MA-BERT: TOWARDS MATRIX ARITHMETIC-ONLY BERT INFERENCE BY ELIMINATING COMPLEX NON-LINEAR FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Due to their superior results, Transformer-based models such as BERT have become de facto standards in many Natural Language Processing (NLP) applications. However, the intensive use of complex non-linear functions within the Transformer architecture impairs its computing efficiency and complicates corresponding accelerator designs, because non-linear functions are generally computation-intensive and require special hardware support. In light of this, we propose MA-BERT, which allows matrix arithmetic-only operations in Transformer-based NLP models and achieves efficient inference with negligible accuracy loss. Specifically, we propose four correlated techniques that include approximating softmax with a two-layer neural network, replacing GELU with ReLU, fusing normalization layers with adjacent linear layers, and leveraging knowledge transfer from baseline models. Through these techniques, we are able to eliminate the major non-linear functions in Transformer-based models and obtain MA-BERT with only matrix arithmetic and trivial ReLU operations without compromising on accuracy. With mainly regular matrix arithmetic operations, MA-BERT enables hardware-friendly processing on various computing engines, including CPUs, GPUs, and customized neural network accelerators. Our experimental results on CPUs show that MA-BERT achieves up to 27% reduction in inference time with comparable accuracy on many downstream tasks compared to the baseline BERT models.

1 INTRODUCTION

Recently, pretrained Transformer-based models such as GPT (Radford et al., 2018) and BERT (Devlin et al., 2018), have consistently dominated the leaderboards for a variety of NLP tasks and even surpassed the human baseline. Consequently, there has been a strong push for these models to be used in many NLP applications. At the same time, they have become a popular research topic in academia and industry. This in turn led to the creation of even better models such as GPT3 (Brown et al., 2020), RoBERTa (Liu et al., 2019), and DeBERTa (He et al., 2020), which further stretched the limits of what such models can achieve. Clearly, the advent of Transformer-based models has brought about a paradigm shift in the NLP field from the pre-Transformer era when recurrent neural networks and their variants used to dominate.

Nonetheless, the exceptional performance of these Transformer-based models partly stems from their deep structure which involves a huge number of parameters and operations during a single forward propagation. These characteristics make it challenging to meet timing requirements and implement on devices with limited memory and computational power. Consequently, many notable works have been proposed to improve their inference performance. In particular to BERT, some works seek to compress the model by reducing the number of encoder layers via knowledge distillation (Sanh et al., 2019; Aguilar et al., 2020; Sun et al., 2020; 2019; Jiao et al., 2019; Xu et al., 2020) with a minor accuracy penalty. Other works focus on model pruning (Gao et al., 2021; Gordon et al., 2020; Voita et al., 2019; Wang et al., 2021), which involves eliminating unessential weights or attention heads, leading to sparser models with a reduced number of parameters and computing operations. Additionally, several works leverage quantization (Kim et al., 2021; Bhandare et al.,

2019; Zafrir et al., 2019) to reduce the precision of weights and computing operations, which lowers memory demands and speeds up inference.

Despite these works, one non-negligible overhead in Transformer-based models is often overlooked – the intensive use of complex non-linear functions. In BERT, these non-linear functions include the softmax operation, Gaussian Error Linear Unit (GELU) activation function (Hendrycks & Gimpel, 2016), and Layer Normalization (LayerNorm) (Ba et al., 2016). Although these functions undeniably play a role in helping the model to learn better during training time, they become a considerable bottleneck during inference as they are not straightforward to evaluate. Furthermore, in hardware accelerator designs, these non-linear functions typically require a separate hardware for acceleration (Liu et al., 2021; Khan et al., 2021), which makes it challenging to be deployed on resource-constrained computing platforms. In Figure 1 and Table 1, we show the breakdown of cycle budget for the BERT$_{base}$ model. For non-linear operations, the cycle budget is defined as the equivalent number of cycles that the matrix multiply takes to process, given the matrix multiply dimensions and the number of multiplications per cycle (Khan et al., 2021). From the figure, we can see that those non-linear operations can take as much as 43% of the processing time. The computation efficiency of BERT has a huge potential to improve if we can optimize these non-linear operations.

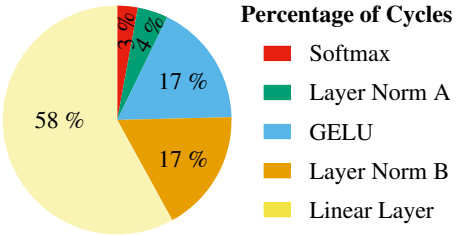

Figure 1: Breakdown of cycle budget

Table 1: Cycle budget for BERT$_{base}$ with sequence length of 128 and 2048 multiplications per cycle (Khan et al. (2021))

Functions	Cycle Budget	Percentage
Softmax	512*	0.8%
Layer Norm A	36,864	4.7%
GELU	147,456	18.6%
Layer Norm B	147,456	18.6%
Linear Layer	454,656	57.4%

* Per Attention Head

In view of this, we introduce an efficient BERT, termed MA-BERT, which makes a novel attempt to completely eliminate the complex non-linear functions in BERT by substituting them with simpler functions. In MA-BERT, four correlated techniques are applied: (1) approximating softmax with a two-layer neural network, (2) replacing GELU with ReLU, (3) fusing normalization layers with adjacent linear layers, and (4) leveraging knowledge transfer from pretrained baseline models. Through these techniques, MA-BERT achieves matrix arithmetic-only and trivial ReLU operations with negligible accuracy loss.

Our experiments show that the performance of MA-BERT on downstream tasks is on par with the corresponding BERT baseline and yet achieves up to 27% performance improvement on general CPUs. By eliminating the complex non-linear functions, the majority of the processing in MA-BERT becomes regular matrix-matrix arithmetic and can be potentially deployed on various computing engines including CPUs, GPUs, and neural network processing units (NPUs) without any hardware modification.

2 Related Work

While so many works focus on model compression techniques, only some paid attention to the optimization of non-linear functions in BERT and other variants of Transformer-based models. For instance, in MobileBERT, Sun et al. (2020) proposed a thin version of BERT$_{\text{large}}$ with reduced hidden dimensions. As part of their operational optimization, they replaced GELU with ReLU and removed LayerNorm to further reduce MobileBERT's latency. However, their results showed that the inclusion of these optimizations degrades the model's performance. Kim et al. (2021), on the other hand, quantized the non-linear functions in BERT to 32-bit integer precision, allowing these non-linear functions to be computed using faster integer arithmetic.

Specific to tackling the latency overhead of LayerNorm, Zhang & Sennrich (2019) proposed root mean square layer normalization which reduces the amount of computation compared to LayerNorm by only using root mean square statistics to normalize inputs. In the field of computer vision, Yao et al. (2021) reintroduced Batch Normalization (BatchNorm) (Ioffe & Szegedy, 2015) as a replacement for LayerNorm (Ba et al., 2016) into vision Transformers and demonstrated that the resulting architecture performs on par with its LayerNorm-based counterpart. Similarly, Shen et al. (2020) proposed PowerNorm as a replacement for LayerNorm in Transformers and demonstrated that PowerNorm can outperform LayerNorm in neural machine translation and language modelling tasks. Due to its proven performance and use of batch statistics, we have decided to adopt PowerNorm in our paper to minimize the overhead introduced by the normalization operations in BERT during inference.

Leaning onto the hardware side, Yu et al. (2022) proposed a general approximation framework which enabled them to convert all the non-linear operations into table look-ups. This helped to reduce the cost of non-linear functions but incur new type of operations and corresponding hardware.

Different from these works, our work strives to completely eliminate the complex non-linear functions in BERT and achieve matrix arithmetic-only operations with trivial ReLU, which could benefit inference on both general computing units and accelerator designs for edge applications.

3 METHOD

This section gives a brief overview of the non-linear functions that exist in BERT and highlights their computation inefficiency during inference. Following that, more efficient methods are proposed.

3.1 SOFTMAX: APPROXIMATING WITH A 2-LAYER NEURAL NETWORK

Every encoder layer in BERT contains a multi-head self-attention (MHA) sublayer followed by a feed forward network (FFN) sublayer. The MHA sublayer endows the model the ability to capture various dependencies between different parts of an input sequence (Vaswani et al., 2017) and is defined as follows:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{1}$$

$$\text{where head}_i = \text{softmax}(\frac{XW_{Q_i}W_{K_i}^T X^T}{\sqrt{d}})XW_{V_i} \tag{2}$$

In each self-attention head, a $s \times s$ scaled dot product of the query and key matrices is passed through a softmax function to obtain the weights on the value matrix. Here, s denotes the sequence length. The softmax function is computed as:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{s} \exp(x_j)} \tag{3}$$

From Eq. 3 it is clear that all inputs along the sequence dimension have to be processed before the softmax value of a particular input can be computed. This becomes prohibitively expensive as the input sequence length increases (Du et al., 2019; Stevens et al., 2021). Moreover, the softmax function contains expensive exponentiation and division operations which further complicate accelerator designs (Hu et al., 2018). In view of this, we decided to find an alternative for the softmax function.

The universal approximation theorem (Hornik et al., 1989) states that a neural network with a single hidden layer and non-linear activation function can approximate any function, given sufficient hidden neurons. As such, we propose to substitute the softmax function in each layer of BERT with a simple 2-layer neural network. ReLU is chosen as the activation function of the hidden layer with hardware reuse in mind since we will also be adopting it in the FFN sublayer. Note that this modification will inevitably fix the maximum input sequence length of our modified model to the input layer size of the neural network. To eliminate the effects of any padding tokens in an input sequence, we multiplied the scaled dot product of the query and key matrices with the input attention mask before feeding it as an input to the neural network.

Given that the same softmax function is being approximated, it should also be possible to approximate softmax using the same neural network across the encoder layers, allowing us to reduce the parameters introduced.

3.2 GELU: REPLACING WITH RELU

Unlike the original Transformer architecture proposed by Vaswani et al. (2017), many Transformer-based NLP models such as BERT use the GELU activation function instead of ReLU in their FFN layers to enhance their learning capability. GELU is defined as follows:

$$\text{GELU}(x) = x \cdot \Phi(x), \text{ where } \Phi(x) = \frac{1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} \exp(-t^2)\, dt}{2} \tag{4}$$

According to Eq. 4, it is clear that the GELU involves complex operations such as exponentiation and integration which are computation-intensive and hard to be implemented efficiently in accelerator designs. One possible way is to employ a look up table to simplify the operations, but doing so entails a trade-off between accuracy and hardware utilization. Other implementations involving function approximations (Hendrycks & Gimpel, 2016; Kim et al., 2021) also exist, but they often involve multiple operations which ends up consuming a notable portion of inference time (Pati et al., 2021). In contrast, ReLU simply outputs zero or the input value depending on the input's sign, and can be implemented easily in hardware using a single multiplexer. With the potential simplification in hardware and speed-up during inference (Sun et al., 2020), we propose to bring ReLU back as a replacement for GELU.

3.3 LAYERNORM: FUSING NORMALIZATION LAYERS WITH ADJACENT LINEAR LAYERS

Every encoder layer in BERT involves two LayerNorm sublayers – one after the MHA sublayer and another after the FFN sublayer. LayerNorm involves the calculation of the mean μ and standard deviation σ statistics along the feature dimension of size K, followed by an affine transformation:

$$\bar{x_i} = \gamma \cdot \frac{x_i - \mu}{\sigma} + \beta, \text{ where } \mu = \frac{1}{K}\sum_{i=1}^{K} x_i,\ \sigma = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(x_i - \mu)^2} \tag{5}$$

This remains true even during inference and demands the need to wait for all features of an input token to be processed before these statistics can be computed. It would be ideal if some form of batch statistics can be used as an alternative to normalize the inputs. Doing so will eliminate the need to compute these statistics and present us the opportunity to fuse with adjacent linear layers in BERT during inference.

Recently, Shen et al. (2020) proposed PowerNorm as a replacement for LayerNorm in Transformers and demonstrated that PowerNorm can outperform LayerNorm in neural machine translation and language modelling tasks. Similar to BatchNorm, PowerNorm normalizes the inputs along the batch dimension of size B and uses running statistics during inference. However, it enforces only unit quadratic mean:

$$\bar{x_i} = \gamma \cdot \frac{x_i}{\psi} + \beta, \text{ where } \psi^2 = \frac{1}{B}\sum_{i=1}^{B} x_i^2 \tag{6}$$

During inference, the parameters γ, ψ, and β become constants which can be merged with the parameters in the adjacent linear layers. Specifically, if we were to replace LayerNorm with PowerNorm, the PowerNorm after the MHA sublayer can be fused with either the linear projection after the concatenation operation in Eq. 1 or the linear projection from the input layer to the intermediate layer of the following FFN. Similarly, the PowerNorm after the FFN sublayer can be fused with either the linear projection from the intermediate layer to the output layer of the FFN or with the computation of the key, query, and value matrices in the next encoder layer. That said, due to the post-normalization architecture of BERT, it is not possible to do a complete fusion. Nonetheless, we can still reduce the two PowerNorm operations in every MA-BERT encoder layer to a single scaling operation during inference, which offers us considerable reduction in computation.

Given the prospect of minimizing the overhead of the normalization operation during inference, we propose to substitute LayerNorm in BERT with PowerNorm. Note that, unlike their work, we did not include a layer-scale operation before PowerNorm to prevent any additional overhead from being introduced.

3.4 KNOWLEDGE TRANSFER

With the three optimization techniques proposed in the previous subsections, we successfully achieve a BERT model with only matrix arithmetic and trivial ReLU operations. However, as a result of the modifications, a large accuracy gap exists between the untrained MA-BERT and the baseline. In order to bridge this gap, we propose to leverage knowledge transfer.

Many studies have demonstrated the effectiveness of knowledge distillation (Hinton et al., 2015) in bridging the performance gap between a student model and a teacher model. The student model is trained to mimic the behaviour of a teacher model using a well-designed cost function that takes into account of the differences between the student output and the teacher output. Unlike many studies which used knowledge distillation to compress BERT into smaller models, we leverage knowledge transfer with the intention to train a model of the same depth and hidden dimensions with a more efficient and hardware-friendly architecture. Note that we are using the term knowledge transfer rather than knowledge distillation, given that the latter is often associated with a smaller model learning from a larger model. The following describes our procedure to transfer knowledge from an existing pretrained $\text{BERT}_{\text{base}}$ model to our modified model, $\text{MA-BERT}_{\text{base}}$.

3.4.1 TEACHER MODEL

We used the pretrained $\text{BERT}_{\text{base}}$ model from HuggingFace as our teacher model. All parameters of the teacher model were frozen to prevent the student model from chasing a moving target.

3.4.2 STUDENT MODEL

For the student model ($\text{MA-BERT}_{\text{base}}$), we aggregated and implemented the three modifications described in the previous subsections on a $\text{BERT}_{\text{base}}$ model and initialized it with its pretrained parameters, wherever possible. Furthermore, the newly added 2-layer neural network was pretrained in advance to approximate the softmax function. For simplicity, the hidden layer size of the 2-layer neural network was set to be the same as the input sequence length. The proposed modifications were only applied on the embedding and encoder layers of BERT, and all parameters in these layers were left unfrozen, except for the position and token type embeddings.

3.4.3 KNOWLEDGE TRANSFER OBJECTIVE

In order for the student to replicate the teacher's behaviour as closely as possible, we adopted a similar layer-wise knowledge transfer technique proposed by Jiao et al. (2019).

First of all, we used the cross entropy (CE) between the student's logits $\mathbf{z}^{\text{S}}$ and the actual labels $\mathbf{y}$ to penalize any wrong classification:

$$L_{hard} = CE(\mathbf{z}^S, \mathbf{y}) \tag{7}$$

Next, we considered the cross entropy between the student's logits and the teacher's logits, $\mathbf{z}^{\text{T}}$:

$$L_{soft} = CE(\mathbf{z}^S/t, \mathbf{z}^T/t) \tag{8}$$

Compared to the actual labels, the teacher's logits provide much more information for the student to learn. A higher temperature t smoothens the probability distribution, which allows the student to better learn the intricacies of the teacher model.

Thirdly, to ensure each layer of our student model, including the embedding layer, learns from the corresponding layer of teacher model, we included the following mean-squared error (MSE) objective:

$$L_{hidn} = \sum_{i=0}^{l} MSE(\mathbf{H}_i^S, \mathbf{H}_i^T) \tag{9}$$

$\mathbf{H}_i^S$ and $\mathbf{H}_i^T$ are the hidden states of the student and teacher models, respectively, and l is the number of layers in the model.

Lastly, given that we are replacing the softmax function in the MHA sublayer, we also penalized the discrepancy between the student's attention outputs $\mathbf{A}_i^S$ and teacher's attention outputs $\mathbf{A}_i^T$ using:

$$L_{attn} = \sum_{i=1}^{l} MSE(\mathbf{A}_i^S, \mathbf{A}_i^T) \tag{10}$$

Altogether, the overall objective function for knowledge transfer can be summarized as follows:

$$L_{net} = (1-\alpha) * L_{hard} + \alpha * L_{soft} + \beta\,(L_{hidn} + \gamma * L_{attn}) \tag{11}$$

α is used to balance the importance of following the actual hard target and the teacher's soft target, β is used to balance the importance of mimicking the teacher's internal layer outputs relative to mimicking the final output, and γ is a factor to scale L_{attn} to a similar magnitude as L_{hidn}. With this overall objective, the student is compelled to not only mimic the final output of the teacher, but also from its hidden layer and attention outputs. Throughout our study, we kept $t = 15$, $\alpha = 0.9$, $\beta = 1$, and $\gamma = 100$.

4 EXPERIMENTS

In this section, we detail our pretraining setup and evaluate the performance of MA-BERT$_{\text{base}}$ on several downstream tasks. We also describe the details of additional studies that we have conducted which attempt to highlight and exploit the benefits of our approach. Throughout this section, all results pertain to a model input sequence length of 128 tokens.

4.1 PRETRAINING SETUP

We pretrained MA-BERT$_{\text{base}}$ via knowledge transfer on a concatenation of BookCorpus (Zhu et al., 2015) and English Wikipedia using only the Masked Language Modelling objective (Devlin et al., 2018) with masking probability of 15%. We used AdamW optimizer with learning rate of $2e^{-5}$, 10,000 warm-up steps from a base learning rate of $5e^{-7}$, and a linear decay of learning rate. Training was done with a batch size of 256 over 3 epochs, with each epoch using a differently masked dataset, on a single NVIDIA GeForce RTX 3090 for a total of 3.5 days.

4.2 EVALUATION ON DOWNSTREAM TASKS

To evaluate the performance of MA-BERT$_{\text{base}}$, we used the GLUE benchmark (Wang et al., 2018), which consists of nine natural language understanding tasks, and the IMDb sentiment classification task (Maas et al., 2011). Following many studies, we excluded WNLI given its small dataset. For each task, we applied task-specific knowledge transfer by having MA-BERT$_{\text{base}}$ learn from a fine-tuned BERT$_{\text{base}}$. The same knowledge transfer objective and hyperparameter settings outlined in section 3.4.3 were used. As our goal is to replicate the performance of the teacher, our baseline for comparison is simply the teacher, which is BERT$_{\text{base}}$ in this case.

For smaller datasets (CoLA, MRPC, RTE and STS-B), we used a batch size of 16 and fine-tuned for 10 epochs, and for the remaining datasets, we used a batch size of 32 and fine-tuned for 5 epochs. The same learning rate of $2e^{-5}$ was used across all tasks and the sequence length was fixed at 128. To account for the variance of different runs, we report the median of 5 runs with different random initialization seeds for each task. The results are shown in Table 2 and 3. Note that the teacher with the best performance among the 5 runs was used to finetune the student during task-specific knowledge transfer. The best performance of both the teacher and student is reported under Appendix A.1.

On average, despite the modifications, MA-BERT$_{\text{base}}$ was able to achieve similar performance on the downstream tasks as compared to BERT$_{\text{base}}$. In fact, for several tasks, MA-BERT$_{\text{base}}$ managed to outperform its teacher. This demonstrates that adopting our approach does not lead to a degradation in the average performance of the model.

4.3 HYPERPARAMETER TUNING OF α AND β

Prior to the collection of results in Table 2 and 3, we conducted a simple hyperparameter study to assess the impact of our choices for α and β in our knowledge transfer objective during task-specific

knowledge transfer. We varied α and β, which effectively control how strictly the student should replicate the teacher's final and hidden layer outputs, and evaluated MA-BERT$_{base}$ on the 4 smaller datasets of the GLUE benchmark. The results are shown in Table 4. Empirically, we found that MA-BERT$_{base}$ performs the best on downstream tasks under the strict guidance of the finetuned BERT$_{base}$ model with the optimal hyperparameter values of $\alpha = 0.9$ and $\beta = 1.0$.

Table 2: Median performance on the development set of GLUE benchmark. Matthews correlation is reported for CoLA while Pearson correlation is reported for STS-B. Accuracy is reported for the remaining tasks. MNLI is the average of matched and mismatched sets. AVG denotes the average of the 8 GLUE tasks.

Model	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	**AVG**
BERT$_{base}$	58.8	84.5	86.3	91.4	91.2	69.3	92.8	89.3	83.0
MA-BERT$_{base}$	59.8 (▲1.0)	84.8 (▲0.3)	87.5 (▲1.2)	91.0 (▼0.4)	91.2 (0.0)	68.6 (▼0.7)	92.5 (▼0.3)	87.8 (▼1.5)	82.9 (▼0.2)
MA-BERT$_{base}$ (Shared Softmax)	59.0 (▲0.2)	84.7 (▲0.2)	87.0 (▲0.7)	91.1 (▼0.3)	91.1 (▼0.1)	70.0 (▲0.7)	92.4 (▼0.4)	87.3 (▼2.0)	82.8 (▼0.3)
DistilBERT	55.1	82.3	85.0	88.4	90.3	62.5	90.8	87.1	80.2
MA-DistilBERT	51.3 (▼3.8)	82.3 (0.0)	86.7 (▲1.7)	88.4 (0.0)	90.3 (0.0)	65.3 (▲2.8)	90.8 (0.0)	87.5 (▲0.4)	80.3 (▲0.1)
DistilRoBERTa	58.3	84.3	85.8	91.2	90.9	69.3	92.9	88.4	82.6
MA-DistilRoBERTa	51.7 (▼6.6)	83.8 (▼0.5)	86.0 (▲0.2)	90.8 (▼0.4)	90.7 (▼0.2)	67.5 (▼1.8)	91.6 (▼1.3)	87.8 (▼0.6)	81.2 (▼1.4)
MA-DistilRoBERTa (Linear Attention)	47.6 (▼10.7)	83.2 (▼1.1)	83.8 (▼2.0)	90.5 (▼0.7)	90.5 (▼0.4)	65.7 (▼3.6)	90.8 (▼2.1)	87.1 (▼1.3)	79.9 (▼2.7)

Table 3: Accuracy on IMDb test set.

Model	IMDb
BERT$_{base}$	89.3
MA-BERT$_{base}$	89.6 (▲0.3)
MA-BERT$_{base}$ (Shared Softmax)	89.6 (▲0.3)
DistilBERT	88.1
MA-DistilBERT	88.2 (▲0.1)
DistilRoBERTa	89.9
MA-DistilRoBERTa	89.8 (▼0.1)
MA-DistilRoBERTa (Linear Attention)	89.6 (▼0.3)

Table 4: α and β hyperparameter tuning results on selected GLUE tasks for MA-BERT$_{base}$. AVG denotes the average of the 4 selected GLUE tasks.

α, β	CoLA	MRPC	RTE	STS-B	**AVG**
0.3, 0.3	58.3	84.3	61.0	86.6	72.6
0.6, 0.6	59.9	86.5	64.6	86.7	74.4
0.9, 0.9	60.4	87.0	68.2	87.7	75.8
0.3, 1.0	60.0	86.8	65.0	87.1	74.7
0.6, 1.0	59.8	86.5	65.3	87.1	74.7
0.9, 1.0	59.8	87.5	68.6	87.8	**75.9**

4.4 Sharing Softmax Approximation Neural Network Across Layers

In the previous subsection, the softmax function in each encoder layer of MA-BERT$_{base}$ were approximated using a different 2-layer neural network. However, given that it is the same softmax function, it would make sense if we can share the same 2-layer neural network across all layers. Doing so would also help us reduce the additional parameters introduced by our approach. In fact, we found that the weights learnt by the 2-layer neural network for each encoder layer in MA-BERT$_{base}$ are largely similar. A visualization of the weights learnt can be found in Figure 5 under Appendix A.2. We attempted to retrain MA-BERT$_{base}$ with parameter sharing in the softmax approximation neural network using the same training procedure. Empirically, we found that sharing the 2-layer neural network to approximate softmax preserves the performance of the modified model.

4.5 Versatility Study

To evaluate the versatility of our approach, we also attempted our modifications and training procedure on HuggingFace's DistilBERT (Sanh et al., 2019) and DistilRoBERTa to produce MA-DistilBERT and MA-DistilRoBERTa, respectively. Following DistilRoBERTa's training procedure, MA-DistilRoBERTA was instead pretrained on the OpenWebText (Gokaslan & Cohen, 2019)

dataset. The remaining training procedure was kept unchanged. We observed that both MA-DistilBERT and MA-DistilRoBERTa suffered a significant drop in performance on the CoLA task. Nonethless, in terms of average performance, MA-DistilBERT managed to slightly outperform its teacher, DistilBERT. On the other hand, although MA-DistilRoBERTa was not able to achieve similar average performance as DistilRoBERTa, it was still able to outperform DistilBERT, which has the same hidden dimensions and depth.

This versatility study also illustrates that our approach is orthogonal to model compression via knowledge distillation. We can first distil a model into a shallower version, then apply our modifications to make it even more efficient and hardware friendly for inference.

4.6 Adopting Linear Self-Attention

With the use of a neural network to approximate the softmax operation, we have inevitably introduced additional matrix multiplication operations. The increase in matrix multiplication scales in $O(s^3)$ if the hidden layer size of the neural network follows the input sequence length s. That said, for relatively short sequence lengths, the increase is marginal. For $s = 128$, the increase in the total number of matrix multiplication compared to the baseline is around 5.4% (Figure 2). Nonetheless, the issue of scalability comes to light.

For longer sequence lengths, we suggest a possible solution that can reduce this overhead. Recently, Wang et al. (2020) proposed self-attention with linear complexity. This is achieved by projecting the key and value matrices from dimensions $\mathbb{R}^{s\times d}$ into smaller dimensions of $\mathbb{R}^{k\times d}$ using the projection matrices $E, F \in \mathbb{R}^{k\times s}$ and setting $k \ll s$:

$$head_i = softmax(\frac{XW_{Q_i}W_{K_i}^T X^T E^T}{\sqrt{d}})FXW_{V_i} \tag{12}$$

Here, d denotes the hidden dimension of the model while k denotes the new projected dimension. By adopting linear self-attention, softmax has to be computed only along the reduced dimension k, which enables us to use a smaller 2-layer neural network to replace softmax. This not only helps to reduce the overheads introduced by the additional neural network to approximate softmax, but also allows us to leverage the speedup offered by the linear self-attention. For instance, by letting $k = 0.5s$, significant savings can be achieved as illustrated in Figure 2. To take this even further, we can also fuse the linear operations in the softmax approximation neural network with both the projection of the key and value matrices during inference, hiding the overhead introduced.

To test this out, we trained MA-DistilRoBERTa with linear attention by setting $k = 64$ using the same procedure. The loss term L_{attn} was removed from the knowledge transfer objective due to the different attention output dimensions of the modified model and DistilRoBERTa. The results are reported in Table 2 and 3. We noticed a 3.3% drop in performance of the modified model compared with the baseline DistilRoBERTa. Nonetheless, it was still able to achieve comparable average performance (0.37% drop) as baseline DistilBERT. A revision on the training procedure might be necessary to further improve its performance and we will leave this to future studies.

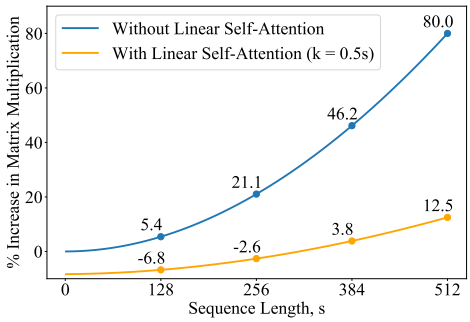

Figure 2: Percentage increase in matrix multiplication in an encoder layer of MA-BERT with and without linear self-attention compared to BERT against input sequence length

4.7 Evaluation on Computational Efficiency

To evaluate the time efficiency of MA-BERT during inference, we implemented DistilBERT and MA-DistilBERT on three different hardware, namely, i7-1260P CPU, RTX 3090 GPU, and Google TPUv2.8. We used C++ with Eigen3 library for CPU implementation, Cuda C++ with Thrust and cuBLAS libraries for GPU implementation, and PyTorch for TPU implementation. Using the IMDb test dataset, we measured the average time taken for a single encoder layer inference. To account for the variance in the time taken, we report the median of 5 different runs. The results are summarized in Table 5. Despite the higher number of matrix multiplications involved, MA-DistilBERT is able to offer an inference time speedup on various hardware. Note that the speedups reported also applies for MA-BERT$_{\text{base}}$ and MA-DistilRoBERTa, given that their encoder layers are all identical. This demonstrates that our approach not only eliminates the need to compute any complex non-linear functions, but also helps to reduce the inference time.

Table 5: Forward propagation time of a single encoder layer in DistilBERT and MA-DistilBERT on different hardware. All parameters and calculations were in 32-bit floating-point precision.

	i7-1260P CPU	RTX 3090 GPU	Google TPUv2.8U
DistilBERT	93.0 ms	2.37 ms	0.584 ms
MA-DistilBERT	68.0 ms	1.39 ms	0.524 ms
Speedup	1.27×	1.41×	1.11×

5 Ablation Study

In this last section, we highlight the importance of having the baseline teacher's guidance during MA-BERT's finetuning phase.

5.1 Removal of Teacher's Guidance

To demonstrate the importance of task-specific knowledge transfer during finetuning, we repeated our finetuning procedure on the GLUE tasks by replacing the knowledge transfer objective with only the hard target loss for our student model to optimize. The results are reported in Table 6. We found that without any guidance from the teacher, the finetuning process is often unstable with the student model ending up performing worse in all tasks. Clearly, knowledge transfer plays a significant role in the success of our approach.

Table 6: Effects of removing the teacher's guidance during finetuning on the GLUE tasks.

Model	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
MA-BERT$_{\text{base}}$								
- With Knowledge Transfer	59.8	84.8	87.5	91.0	91.2	68.6	92.5	87.8
- No Knowledge Transfer	31.5	35.3	80.9	50.5	63.2	61.4	86.8	81.6
MA-DistilBERT								
- With Knowledge Transfer	51.3	82.3	86.7	88.4	90.3	65.3	90.8	87.5
- No Knowledge Transfer	1.8	35.3	68.4	50.7	63.2	53.1	85.6	85.3

6 Conclusion

In this paper, we introduced MA-BERT, an efficient and hardware-friendly version of BERT with all its complex non-linear functions eliminated. Through knowledge transfer during pretraining and finetuning, MA-BERT is able to attain comparable performance as BERT. Yet, compared to BERT, MA-BERT offers a reduction in inference time with potential savings in hardware resources, making it a suitable solution for applications on edge devices. In the future, it would be interesting to study the applicability of our approach on other transformer-based models.

REFERENCES

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7350–7357, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*, 2019.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gaoming Du, Chao Tian, Zhenmin Li, Duoli Zhang, Yongsheng Yin, and Yiming Ouyang. Efficient softmax hardware architecture for deep neural networks. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 75–80, 2019.

Yang Gao, Nicolo Colombo, and Wei Wang. Adapting by pruning: A case study on bert. *arXiv preprint arXiv:2105.03343*, 2021.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Ruofei Hu, Binren Tian, Shouyi Yin, and Shaojun Wei. Efficient hardware architecture of softmax layer in deep neural network. In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pp. 1–5. IEEE, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Hamza Khan, Asma Khan, Zainab Khan, Lun Bin Huang, Kun Wang, and Lei He. Npe: An fpga-based overlay processor for natural language processing. *arXiv preprint arXiv:2104.06535*, 2021.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pp. 5506–5518. PMLR, 2021.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Zejian Liu, Gang Li, and Jian Cheng. Hardware acceleration of fully quantized bert for efficient natural language processing. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 513–516. IEEE, 2021.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Suchita Pati, Shaizeen Aga, Nuwan Jayasena, and Matthew D Sinclair. Demystifying bert: Implications for accelerator design. *arXiv preprint arXiv:2104.08335*, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Powernorm: Rethinking batch normalization in transformers. In *International Conference on Machine Learning*, pp. 8741–8751. PMLR, 2020.

Jacob R Stevens, Rangharajan Venkatesan, Steve Dai, Brucek Khailany, and Anand Raghunathan. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 469–474. IEEE, 2021.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 97–110. IEEE, 2021.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. Bert-of-theseus: Compressing bert by progressive module replacing. *arXiv preprint arXiv:2002.02925*, 2020.

Zhuliang Yao, Yue Cao, Yutong Lin, Ze Liu, Zheng Zhang, and Han Hu. Leveraging batch normalization for vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 413–422, 2021.

Joonsang Yu, Junki Park, Seongmin Park, Minsoo Kim, Sihwa Lee, Dong Hyun Lee, and Jungwook Choi. Nn-lut: neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 577–582, 2022.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pp. 36–39. IEEE, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

A Appendix

A.1 Performance of Best Performing Teacher and Student Models

During task-specific knowledge transfer, the best-performing finetuned teacher was used as the teacher for the pretrained student model. Table 7 shows the best performance achieved by the baseline teacher models and the modified student models among the 5 runs for each task.

Table 7: Best performance on the dev set of GLUE benchmark and test set of IMDb. Matthews correlation is reported for CoLA while Pearson correlation is reported for STS-B. Accuracy is reported for the remaining tasks. MNLI is the average of matched and mismatched sets

Model	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	IMDb
$\text{BERT}_{\text{base}}$	59.4	84.9	87.5	91.5	91.4	69.7	93.0	89.5	89.5
$\text{MA-BERT}_{\text{base}}$	61.3 (▲1.9)	84.9 (0.0)	88.0 (▲0.5)	91.1 (▼0.4)	91.3 (▼0.1)	70.8 (▲1.1)	92.7 (▼0.3)	88.1 (▼1.4)	89.8 (▲0.3)
$\text{MA-BERT}_{\text{base}}$ (Shared Softmax)	62.1 (▲2.7)	85.0 (▲0.1)	87.3 (▼0.2)	91.3 (▼0.2)	91.2 (▼0.2)	70.4 (▲0.7)	92.8 (▼0.2)	87.6 (▼1.9)	89.7 (▲0.2)
DistilBERT	55.5	82.5	86.0	88.7	90.4	64.3	91.4	87.4	88.2
MA-DistilBERT	52.4 (▼3.1)	82.4 (▼0.1)	87.0 (▲1.0)	88.7 (0.0)	90.4 (0.0)	66.4 (▲2.1)	91.4 (0.0)	87.7 (▲0.3)	88.3 (▲0.1)
DistilRoBERTa	59.8	84.5	86.3	91.3	91.0	71.8	93.0	88.8	90.2
MA-DistilRoBERTa	51.8 (▼8.0)	83.9 (▼0.9)	86.8 (▲0.5)	91.1 (▼0.2)	90.8 (▼0.2)	68.2 (▼3.6)	92.0 (▼1.0)	87.9 (▼0.9)	89.8 (▼0.4)
MA-DistilRoBERTa (Linear Attention)	49.7 (▼10.1)	83.4 (▼1.1)	85.8 (▼0.5)	90.6 (▼0.7)	90.5 (▼0.5)	66.4 (▼5.4)	91.1 (▼1.9)	87.5 (▼1.3)	89.7 (▼0.5)

A.2 Interesting Findings on Softmax Approximation

After pretraining $\text{MA-BERT}_{\text{base}}$, we decided to evaluate the effectiveness of the 2-layer neural network in approximating softmax. We fed $\text{MA-BERT}_{\text{base}}$ with the sample input sequence "the quick brown fox jumps over the lazy dog" and plotted out the multi-head self-attention scores produced in each of its encoder layer. We then compared them with the corresponding encoder layer attention scores in $\text{BERT}_{\text{base}}$. Figure 3 shows a side-by-side comparison of the attention scores produced in the 4th, 8th, and 12th encoder layers of $\text{MA-BERT}_{\text{base}}$ and $\text{BERT}_{\text{base}}$. We observe that the attention scores produced by both models are similar, which demonstrates the effectiveness of the 2-layer neural network in approximating softmax. The same observation is made when we compare the attention outputs in MA-DistilBERT and DistilBERT in Figure 4.

Moreover, we decided to look into the weights that are learnt by the 2-layer neural network to investigate if any meaningful mappings are learnt. Figure 5 shows the learnt weights mapping the input layer to the hidden layer and the hidden layer to the output layer for each encoder layer. Interestingly, despite giving $\text{MA-BERT}_{\text{base}}$ the opportunity to learn a different mapping function to approximate softmax in different encoder layers, each 2-layer neural network ended up learning a similar mapping. In fact, when we added the constraint of sharing the 2-layer neural network across all 12 encoder layers in $\text{MA-BERT}_{\text{base}}$ (Shared Softmax), a similar mapping is also learnt. Note

that the mappings are rather sparse and do not seem to be random, which suggests that the 2-layer neural network has learnt something meaningful. The same observation is made when we looked into the learnt weights in MA-DistilBERT in Figure 6. In fact, we can see that the mappings learnt in MA-DistilBERT and MA-BERT$_{\text{base}}$ are similar as well. This further suggests that there is some underlying function that can approximate the role of softmax in self-attention and we believe that this will serve as an interesting point for future research.

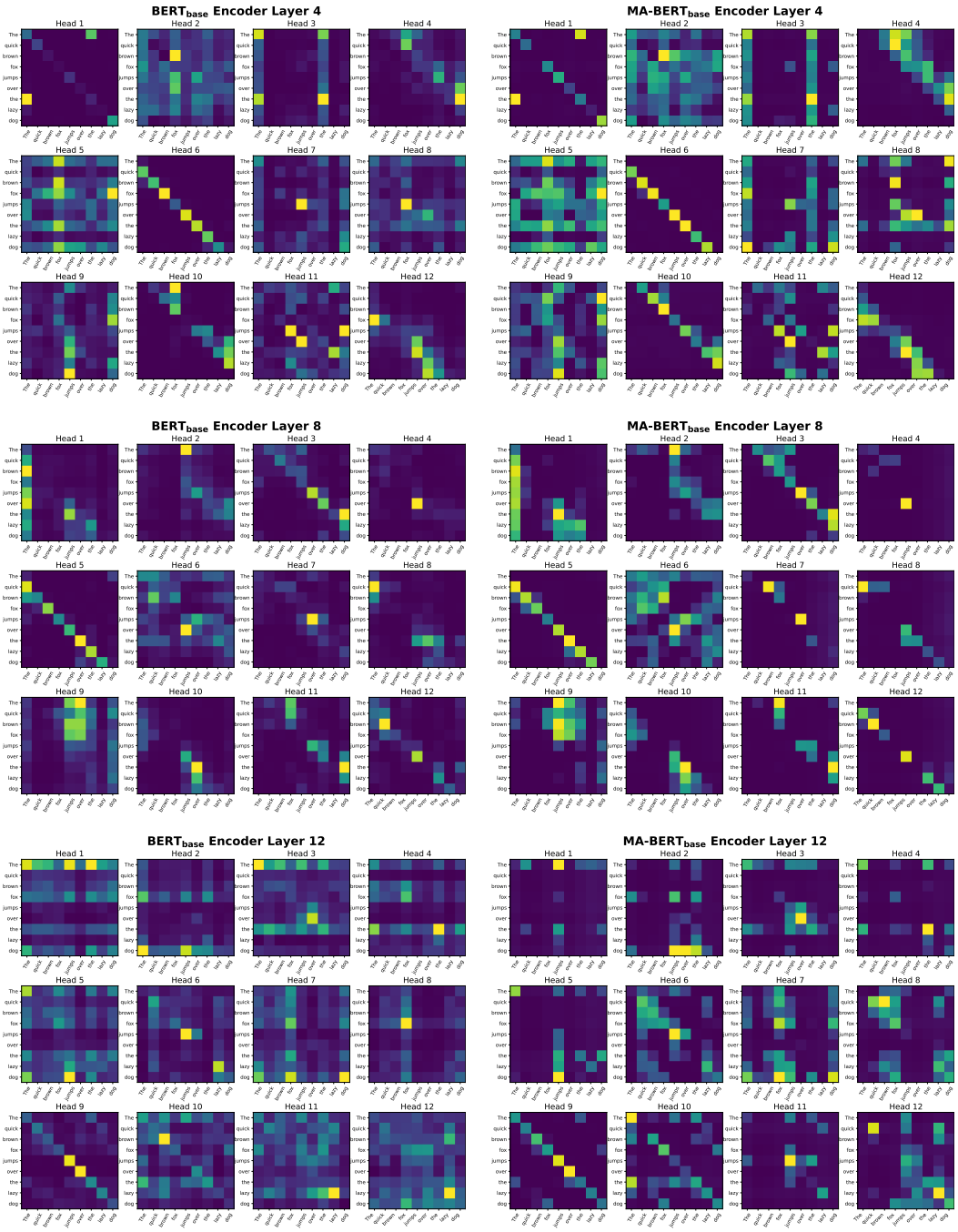

Figure 3: Attention scores produced after multi-head self-attention in encoder layer 4, 8, 12 of BERT$_{\text{base}}$ (left) and MA-BERT$_{\text{base}}$ (right).

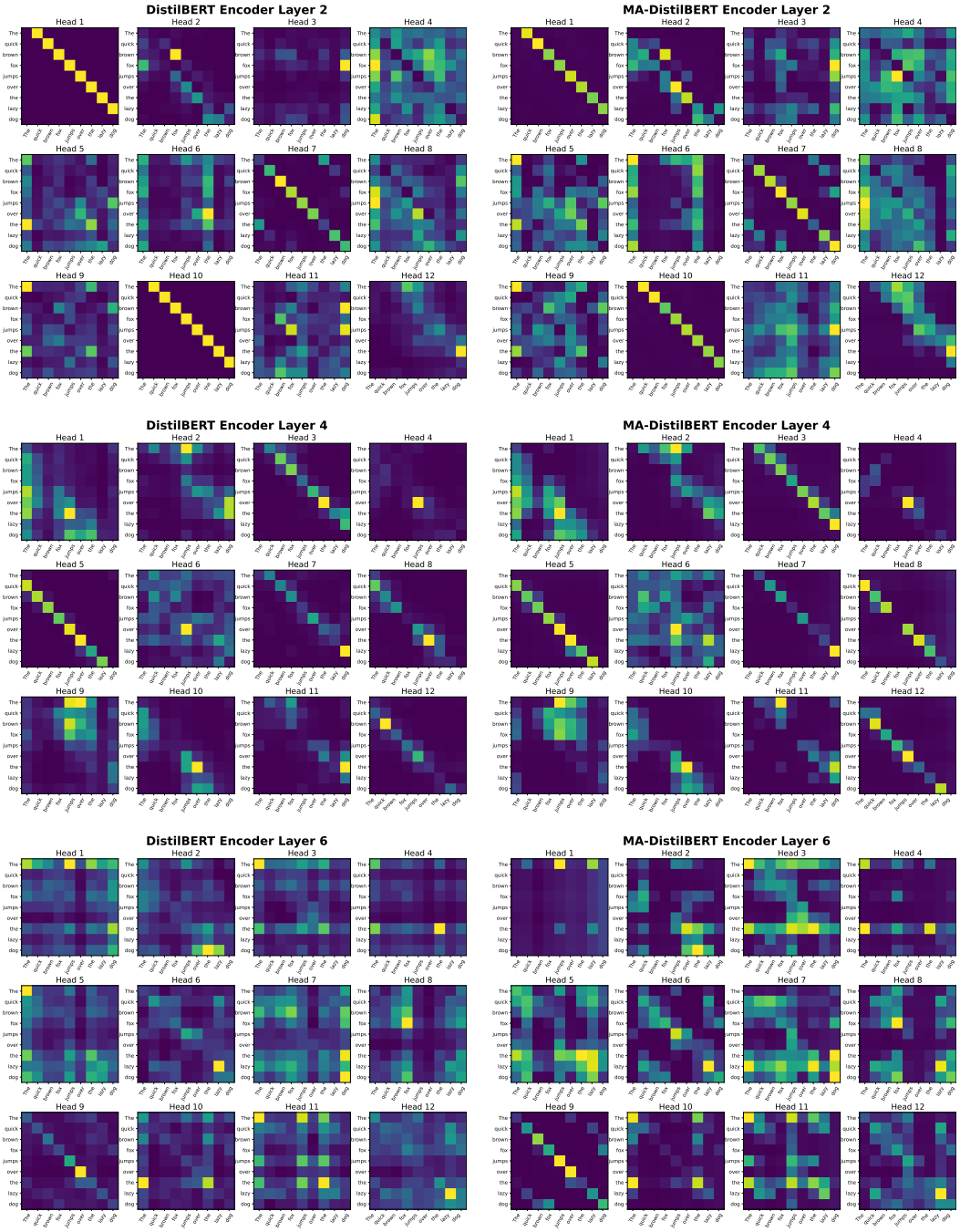

Figure 4: Attention scores produced after multi-head self-attention in encoder layer 2, 4, 6 of DistilBERT (left) and MA-DistilBERT (right).

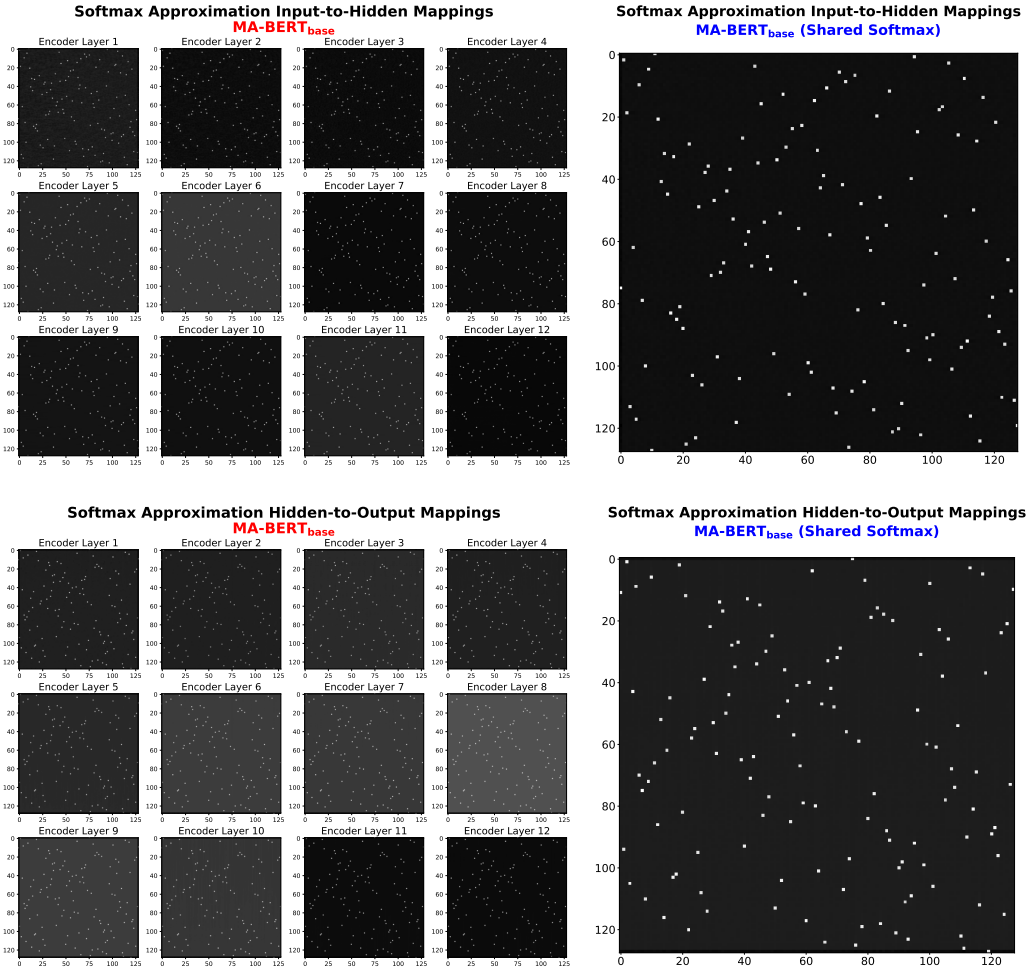

Figure 5: Input-to-hidden (top) and hidden-to-output (bottom) mappings learnt by the 2-layer neural network in each MA-BERT$_{\text{base}}$ encoder layer and the shared 2-layer neural network in MA-BERT$_{\text{base}}$ (Shared Softmax).

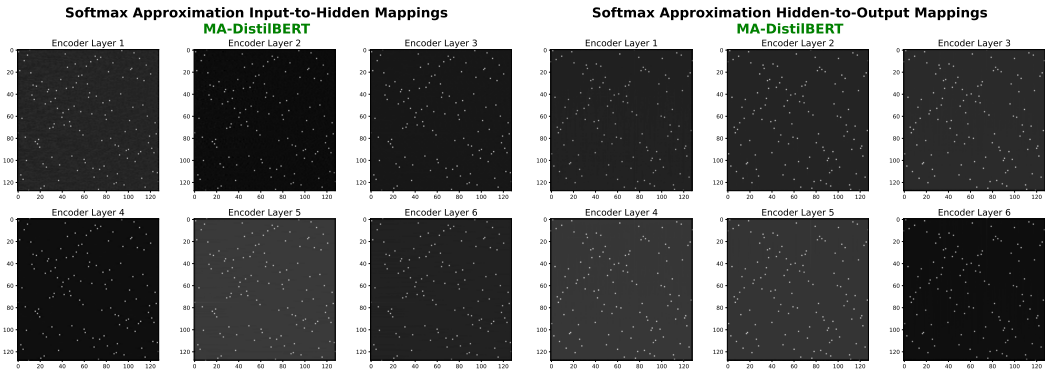

Figure 6: Input-to-hidden (left) and hidden-to-output (right) mappings learnt by the 2-layer neural network in each MA-DistilBERT encoder layer.