Simple Recurrence Improves Masked Language Models

Anonymous ACL submission

Abstract

In this work, we explore whether modeling recurrence into the Transformer architecture can both be beneficial and efficient, by building an extremely simple recurrent module into the Transformer. We compare our model to baselines following the training and evaluation recipe of BERT. Our results confirm that recurrence can indeed improve Transformer models by a consistent margin, without requiring low-level performance optimizations, and while keeping the number of parameters constant. For example, our base model achieves an absolute improvement of 2.1 points averaged across 10 tasks and also demonstrates increased stability in fine-tuning over a range of learning rates.

1 Introduction

While the Transformer (Vaswani et al., 2017) relies solely on attention mechanisms for sequence modeling, many recent works have incorporated recurrence into the architecture and demonstrated superior performance in various applications. For example, such modifications were shown to be beneficial for modeling long-range inputs (Hutchins et al., 2022), accelerating language model training (Lei, 2021) and improving translation and speech recognition systems (Hao et al., 2019; Pan et al., 2022; Chen et al., 2018).

Even though combining attention and recurrence is useful in many cases, very little efforts have gone into language model pre-training and fine-tuning. In particular, one open question is whether a combined model can be pre-trained and fine-tuned to achieve stronger accuracy compared to its attention-only counterparts.

We study this question in the case of masked language model training, specifically BERT (Devlin et al., 2019). Unlike previous work (Huang et al., 2020), we are interested in retaining the training efficiency of the model when combining attention and recurrence. That is, the amount of parameters and computation should remain comparable to the baseline Transformer model. However, making a recurrent model operating at a similar computation throughput as attention can be challenging, such as requiring CUDA implementations for GPUs (Appleyard et al., 2016; Bradbury et al., 2017). To mitigate this issue, we propose a simple recurrent implementation which we call SwishRNN. SwishRNN uses minimal operations in the recurrence step in order to accelerate computation, and can run on both TPUs and GPUs using a few lines of code in machine learning libraries such as Tensorflow. We incorporate SwishRNN into BERT by substituting the feed-forward layers and keeping the same number of model parameters.

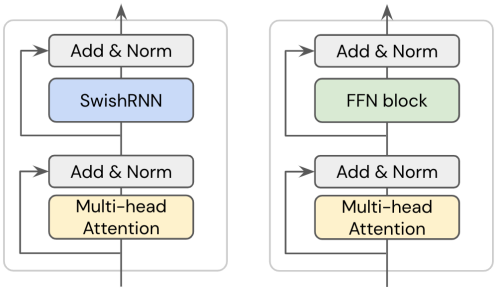

Figure 1: Our model architecture (left) and the original Transformer (right) for masked language modeling. We replace the feed-forward blocks with light-weight recurrence, which is interleaved with attention blocks.

We pre-train our model and BERT baselines using the standard Wikipedia+Book corpus, and compare their fine-tuning performance on 10 tasks selected in the GLUE and SuperGLUE benchmark. Our results confirm that modeling recurrence jointly with attention is indeed helpful, resulting in an average improvement of 2.1 points for the BERT-base models and 0.6 points for the large models. The combined model also exhibits better stability, achieving more consistent fine-tuning results over a range of learning rates.

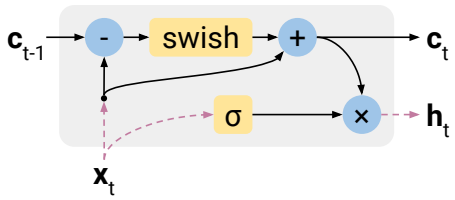

Figure 2: The SwishRNN cell. A red dotted line represents a linear transformation.

2 Model

In this section, we first give a quick overview of our model architecture and then describe the recurrence module SwishRNN in more details.

2.1 Notation and Background

The Transformer architecture interleaves a multi-headed attention block, F_{att}, with feed forward block, F_{ffn}, as shown in Figure 1. Between each block is a residual connection and layer normalization that we denote as $F_{\text{Add+Norm}}$. These functions are defined in the Appendix for completeness.

At each layer k, the hidden state of a Transformer is represented by an $l \times d$ matrix $\boldsymbol{X}^k$, where l is the sequence length and d the hidden size[1]. We define the intermediate hidden state $\bar{\boldsymbol{X}}^k$ and input to the next layer $\boldsymbol{X}^{k+1}$ as:

$$\bar{\boldsymbol{X}}^k := F_{\text{Add+Norm}}\left(F_{\text{att}}(\boldsymbol{X}^k), \boldsymbol{X}^k\right)$$
$$\boldsymbol{X}^{k+1} := F_{\text{Add+Norm}}\left(F_{\text{ffn}}(\bar{\boldsymbol{X}}^k), \bar{\boldsymbol{X}}^k\right) \quad (1)$$

2.2 Architecture

Compared to the original architecture, we simply replace every feed-forward block F_{ffn} with a recurrence block as shown in Figure 1.

SwishRNN Modern accelerator hardwares such as TPUs and GPUs are highly optimized for matrix multiplications, making feed-forward architectures such as attention very efficient. Recurrent networks (RNNs) however involves sequential operations that cannot run in parallel. In order to achieve a training efficiency comparable to the original Transformer, we use minimal sequential operations and demonstrate they are sufficient to improve the modeling power.

Specifically, SwishRNN uses only two matrix multiplications and an element-wise sequential pooling operation. Let $\bar{\boldsymbol{x}}[i] := \bar{\boldsymbol{X}}[i, :]$ be the intermediate hidden vector of the i-th position (from Eq. 1). SwishRNN first computes two linear transformations of $\bar{\boldsymbol{X}}$:

$$\bar{\boldsymbol{X}}_1 = \bar{\boldsymbol{X}}\boldsymbol{W}_1, \quad \bar{\boldsymbol{X}}_2 = \bar{\boldsymbol{X}}\boldsymbol{W}_2 \quad (2)$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are $d \times d'$ parameter matrices optimized during training, d is the input and output dimension of the model, and d' is the intermediate dimension for recurrence. The hidden vectors $\{\boldsymbol{c}[i]\}_{i=1}^{l}$ are calculated as follows

$$\boldsymbol{c}[i] = \texttt{Swish}\left(\boldsymbol{c}[i\text{-}1] - \bar{\boldsymbol{x}}_1[i]\right) + \bar{\boldsymbol{x}}_1[i] \quad (3)$$

where $\texttt{Swish}()$ is the element-wise Swish activation function (Ramachandran et al., 2018).[2] We use a $l \times d'$ matrix $\boldsymbol{C}$ to represent the concatenated version of $\{\boldsymbol{c}[i]\}_{i=1}^{l}$, and set $\boldsymbol{c}[0]$ as an all-zero vector for simplicity. Intuitively, step (3) can be interpreted as a pooling operator where the greater value between $\boldsymbol{c}[i\text{-}1]$ and $\bar{\boldsymbol{x}}_1[i]$ are selected.[3]

The output vectors are obtained using a multiplicative gating similar to other RNNs such as LSTM, followed by a linear layer with weights $\boldsymbol{W}_3$:

$$\mathbf{H} = \boldsymbol{W}_3\left((\mathbf{C} + \boldsymbol{b}_c) \odot \sigma(\mathbf{X}_2 + \boldsymbol{b}_\sigma)\right) + \boldsymbol{b}_3 \quad (4)$$

where $\sigma()$ is a gating activation function. We experimented with `sigmoid` activation and the `GeLU` activation (Hendrycks and Gimpel, 2016) used in BERT, and found the latter to achieve lower training loss. Finally, analogous to Eq. 1, we set $\boldsymbol{X}_{k+1} := F_{\text{Add+Norm}}(\boldsymbol{H}, \bar{\boldsymbol{X}}^k)$.

Speeding up recurrence In our experiments, we implement the recurrence step (2) using the `scan()` function in Tensorflow. Our model using this simple implementation runs 40% slower than the standard Transformer, but is already much faster than other heavier RNNs such as LSTM. For example, a Transformer model combined with LSTM can run multiple times slower (Huang et al., 2020).

We further improve the speed by increasing the step size for the RNN. Specifically, $\mathbf{c}[i]$ is calculated using $\mathbf{c}[i-k]$ and $\mathbf{x}_1[i]$ with a step size $k > 1$. Each recurrent step can process k consecutive tokens at a time and only $\lceil l/k \rceil$ steps are needed. In our experiments, we interleave the step size $k \in \{1, 2, 4\}$ across recurrent layers and found this to perform on par with using a fixed step size of 1.

[1] For simplicity of notation, the l superscript is only included when necessary

[2] $\texttt{Swish}(\boldsymbol{x}) = \texttt{sigmoid}(\alpha \cdot \boldsymbol{x} + \beta) \cdot \boldsymbol{x}$. We initialize $\alpha = \mathbf{1}$ and $\beta = \mathbf{0}$ and optimize both scalar vectors during training.

[3] To see this, note $\boldsymbol{c}[i] = \bar{\boldsymbol{x}}[i]$ if $\bar{\boldsymbol{x}}[i] \gg \boldsymbol{c}[i\text{-}1]$, and $\boldsymbol{c}[i] = \boldsymbol{c}[i\text{-}1]$ if $\bar{\boldsymbol{x}}[i] \ll \boldsymbol{c}[i\text{-}1]$.

	BoolQ	CoLA	MNLI	MRPC	MultiRC	QNLI	QQP	RTE	SST2	STSB	Avg
Base model (12 layers, $d = 768$)											
BERT-orig	73.3%	**82.0%**	84.8%	88.5%	69.5%	91.0%	87.7%	64.0%	93.7%	84.2%	81.9%
BERT-rab	70.1%	73.7%	85.4%	89.8%	70.3%	92.1%	87.5%	66.4%	91.1%	83.8%	81.0%
Ours	**77.9%**	80.8%	**85.9%**	**90.4%**	**74.2%**	**92.5%**	**88.2%**	**70.8%**	**93.8%**	**85.7%**	**84.0%**
Large model (24 layers, $d = 1024$)											
BERT-orig	84.5%	81.4%	**89.0%**	**93.5%**	79.6%	94.2%	88.6%	84.0%	95.3%	**87.9%**	87.8%
BERT-rab	84.5%	75.0%	88.9%	92.2%	80.9%	94.2%	88.4%	78.8%	93.1%	87.2%	86.3%
Ours	**86.1%**	**84.8%**	88.9%	92.9%	**81.2%**	**94.3%**	**88.7%**	**85.0%**	95.3%	87.2%	**88.4%**
Previously reported results (Large model)											
RoBERTa†	-	66.3%	89.0%	90.2%	-	93.9%	91.9%	84.5%	95.3%	91.6%	-
BERT†	-	60.6%	86.6%	88.0%	-	92.3%	91.3%	70.4%	93.2%	90.0%	-
BERT‡	-	61.2%	86.6%	79.5%	-	93.1%	88.4%	68.9%	94.7%	89.6%	-

Table 1: Averaged development set results of all models. We perform 3 independent fine-tuning runs for each model and dataset. For comprehensive study, we also include previously reported results of large BERT models, although training details may differ in this and previous work. † indicate results from Liu et al. (2019) and ‡ are results from Wettig et al. (2022) using an efficient training recipe and 40% masking rate. Our baseline models are strong compared to previously reported results.

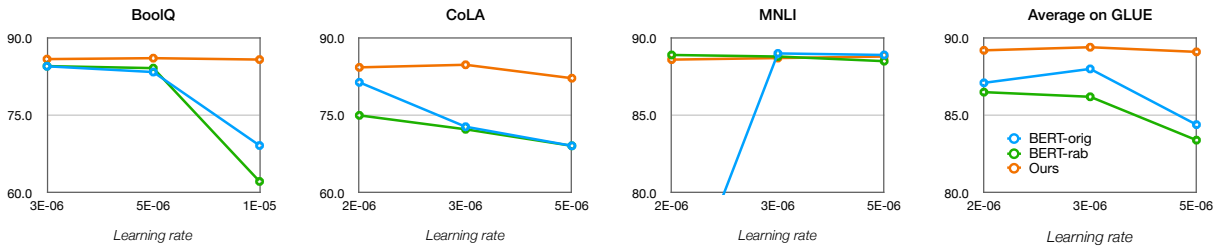

Figure 3: Stability of fine-tuning results given different learning rates. Results are averaged across 3 independent runs for each setting. Our model is more robust to the range of learning rates tested.

Our model with variable step sizes has a marginal 20% - 30% slow-down compared to the standard Transformer model when training on TPUs.

Note that SwishRNN can be made significantly faster using optimized implementation such as CUDA kernel fusion adopted in QRNN and SRU (Bradbury et al., 2017; Lei et al., 2018). We leave this for future work as custom kernel fusion is not readily available for TPUs.

3 Experimental Setup

Datasets Following BERT (Devlin et al., 2019), we evaluate all models by pre-training them with the masked language model (MLM) objective and then fine-tuning them on a wide range of downstream tasks. We use the Wikipedia and BookCorpus (Zhu et al., 2015) for pre-training, and 10 datasets from the GLUE (Wang et al., 2018) and SuperGLUE benchmark (Wang et al., 2019) including the BoolQ, CoLa, MNLI, MRPC, MultiRC, QNLI, QQP, RTE, SST2 and STS-B datasets.

Baselines We compare with two BERT variants. **BERT-orig** is the original BERT model using the multi-head attention described in Vaswani et al. (2017) and learned absolute positional encoding. The second variant **BERT-rab** adds the relative attention bias to each attention layer, following the T5 model (Raffel et al., 2020). Our model is the same as **BERT-rab** except we replace every FFN block with SwishRNN. The inner hidden size d' of SwishRNN blocks is decreased such that the total number of parameters are similar to the BERT baselines. Following Devlin et al. (2019), we experiment with two model sizes – a base model setting consists of 12 Transformer layers and a large model setting using 24 layers. The detailed model configurations are given in Appendix C.

Training Our pre-training recipe is similar to recent work (Liu et al., 2019; Izsak et al., 2021; Wettig et al., 2022). Specifically, we do not use the next sentence prediction objective and simply replace 15% input tokens with the special `[MASK]` token. We also use a larger batch size and fewer

Step size(s) of RNNs	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST2	STSB	Time
Base model (12 layers, $d = 768$)									
1	79.4%	85.9%	90.8%	92.1%	88.1%	68.0%	92.7%	86.3%	1.4×
2	75.5%	85.9%	88.6%	92.4%	88.1%	64.9%	92.1%	83.1%	1.2×
$\{1, 2, 4\}$	80.8%	85.9%	90.4%	92.5%	88.2%	70.8%	93.8%	85.7%	1.2×
Large model (24 layers, $d = 1024$)									
1	84.7%	89.0%	92.5%	94.5%	88.5%	83.9%	95.6%	87.9%	1.4×
$\{1, 2, 4\}$	84.8%	88.9%	92.9%	94.3%	88.7%	85.0%	95.3%	87.2%	1.3×

Table 2: Fine-tuning results on the GLUE datasets using different step sizes for the recurrent module in our model. We report averaged results and the pre-training time in relative to that of BERT-rab model. Using variable step sizes trains faster and obtains results on par with using step size 1.

training steps following recent work. Specifically, we use a batch size of 1024 for base models and 4096 for large models. The maximum number of pre-training steps is set to 300K.

To reduce the variance, we run 3 independent fine-tuning trials for every model and fine-tuning task, and report the averaged results. We also tune the learning rate separately for each model and fine-tuning task. The training details are provided in Appendix C.

4 Results

Overall results Table 1 presents the fine-tuning results on 10 datasets. Our base model achieves a substantial improvement, outperforming the BERT-orig and BERT-rab baselines with an average of 2.1 absolute points. The improvement is also consistent, as our base model is better on 9 out of the 10 datasets.

The improvement on large model setting is smaller. Our model obtains an increase of 0.6 point and is better on 6 datasets. We hypothesize that the increased modeling power due to recurrence can be saturating, as making the model much deeper and wider can already enhance the modeling capacity. The gains are still apparent on more challenging datasets such as BoolQ where the input sequences are much longer.

Stability One interesting observation in our experiments is that combining recurrence and attention improves fine-tuning stability. Figure 3 analyzes model stability by varying the learning rate. We showcase the results on the first 3 datasets (namely BoolQ, CoLA and MNLI) as well as the averaged results on 8 datasets in GLUE. For both BERT model variants, fine-tuning requires more careful tuning of the learning rate. In comparison, our model performs much more consistently across the learning rates tested.

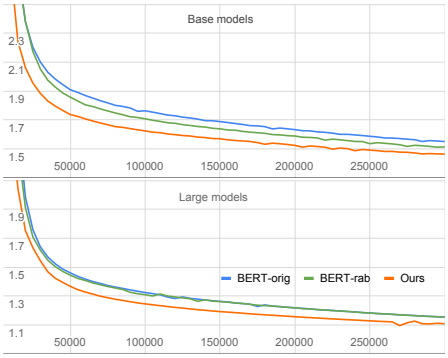

Figure 4: MLM pre-training loss of BERT-orig, BERT-rab and our model architectures.

Step size of RNN Table 2 shows the effect of changing the step size of SwishRNN. Using a step size of 1 is the slowest, since running smaller and more steps adds computational overhead. On the other hand, using a fixed step size of 2 reduces the training cost but hurts the fine-tuning results especially on the CoLA, MRPC, RTE and STS-B datasets. Our best model alternates the step size between 1, 2, and 4 across the recurrent layers, resulting in both faster training and stronger results.

Pre-training loss Figure 4 shows the training curves of all models during masked language model training. Our model achieves better loss, indicating increased modeling capacity.

5 Conclusion

In this work, we proposed incorporating an extremely simple recurrent module, SwishRNN, that when incorporated into BERT achieves consistent improvements without requiring low-level performance optimizations. Future directions include extending our work to encoder-decoder pretraining (Song et al., 2019) and exploring other domains such as protein modeling (Elnaggar et al., 2020).

References

Jeremy Appleyard, Tomas Kocisky, and Phil Blunsom. 2016. Optimizing performance of recurrent neural networks on gpus.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-Recurrent Neural Networks. *International Conference on Learning Representations (ICLR 2017)*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages 1877–1901.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George F. Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas B. Fehér, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. 2020. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *bioRxiv*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. 2019. Modeling recurrence for transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus).

Zhiheng Huang, Peng Xu, Davis Liang, Ajay Mishra, and Bing Xiang. 2020. Trans-blstm: Transformer with bidirectional lstm for language understanding.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to train BERT with an academic budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings*

of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

Tao Lei. 2021. When attention meets fast recurrence: Training language models with reduced compute. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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

Jing Pan, Tao Lei, Kwangyoun Kim, Kyu J. Han, and Shinji Watanabe. 2022. Sru++: Pioneering fast recurrence with attention for speech recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2018. Searching for activation functions.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. In *ICML*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7.

Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. 2022. Should you mask 15masked language modeling?

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. 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 (ICCV)*.

A Limitations

The model and results presented in this work have a few limitations. First, we focus our evaluation on BERT which is an encoder-only model. However, the proposed architecture can be incorporated into encoder-decoder models and decoder-only models as well, such as the T5 (Raffel et al., 2020), GPT-3 (Brown et al., 2020) and the recent PaLM model (Chowdhery et al., 2022). In addition, our fine-tuning datasets are primarily classification datasets, and contain fewer training examples compared to other generation tasks such as machine translation and summarization. As future work, our model architecture can be extended to other types of language models and language generation tasks.

The training of language models using large-scale web text is computationally expensive and can capture various biases associated with the data. We performed speed optimization that improves the computational efficiency of our model, as described in Section 2. We believe more research on reducing data biases and computational cost of large language models are important.

B Transformer Architecture

For completeness we review the F_{att}, F_{ffn}, and $F_{\text{Add+Norm}}$ blocks used in the Transformer architecture (Figure 1). We omit all bias terms for simplicity.

Attention block (F_{att}) Multi-headed attention with h heads first calculates query $\boldsymbol{Q}_m$, key $\boldsymbol{K}_m$, and value $\boldsymbol{V}_m$ matrices for each head $m \in \{1, ..h\}$ by applying linear transformations to the input:

$$\boldsymbol{Q}_m = \boldsymbol{X}\boldsymbol{W}_m^Q,\ \boldsymbol{K}_m = \boldsymbol{X}\boldsymbol{W}_m^K,\ \boldsymbol{V}_m = \boldsymbol{X}\boldsymbol{W}_m^V$$

Each transformation matrix $\boldsymbol{W}_m^Q, \boldsymbol{W}_m^K, \boldsymbol{W}_m^V$ is of dimension $d \times d_h$ where $d_h = d/h$. Attention vectors are then computed for each head, concatenated and multiplied by a linear transformation $\boldsymbol{W}^O$ of dimension $d \times d$:

$$\boldsymbol{Z}_m = \text{softmax}\left(\frac{\boldsymbol{Q}_m\boldsymbol{K}_m^\top}{\sqrt{d_h}}\right)\boldsymbol{V}_m$$

$$F_{\text{att}}(\boldsymbol{X}) = \text{Concat}([\boldsymbol{Z}_1, ..., \boldsymbol{Z}_h])\boldsymbol{W}^O$$

Feed forward block (F_{ffn}) Following BERT, we use a GeLU nonlinearity (Hendrycks and Gimpel, 2016) i.e. $F_{\text{ffn}}(\boldsymbol{X}) = \boldsymbol{W}_{f2}(\text{GeLU}\,(\boldsymbol{W}_{f1}\boldsymbol{X}))$.

Residual connection and layer normalization ($F_{\text{Add+Norm}}$) . This block applies layer normalization (Ba et al., 2016) to the addition of the two inputs: $F_{\text{Add+Norm}}(\tilde{\boldsymbol{X}}, \boldsymbol{X}) = \text{LayerNorm}(\tilde{\boldsymbol{X}} + \boldsymbol{X})$.

C Training details

Pre-training The detailed hyper-parameter configuration for BERT training is shown in Table 3. The training recipe is based on previous works such as RoBERTa (Liu et al., 2019) and the 24-hour BERT (Izsak et al., 2021). Specifically, compared to the original BERT training recipe which uses 1M training steps and a batch size of 256, the new recipe increases the batch size. The models are trained with much fewer steps and a larger learning rate as a result, which reduces the overall training time. We train base models using 16 TPU v4 chips and large models using 256 chips. For fine-tuning we use only 1 or 2 TPU v4 chips respectively.

Fine-tuning We fine tune our pretrained models on 10 tasks including BoolQ (Clark et al., 2019), CoLA (Warstadt et al., 2019), MNLI (Williams et al., 2018), MRPC (Dolan and Brockett, 2005), MultiRC (Khashabi et al., 2018), QNLI (Rajpurkar et al., 2016), QQP[4], RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), SST-2 (Socher et al., 2013) and STS-B (Cer et al., 2017).

We use a batch size of 32 for fine-tuning and evaluate the model performance every 1000 steps. We use Adam optimizer without weight decay during fine-tuning. We use a fixed learning rate tuned among {1e-5, 5e-6, 3e-6, 2e-6} and warm up the learning rate for 1000 steps. The maximum number of training steps of each dataset is presented in Table 4. We set the number proportionally to the size of the dataset and do not tune it in our experiments.

[4] https://quoradata.quora.com/

	Base model	Large model
Number of layers	12	24
Hidden size	768	1024
Inner hidden size – FFN	3072	4096
Inner hidden size – SwishRNN	2048	2752
Attention heads	12	16
Attention head size	64	64
Dropout	0.1	0.1
Attention dropout	0.1	0.1
Learning rate	0.0003	0.0002
Learning rate warmup steps	20,000	20,000
Learning rate decay	Linear	Linear
Adam β_1	0.9	0.9
Adam β_2	0.98	0.98
Weight decay	0.01	0.01
Batch size	1024	4096
Sequence length	512	512
Training steps	300,000	300,000

Table 3: Hyper-parameters for pre-training the base models and large models in our experiments.

BoolQ	CoLA	MNLI	MRPC	MultiRC	QNLI	QQP	RTE	SST2	STSB
50K	50K	200K	20K	50K	100K	150K	20K	80K	30K

Table 4: Maximum number of fine-tuning step used for each dataset in our experiments.