| Reviewer 1 |
| --- |
| Thank you for taking the time to review our paper. We are happy to hear that you have found our work well-motivated and easy to understand.<br><br>The following are our responses to your comments on the shortcomings of our paper. |
| **Comment 1:** In general, the contribution of this paper is rather limited since each of those techniques has been previously proposed. |
| While we acknowledge that some of the techniques used in our paper have been previously proposed, we believe the following contributions are noteworthy:<br><br>1. **The use of a 2-layer neural network to approximate the softmax function in the self-attention layers**. We have added additional empirical evidence in the appendix that proves its success and further suggests the existence of some underlying function that can approximate what softmax is doing. We hope you can take some time to look at the new materials that we have added.<br><br>2. **The use of *purely* PowerNorm to replace LayerNorm to achieve similar results with baselines**. To elaborate on this, as mentioned in the original PowerNorm paper, Shen et al. (2020) involved a layer-scale layer before applying PowerNorm. The layer-scale layer is in fact the root mean square layer normalization (RMS LayerNorm) proposed by Zhang & Sennrich (2019). As such, what Shen et al. (2020) have done was replace LayerNorm in transformers with RMS LayerNorm and PowerNorm to achieve better performance in several NLP tasks. This is different from what we implemented as we did not include RMS LayerNorm in MA-BERT. Moreover, given that RMS LayerNorm was shown to yield comparable performance against LayerNorm, there exists some ambiguity on the role of PowerNorm in helping transformers achieve good performance.<br><br>3. **The complete removal of all complex non-linear functions (GELU, Softmax and LayerNorm) in BERT** to achieve matrix arithmetic-only operations with trivial ReLU in MA-BERT, which can bring potential hardware savings. To the best of our knowledge, we believe that our paper is the first to attempt to eliminate the complex non-linear functions in BERT and we have demonstrated that it is possible to achieve similar performance as the baseline with the aid of knowledge transfer. |
| **Comment 2:** There are no results for inference on GPUs while it has been mentioned as one of the motivations of this work in the introduction. |
| Due to the lack of time when we first submitted the paper, we were unable to include any GPU results then. However, we now have the |

inference results on GPU. We implemented DistilBERT and MA-DistilBERT in Cuda C++ using Thrust and cuBLAS libraries with a single NVIDIA GeForce RTX 3090 as the GPU. The following shows the median GPU inference time per encoder layer for DistilBERT and MA-DistilBERT after 5 different runs of 1000 inferences each using the IMDb test dataset:

| | |
|---|---|
| DistilBERT | 2.37 ms |
| MA-DistilBERT | 1.39 ms |

Based on our implementation, we see that MA-DistilBERT is able to offer around 41% speedup on GPU as compared to the baseline DistilBERT. Note that $BERT_{base}$ is just DistilBERT with twice the number of encoder layers and the speedup is applicable to $BERT_{base}$ as well.

**Comment 3**: There is no direct comparison with prior works such as Linformer?

The following reasons are why we were unable to make a direct comparison with Linformer:

1. The speedup presented in their paper applies for an input sequence length of 512 and beyond as they are focusing on general Transformers rather than BERT. Note that BERT has a maximum input sequence length of just 512 and our paper focuses on a sequence length of 128, which is sufficient for many NLP tasks.

2. The source code for Linformer has not been open-sourced. While there exist some GitHub repositories that attempted to implement Linformer, they are not produced by the authors of the paper. Even if we utilize these repositories, the pre-trained checkpoints of Linformer aren't available. This means that for us to compare accuracy performance with Linformer, we will need to train Linformer from scratch, which is prohibitively expensive.

**Comment 4**: Where does the proposed method stand w.r.t. prior works that also try to reduce the complexity of BERT models such as Linformer and I-BERT? The GELU and softmax functions can be approximated using polynomials similar to I-BERT with no impact on accuracy.

**Comparison with Linformer**

Linformer's goal was to reduce the time and space complexity of Transformers, particularly, in its self-attention mechanism. Different from our work, the complex non-linear functions (Softmax, LayerNorm, and GeLU) still exist in Linformer's architecture and so do the drawbacks of these non-linear functions.

As illustrated in our paper, rather than viewing Linformer's method as a competing method, we believe our method can be complementary to Linformer's and vice-versa. Eliminating complex non-linear functions allows Linformer to have a more hardware-efficient inference while adopting Linformer's method in MA-BERT allows us to leverage its time-saving.

**Comparison with I-BERT**

I-BERT's goal was to achieve integer-only arithmetic and this was achieved via using integer-only approximation methods for nonlinear operations. In comparison, our main objective is to achieve a BERT model with purely matrix arithmetic and trivial ReLU operations, which is accomplished by eliminating all complex non-linear functions.

As stated in our paper, for LayerNorm, all features of an input token need to be processed before the mean and standard deviation statistics can be computed. Similarly, for softmax during self-attention, all inputs along the sequence dimension have to be processed before the softmax of a particular input can be computed. Both drawbacks remain in I-BERT since the non-linear functions are simply replaced with integer approximations. However, in MA-BERT, PowerNorm (which uses batch statistics during inference) takes the place of LayerNorm while a 2-layer neural network replaces Softmax. This has allowed us to achieve a modified BERT architecture where the aforementioned drawbacks are no longer present.

Moreover, for accelerator designs, with all operations (except ReLU) converted to matrix arithmetic, the same existing matrix arithmetic hardware can potentially be reused. On the other hand, the integer approximations of Softmax, LayerNorm, and GELU still require separate hardware.

In terms of speedup, it is difficult for us to do a fair comparison with I-BERT as they have quantized all operations to integer precision while our operations are still in 32-bit floating point precision. Nonetheless, with all operations converted to matrix arithmetic, we are optimistic that we can convert them to integer precision in the future to achieve even faster inference speedup than what we have achieved currently with floating point precision. In fact, we believe that quantizing MA-BERT would be less challenging than quantizing BERT as we no longer have to consider the quantization and de-quantization procedures that were needed to transition between linear operations and non-linear operations.

**Comment 5**: Finally, what speedup can be obtained when using MA-DistilBERT or MA-DistilRoBERTa?

Kindly note that the speedup presented under section 4.7 Evaluation of Computational Efficiency refers to the speedup achieved for a single encoder layer in BERT and MA-BERT. The encoder layers in $\text{BERT}_{\text{base}}$, DistilBERT and DistilRoBERTa are identical. The only difference is that DistilBERT and DistilRoBERTa consist of a stack of 6 encoder layers while $\text{BERT}_{\text{base}}$ has a stack of 12 encoder layers. Hence, the speedup presented is applicable to all variants of MA-BERT.

We hope that we have addressed most, if not all, of your current concerns. Nonetheless, if you still have any other doubts about our

paper, do let us know and we would be more than happy to clarify them.

**Reviewer 2**

Thank you for taking the time to review our paper. We are glad to hear that you enjoyed our introduction and found our paper simple to understand.

The following are our responses to your comments on the shortcomings of our paper.

**Comment 1:** The reasons why you can approximate the softmax using a 2-layer NN are unclear. The argument in this paper is poor (universal approx. theory). The reasons why the approximations work is not clear. At least an intuitive explanation would have been good. Universal approx. theory-based explanation is usually too sparse. The authors should explain why softmax can be approximated using a 2-layer FFN with ReLU. It is not clear to me. Citing universal approx. theory is not enough.

Firstly, we show that as the number of hidden neurons increases, the mean squared error decreases.

| Number of Hidden Neurons | Mean Squared Error |
|---|---|
| 32 | 1.76e-4 |
| 128 | 9.01e-5 |
| 512 | 6.08e-5 |
| 2048 | 5.29e-5 |
| 8192 | 4.90e-5 |
| 16384 | 3.59e-5 |

To further show that a 2-layer neural network can approximate the workings of softmax during self-attention, we added the following empirical evidence to the appendix of our paper:

1. A visualization of the output of multi-head self-attention for various layers of MA-BERT$_{base}$ and BERT$_{base}$. It can be observed that the outputs produced by MA-BERT$_{base}$ and BERT$_{base}$ after multi-head self-attention are similar. Do note that these are the outputs produced right after the 2-layer neural network in MA-BERT$_{base}$ and the softmax function in BERT$_{base}$.

2. A visualization of the learnt weights mapping the input layer to the hidden layer and the hidden layer to the output layer for the 2-layer neural networks. We found that despite giving MA-BERT$_{base}$ the opportunity to learn a different neural network for different encoder layers, it ended up learning a similar mapping to approximate softmax in each encoder layer. In fact, when we added the constraint of sharing the 2-layer neural network across all 12 encoder layers in MA-BERT$_{base}$ (Shared Softmax), we found that a similar mapping is 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 observations mentioned in both points 1 and 2 were made for MA-DistilBERT$_{base}$ as well.

We hope that you can take some time to look at the supporting evidence added in the appendix and be convinced that there is perhaps some underlying function capable of approximating what the softmax function is doing in the self-attention layers. Undeniably, it might not be the exact softmax function that the 2-layer neural network is trying to learn. Rather, it could be a generalization of what the softmax function is trying to achieve and we believe that this finding could be of some value to the ICLR community as well.

**Comment 2:** No hardware results, but just software CPU implementation.

Due to the lack of time when we first submitted the paper, we were unable to include any results on AI accelerators and GPUs. However, we now have inference results on GPU and TPU.

**GPU**

We implemented DistilBERT and MA-DistilBERT in Cuda C++ using Thrust and cuBLAS libraries with a single NVIDIA GeForce RTX 3090 as the GPU. The following shows the median GPU inference time per encoder layer for DistilBERT and MA-DistilBERT for 5 different runs of 1000 inferences each using the IMDb test dataset:

| | |
|---|---|
| DistilBERT | 2.37 ms |
| MA-DistilBERT | 1.39 ms |

From our results, we can see that MA-DistilBERT is able to offer around 41% speedup on GPU as compared to the baseline DistilBERT.

**TPU**

Furthermore, we implemented MA-DistilBERT and MA-DistilBERT on Google's TPUv2.8 using PyTorch's framework. The following shows the median time taken per encoder layer for 5 different runs on the entire IMDb test dataset:

| | |
|---|---|
| DistilBERT | 0.584 ms |
| MA-DistilBERT | 0.524 ms |

Despite having more matrix multiplications, MA-DistilBERT can offer a speed-up of around 1.11x on TPU using PyTorch's framework. Note that the non-linear functions in PyTorch are highly optimized and the fact MA-DistilBERT is able to run faster despite the higher number of matrix multiplication clearly demonstrates that our approach does not only just eliminates the need to compute any complex non-linear functions but also helps to reduce the inference time.

Note that $\text{BERT}_{\text{base}}$ is just DistilBERT with twice the number of encoder layers and the speedups reported are applicable to $\text{BERT}_{\text{base}}$ as well.

We hope that we have addressed most, if not all, of your current concerns. Nonetheless, if you still have any other doubts about our paper, do let us know and we would be more than happy to clarify them.

**Reviewer 3**

Thank you for taking the time to review our paper. We are happy to hear that you have found our work relevant to the ICLR community and that our contributions are solid.

The following are our responses to your comments on the shortcomings of our paper.

**Comment 1:** It is recommended to evaluate the proposed method executed on AI accelerators, rather than only on CPUs.

Due to the lack of time when we first submitted the paper, we were unable to include any results on AI accelerators and GPUs. However, we now have inference results on GPU and TPU.

**GPU**

We implemented DistilBERT and MA-DistilBERT in Cuda C++ using Thrust and cuBLAS libraries with a single NVIDIA GeForce RTX 3090 as the GPU. The following shows the median GPU inference time per encoder layer for DistilBERT and MA-DistilBERT for 5 different runs of 1000 inferences each using the IMDb test dataset:

| | |
|---|---|
| DistilBERT | 2.37 ms |
| MA-DistilBERT | 1.39 ms |

From our results, we can see that MA-DistilBERT is able to offer around 41% speedup on GPU as compared to the baseline DistilBERT.

**TPU**

Furthermore, we implemented MA-DistilBERT and MA-DistilBERT on Google's TPUv2.8 using PyTorch's framework. The following shows the median time taken per encoder layer for 5 different runs on the entire IMDb test dataset:

| | |
|---|---|
| DistilBERT | 0.584 ms |
| MA-DistilBERT | 0.524 ms |

Despite having more matrix multiplications, MA-DistilBERT can offer a speed-up of around 1.11x on TPU using PyTorch's framework. Note that the non-linear functions in PyTorch are highly optimized and the fact MA-DistilBERT is able to run faster despite the higher

number of matrix multiplication clearly demonstrates that our approach not just eliminates the need to compute any complex non-linear functions but also helps to reduce the inference time.

Note that $BERT_{base}$ is just DistilBERT with twice the number of encoder layers and the speedups reported are applicable to $BERT_{base}$ as well.

**Comment 2:** The reason why the knowledge transfer is needed (instead of training from scratch the MA-BERT) is unclear. Please provide more details on it.

Intuitively, having a good baseline model to guide a student model allows the student model to learn more easily and quickly as compared to having the student model learn from scratch without any guidance. In fact, we can also draw parallels between how effectively we, as humans, can learn with and without a teacher's guidance.

For more concrete proof, we conducted an experiment by training MA-$BERT_{base}$ from scratch, as you have suggested, and we found that MA-$BERT_{base}$ had difficulty learning - its Masked Language Modelling (MLM) loss did not decrease after 1 epoch of training and the highest MLM accuracy achieved on our validation dataset was only 4.5%. For MA-DistilBERT, while it was able to learn, the highest MLM accuracy achieved after 3 epochs of pretraining was 26.3%. In comparison, with knowledge transfer, MA-$BERT_{base}$ and MA-DistilBERT achieved a maximum MLM accuracy of 56.0% and 53.2%, respectively. Note that these accuracies are similar to what were achieved by $BERT_{base}$ and DistilBERT. Clearly, knowledge transfer helps MA-BERT to learn more easily.

Moreover, in the original BERT paper, Devlin et al. (2018) mentioned that $BERT_{base}$ was pre-trained for approximately 40 epochs over the BookCorpus and Wikipedia Corpus using a total of 16 TPU chips, which took around 4 days to complete. If we were to train MA-BERT from scratch, similar hardware resources and time would be required, assuming that MA-BERT could even achieve a similar performance as BERT. This is clearly prohibitive to anyone who is conducting research with an academic budget. In many cases, only large industry players such as Google, Facebook, and Microsoft can afford to invest such resources. With knowledge transfer, MA-$BERT_{base}$ was able to achieve good performance that is on par with its teacher, $BERT_{base}$, with just 3 epochs of pretraining using a single GPU. The same applies to MA-DistiilBERT and MA-DistilRoBERTa. Evidently, the benefits of knowledge transfer in terms of time and resource savings can be seen.

**Comment 3:** The results do not seem to provide a significant improvement compared to the related works.

While our improvements may not be deemed as significant as compared to related works, we believe that our paper is still of value to the ICLR community, as you have indicated.

In our paper, we proposed several modifications to the architecture of BERT to allow a more hardware-efficient inference. To the best of our knowledge, we believe that our paper is the first to attempt to eliminate the complex non-linear functions in BERT and we have

demonstrated that it is possible to achieve similar performance as the baseline with the aid of knowledge transfer.

Furthermore, we managed to approximate the workings of softmax in the self-attention layers using a 2-layer neural network. We have added additional empirical evidence in the appendix that demonstrates its success and further suggests the existence of some underlying function that can approximate what softmax is doing. We believe that this will be an interesting point for future research.

Lastly, we believe that our work can path the way for future works which can bring MA-BERT to greater heights. One possible direction would be to quantize MA-BERT to integer precisions for even more efficient inference. Due to the complex non-linear functions in BERT, many previous works on quantizing BERT have involved well-thought-out quantization and de-quantization procedures during the transitions between linear operations and non-linear operations. This is due to the fact that non-linear operations typically require higher precision in order to achieve adequate accuracy. As such, given that all operations (except ReLU) in MA-BERT can be expressed as matrix arithmetic, we believe that quantizing MA-BERT would be less challenging and can serve as a possible direction for research.

**Comment 4:** It would be useful to provide the source code for reviewers' inspection during the rebuttal.

As you requested, the source code can be found in the following link: https://anonymous.4open.science/r/MA-BERT-32C1/

You will be able to find the code for MA-BERT$_{base}$, MA-BERT$_{base}$ (Shared Softmax), and MA-DisitlBERT as well as their pre-trained checkpoints that we have obtained through knowledge transfer from BERT$_{base}$ and DistilBERT. We have tested and included a Jupyter Notebook to load the different models from its pre-trained checkpoint for your convenience. Unfortunately, due to copyright reasons, we are unable to release the training code as of now.

We hope that we have addressed most, if not all, of your current concerns. Nonetheless, if you still have any other doubts about our paper, do let us know and we would be more than happy to clarify them.