf{W}_{1,k_{\text{in}}}) \\ \texttt{circ}(\mathbf{W}_{2,1}) & \texttt{circ}(\mathbf{W}_{2,2}) & \ldots & \texttt{circ}(\mathbf{W}_{2,k_{\text{in}}}) \\ \vdots & \vdots & \ddots & \vdots \\ \texttt{circ}(\mathbf{W}_{k_{\text{out}},1}) & \texttt{circ}(\mathbf{W}_{k_{\text{out}},2}) & \ldots & \texttt{circ}(\mathbf{W}_{k_{\text{out}},k_{\text{in}}}) \end{bmatrix} \tag{10}$$

Weights are stored in a 3-tensor $\mathbf{W} \in \mathbb{R}^{k_{\text{out}} \times k_{\text{in}} \times b}$ such that $\mathbf{W}_{i,j} \in \mathbb{R}^b$ is the weight vector for circulant block $\mathbf{C}_{i,j}$. A block-circulant layer has $k_{\text{out}} k_{\text{in}} b \geq \dfrac{d_{\text{out}} d_{\text{in}}}{b}$ parameters, where equality holds when b divides d_{in} and d_{out}. Comparing this to an unstructured weight layer with $d_{\text{out}} d_{\text{in}}$ parameters, we can interpret the block-size b as a compression factor.

Block-circulant products are calculated as sums over $k_{\text{in}} \times k_{\text{out}}$ circulant products, which are computed in parallel. We first reshape our input vector $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ into a matrix $\mathbf{X} \in \mathbb{R}^{k_{\text{in}} \times b}$ by slicing: $\{\mathbf{X}_j = \mathbf{x}[b \cdot (j-1) : b \cdot j]\}_{j=1}^{k_{\text{in}}}$. If $k_{\text{in}} \cdot b > d_{\text{in}}$, we first pad $\mathbf{x}$ with zeros. Then,

$$\mathbf{B}\mathbf{x} = \begin{bmatrix} \sum_{j=1}^{k_{\text{in}}} \texttt{circ}(\mathbf{W}_{1,j})\mathbf{X}_j \\ \sum_{j=1}^{k_{\text{in}}} \texttt{circ}(\mathbf{W}_{2,j})\mathbf{X}_j \\ \vdots \\ \sum_{j=1}^{k_{\text{in}}} \texttt{circ}(\mathbf{W}_{k_{\text{out}},j})\mathbf{X}_j \end{bmatrix} = \begin{bmatrix} \texttt{IRFFT}\Big(\sum_{j=1}^{k_{\text{in}}} \texttt{RFFT}(\mathbf{W}_{1,j})^* \circ \texttt{RFFT}(\mathbf{X}_j)\Big) \\ \texttt{IRFFT}\Big(\sum_{i=j}^{k_{\text{in}}} \texttt{RFFT}(\mathbf{W}_{2,j})^* \circ \texttt{RFFT}(\mathbf{X}_j)\Big) \\ \vdots \\ \texttt{IRFFT}\Big(\sum_{i=j}^{k_{\text{in}}} \texttt{RFFT}(\mathbf{W}_{k_{\text{out}},j})^* \circ \texttt{RFFT}(\mathbf{X}_j)\Big) \end{bmatrix} \tag{11}$$

This yields an activation vector $\mathbf{a} \in \mathbb{R}^{k_{\text{out}} \cdot b}$, which can be sliced to the output dimensionality d_{out} if needed. In the above expression, the IRFFT operations are pulled in front of the sums. This is possible because IRFFT is a linear operator, and it reduces the required computation.

Block-circulant products involve $k_{\text{out}} \cdot k_{\text{in}}$ circulant products, each of size b. Thus, the computational complexity is $O(k_{\text{in}} k_{\text{out}} b \log b) = O\Big(\frac{d_{\text{out}} d_{\text{in}}}{b} \log(b)\Big)$. In the limit where $b = 1$, we obtain $O(d_{\text{out}} d_{\text{in}})$ matrix multiplication.

In addition, backpropogation of gradients through block-circulant layers can be accelerated with the FFT. Given $\frac{\partial \mathcal{L}}{\partial \mathbf{a}} \in \mathbb{R}^{d_{\text{out}}}$, we pad and reshape the gradients into the matrix $\mathcal{L}' \in \mathbb{R}^{k_{\text{in}} \times b}$. Then:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \begin{bmatrix} \texttt{IRFFT}\Big(\sum_{i=1}^{k_{\text{out}}} \texttt{RFFT}(\mathcal{L}'_i)^* \circ \texttt{RFFT}(\mathbf{W}_{i,1})\Big) \\ \texttt{IRFFT}\Big(\sum_{i=1}^{k_{\text{out}}} \texttt{RFFT}(\mathcal{L}'_i)^* \circ \texttt{RFFT}(\mathbf{W}_{i,2})\Big) \\ \vdots \\ \texttt{IRFFT}\Big(\sum_{i=1}^{k_{\text{out}}} \texttt{RFFT}(\mathcal{L}'_i)^* \circ \texttt{RFFT}(\mathbf{W}_{i,k_{\text{in}}})\Big) \end{bmatrix} \text{ sliced to the first } d_{\text{in}} \text{ elements} \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{i,j}} = \texttt{IRFFT}\Big(\texttt{RFFT}(\mathcal{L}'_i)^* \circ \texttt{RFFT}(\mathbf{X}_j)\Big) \quad \text{for } i = 1, \ldots, k_{\text{out}};\ j = 1, \ldots k_{\text{in}} \tag{13}$$

In the limit where $k_{\text{in}} = k_{\text{out}} = 1$ (i.e. $b \geq \max\{d_{\text{in}}, d_{\text{out}}\}$), block-circulant layers reduce to circulant layers. In the opposite limit where $b = 1$, block-circulant layers reduce to unstructured linear layers.

2.3 Pruned Linear Layers

Following the approach of Han et. al. [2], we prune linear layers by first training a normal linear layer, and then entering a cycle of incremental pruning and fine-tuning. After the initial training, we prune by permanently fixing the smallest magnitude weights to zero. We then train further to fine-tune the nonzero weights that remain after pruning. The process of pruning and fine-tuning can be iterated repeatedly.

3 MNIST Benchmark

As a benchmarking experiment, we compared block-circulant compressed models and pruned linear models on the MNIST dataset. This tested the efficacy of our custom PyTorch implemented Block-Circulant and Pruneable Linear layers.

3.1 Data

Our first experiment was conducted on MNIST, a dataset consisting of 60k training images and 10k test images of 28x28 pixel handwritten digits (in black and white).

3.2 Evaluation method

We evaluate performance according to error on the test set: the percentage of times the model output a digit that was not correct (i.e. the handwritten digit in the image).

3.3 Experimental details

We explored a fixed 2-layer fully-connected architecture for the MNIST benchmarking experiment. Our architecture consisted of a single hidden layer that projects the 784 pixels of MNIST digits to an 800 dimensions, followed by an output layer that projects down to 10 dimensions for the 10 digit classes.

In our block-circulant models, block-circulant layers of different block sizes were used for the linear projections. Models with larger block sizes had fewer total parameters than those with small block sizes – thus we were able to explore the effect of compression on test error accuracy after 50 epochs of training.

We also developed a pruned linear model. This model had unstructured linear projection matrices. Following the procedure of Han et. al. [2], after training for 50 epochs, we progressively pruned the model by fixing 1% of weights to zero based on magnitudes, then retrained the network for a single epoch of fine-tuning.

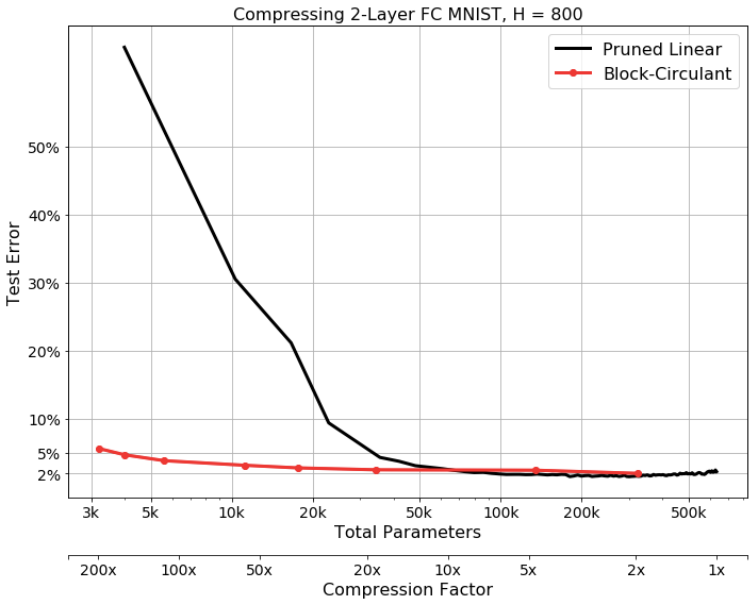

Figure 1: Block-Circulant Network outperforms Pruned Linear Network on MNIST as compression factor increases

3.4 Results

As shown in Figure 1, the architecture compressed using the conventional pruning approach quickly deteriorated in performance as compression factor surpassed 20x, with a test error of about 30% at 50x compression and of over 60% at 200x compression. By contrast, the architecture compressed by using block-circulant weight matrices was able to achieve approximately the same performance at 200x compression as the baseline achieved at 20x compression. These results are better than expected and strongly compelling, particularly since the Pruned Linear models had more training time: 50 epochs at first followed by several iterations of fine-tuning.[1] These results are promising for the prospects of applying block-circulant matrices to compress attention layers in the Transformer architecture (in progress).

In particular, the block-circulant model with $b_{\text{base}} = 128$ had a BLEU score of 23.5, the one with $b_{\text{base}} = 256$ had a score of 21.3, and the one with $b_{\text{base}} = 512$ scored 20.5. The block-circulant transformer with the circulant generator did not have any 4-gram matches in the test set, so it got a BLEU score of 0.

4 The Block-Circulant Transformer

A single encoder or decoder layer within the Transformer network architecture consists of one or two multihead attention units, a positionwise feedforward unit, and sub-layer connections with layer normalization. Of these parts, the positionwise feedforward and multihead attention units involve matrix multiplications by learned weight matrices. In the original work on Transformers [3], these weight matrices were unstructured and uncompressed. In this work, we compare the effects of using pruning linear matrices and using block-circulant weight matrices to compress these weight matrices.

4.1 Block-Circulant Multihead Attention Unit

Our multi-head attention unit takes two input sequences: a query sequence $\mathbf{S}_Q \in \mathbb{R}^{N_Q \times d_{\text{model}}}$ and a key/value sequence $\mathbf{S}_K \in \mathbb{R}^{N_K \times d_{\text{model}}}$. Letting h be the number of attention heads (default 8), we

[1] Given the simplicity of the models (only two layers), the 60% test error is almost entirely attributable to avoidable bias, not overfitting due to too much training.

produce queries, keys, and values in $\mathbb{R}^d_k$ where $d_k = d_{\text{model}}/h$ by projecting the query sequence and key/value sequence by block-circulant matrices.

We could learn $3h$ seperate block-circulant weight matrices $\mathbf{B}^{(i)}_\alpha \in \mathbb{R}^{d_k \times d_{\text{model}}}$, for $i = 1, \ldots, h$ and $\alpha = K, Q, V$. Instead, we learn a 3 block-circulant weight matrices $\mathbf{B}_\alpha \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ for $\alpha = K, V, Q$ and slice the resulting products h times. It is important that these weight matrices $\mathbf{B}_\alpha$ have block-sizes b_{attn} smaller than d_k to ensure that different attention heads are not correlated.

After taking these projections, we compute dot product attention for each attention head:

$$\texttt{attention}(\mathbf{Q}^{(i)}, \mathbf{K}^{(i)}, \mathbf{V}^{(i)}) = \texttt{softmax}\left(\frac{\mathbf{Q}^{(i)}\mathbf{K}^{(i)\top}}{\sqrt{d_{\text{model}}/h}}\right)\mathbf{V}^{(i)} \tag{14}$$

After computing dot-product attention, the output sequences are concatenated into a sequence of shape (N_K, d_{model}) and projected through a block-circulant projection matrix $\mathbf{B}_{\text{proj}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ with block-size b_{attn}.

Thus, each block-circulant multihead attention unit learns 4 block-circulant weight matrices $(\mathbf{B}_Q, \mathbf{B}_K, \mathbf{B}_V, \mathbf{B}_{\text{proj}})$ of shape $d_{\text{model}} \times d_{\text{model}}$ and block-size $b_{\text{attn}} < d_{\text{model}}/h$.

4.2 Block-Circulant Positionwise Feedforward Unit

Our block-circulant positionwise feedforward unit makes use of two block-circulant weight matrices. It applies the transformation:

$$\mathbf{a} = \mathbf{B}_2\ \texttt{ReLU}\big(\mathbf{B}_1\mathbf{x}\big) \tag{15}$$

Here, $\mathbf{B}_1 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ has block-size b_1 and $\mathbf{B}_2 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ has block-size b_2. By default, $d_{\text{ff}} = 4 \cdot d_{\text{model}}$.

4.3 Block Sizes

The block-size parameters b_{attn}, b_1, and b_2 introduce a 3-dimensional hyperparameter space. However, to iterate quickly and test models, we defined a single block-size b_{model}. Relative to b_{model},

$$b_1 = b_{\text{model}} \tag{16}$$
$$b_2 = 4 \cdot b_{\text{model}} \tag{17}$$
$$b_{\text{attn}} = b_{\text{model}}/h \tag{18}$$

Here, the maximum value of b_{model} is d_{model}, at which point it is fully circulant.

5 Experiments

5.1 Data

Our first and primary NMT experiment was conducted on Multi30k Task 1 German-to-English, a task on translating descriptions of images from German to English. The dataset consists of a parallel corpus of 30k descriptions of images in English and in German. It has 29000 training examples (English-German sentence pairs), and approximately 1000 examples each for the validation and test sets. The source (German) vocabulary size in the train set is 8009, and the target (English) vocabulary size in the train set is 6191. This dataset was attractive to us for benchmarking because its small size allowed for rapid prototyping. As we learned in our IWSLT pilot experiment (Section 5.5), we did not have access to the requisite computing power or training time to train for larger datasets.

Two representative examples from the English part of the corpus (sampled at random) are the following:

- "Two men, one black and one white, play their guitars and sing into microphones as they stand outdoors."
- "A man bundled up in the cold weather up in a bucket working on a power line."

5.2 Evaluation

We evaluated performance using corpus BLEU score on a held-out set of 1014 examples, specifically using `nltk.translate.bleu_score.corpus_bleu`. For each pair of German and English sentences in the evaluation set, the corpus BLEU score used the English sentence as its single reference and the top English sentence output of beam search as the hypothesis. (The corpus BLEU was calculated on the aggregation of references and hypotheses over all the pairs.)

5.3 Experimental Details

We used the same base model and optimizer configurations as Vaswani et al.'s base Transformer [3]. Our implementation used the code from OpenNMT's "Annotated Transformer"[2] as a starting point. Each model had 6 layers with 8 attention heads, hidden dimension of 512, and feedforward dimension of 2048. We used a batch size of 1024 instead of Vaswani et al.'s 12000 in order to not run out of memory.

Like Vaswani et al., we used an Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$ and $\epsilon = 10^{-9}$. The learning rate was varied over the training, using the formula $d_{\texttt{model}}^{-0.5} \cdot \min(\texttt{step\_num}^{-0.5}, \texttt{step\_num} \cdot \texttt{warmup\_steps}^{-1.5}$ to increase learning rate linearly for 4000 warmup steps, then decrease it proportionally to the inverse square root of the step number. We also used label smoothing of value $e_{ls} = 0.1$; Vaswani et al. notes that this hurts perplexity since the model learns to be more unsure, but improves accuracy and BLEU score. [3]

We used KL Divergence Loss for training and validation, with Label Smoothing and no size averaging, between the softmax output of the decoder and the one-hot vector of the correct English token.

For inference, we used beam search[4] with beam size 8, and length penalty parameter $\alpha_{\text{len}} = 0.6$. Like Vaswani et al., we penalized short outputs by dividing the sum of the log probabilities of each hypothesis by the length penalty $\left(\frac{5+L}{6}\right)^{\alpha_{\text{len}}}$.

We trained 4 primary models: the standard transformer (which we call "linear"), and the block-circulant version with "model block sizes" b_{model} of 128, 256, and 512. We also trained a "special" block-circulant transformer with $b_{\text{model}} = 512$ and $b_{\text{attn}} = 512$. This "special" block-circulant transformer allowed us to explore the effects of sharing weight parameters between attention heads.

Each model was trained for 20 epochs, chosen because the standard linear transformer model's validation loss started saturated soon thereafter (on the 22nd epoch).

For the baseline linear transformer, we iteratively pruned and fine-tuned using the process described in the "1.3. Pruned Linear Layers" section. On each pruning iteration, we pruned 21% of remaining the non-zero parameters and fine-tuned for 1 epoch.

5.4 Results

In Fig. 2, we compare the BLEU score performance of our compressed models. For the same number of parameters, Pruned Linear Transformers achieved 8 points higher BLEU scores than Block-Circulant Transformers. As model compression increased, performance tended to decrease. One notable exception to this trend was that performance increased by 4 BLEU when we pruned 20% of the Transformer parameters. This increase in performance was likely due to pruning's regularizing effect. We also observed the importance of using different parameters for each attention head. In our most aggressively compressed Block-Circulant Transformer, we set attention block-sizes to 512, thus sharing weight parameters between attention heads. This resulted in a 5 BLEU drop in performance.

[2]The Annnotated Transformer, link: `http://nlp.seas.harvard.edu/2018/04/03/attention.html`

[3]We used OpenNMT's implementation of Label Smoothing and the "NoamOptimizer" that performs the "warmup step" optimization scheme above.

[4]We implemented beam search ourselves using OpenNMT's Beam class and the assignment 4 code as reference implementations. Our contributions include detailed logging and tensor processing to satisfy the input expectations of the decoder.

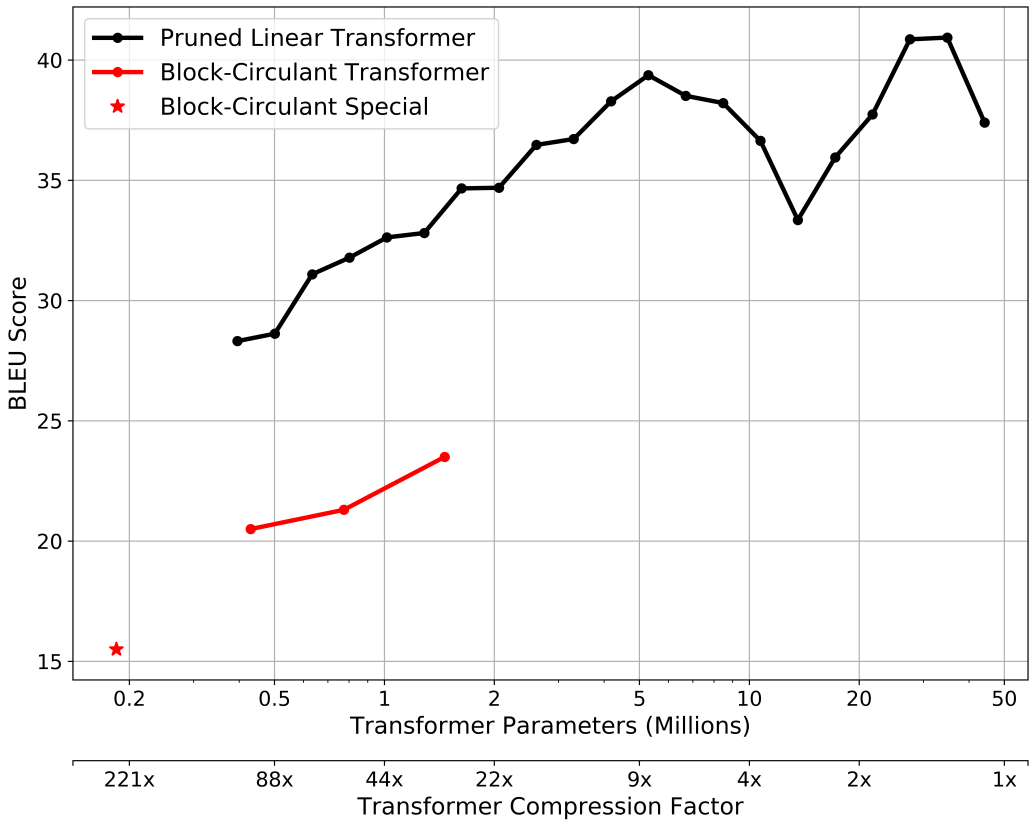

Figure 2: **Comparison of Compressed NMT Transformers on Multi30k De - En Translation** Counts of Transformer Parameters do not include 2,556,300 word embeddings parameters and 3,169,792 generator parameters because we did not attempt to compress these parts of the model. Both Pruned Linear (*black*) and Block-Circulant models (*red*) suffered lower BLEU scores as the number of parameters decreased, with the notable exception of a slight increase in BLEU score for slight pruning, likely due to the regularizing effects of pruning. For the same number of parameters, BLEU scores tended to be 8 points higher for pruned linear models than block-circulant models. When the attention heads shared weight parameters (*red star*), BLEU scores suffered significantly.

5.5 IWSLT Pilot Experiment

We also ran a pilot experiment on the IWSLT 2016 TED talk translation task, again translating from German to English. This dataset is much larger and contains more syntactically diverse sentences than Multi30k. It is comparable in size and character to the WMT 2014 English-German dataset, on which the transformer base model used by Vaswani et al. (2017) [3] took 12 hours (100,000 steps of size 12000) to train on a machine with 8 NVIDIA P100 GPUs. Azure did not let us use P100 GPUs, but rather M60s on NV6s, which are about half as powerful. Our setup used one GPU at a time, so it was $2 * 8 = 16$ times slower, meaning it would take 192 hours to replicate for a single model. Our block-circulant models do not run as fast on the GPUs as the linear models, so it would have taken well over this many hours to replicate with block-circulant layers. But we were able to get results with linear up against our best block-circulant model setup, each trained for about 80,000 steps of size 1024 (about $1/14$ the steps of Vaswani et al. [3]), with linear getting a BLEU score of 3 and block-circulant with a BLEU score of approximately 2.

6 Analysis

Our experiments with model compression on the Multi30k Task 1 German-to-English translation task demonstrate that pruning is much more effective than block-circulant structured weight matrices

at compressing the Transformer network. Pruned linear transformers consistently performed with 8 BLEU higher than block-circulant transformers for the same number of parameters. One explanation for this comes from the basic mathematical definition of circulant products. Circulant matrix-vector products can be mapped to 1D circular convolutions. Circular convolutions over the elements of word-embeddings simply may not be well-suited to the task of modeling language.

More understanding of the performance of block-circulant transformers can be gleaned by studying specific examples of translated text. Interestingly, both the block-circulant models and the pruned linear models almost never included articles at the beginning of the sentence, even if the target sentence did. For example, the most compressed block-circulant model (with a BLEU score of 20.5) translated "A young skier looks at the camera." to "young skier looking into the camera." The pruned linear models were better able to translate verb tense and exact word matches - for example, the 99% pruned model translated it to "young skier looks at the camera." This could have to do with the linear model having the expressiveness to learn a lookup table-like representation for words and tenses, which could perform well given the large number of parameters relative to the vocabulary size; there may have been enough parameters even after pruning that the highest-value weights held onto these word representations.

The outputs were also not always syntactically correct, and would sometimes repeatedly output a token while omitting others. For example, the German sentence with reference English sentence "Girls in black bikinis playing beach volleyball." was translated to "in black bikinis in a black and black bikinis." This could have to do with the circulant property bringing attention up later.

7 Conclusion

Our project found that compression via block-circulant layers outperformed compression via pruning on the MNIST computer vision benchmark, with the discrepancy increasing dramatically as a function of compression level. However, in the domain of transformers for NMT, pruning-based compression outperformed the block-circulant approach, and the discrepancy was comparable across compression levels.

In the future, we believe it would be worthwhile to investigate methods of weight initialization that could work better for block-circulant matrices than Xavier initialization, since Xavier initialization assumes the matrix is unstructured. This matter is particularly important since block-circulant matrices allow architectures to "go deeper" than before with the same number of parameters, and adding more layers in this way makes the problems of vanishing and exploding gradients more salient. We also believe it would be worthwhile to run a full experiment with IWSLT or WMT 2014 in place of Multi30k, to see if block-circulant compression would perform better relative to pruning on larger datasets with more complex and diverse syntax and vocabulary.

8 Additional Information

Our mentor is Michael Hahn.

References

[1] Cheng, Yu & Yu, Felix & S. Feris, Rogerio & Kumar, Sanjiv & Choudhary, Alok & Chang, Shi-Fu. (2015). An Exploration of Parameter Redundancy in Deep Networks with Circulant Projections. 2857-2865. 10.1109/ICCV.2015.327.

[2] Han, Song & Pool, Jeff & Tran, John & Dally, William (2015). Learning both Weights and Connections for Efficient Neural Networks. NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 pp. 1135-1143.

[3] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & N. Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.

[4] Devlin, Jacob, et al. Bert: Pre-training of deep bidirectional transformers for language understanding.