Skip Transformers: Efficient Inference through Skip-Routing

Matthew Peroni
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02142
mperoni1@mit.edu

Dimitris Bertsimas
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142
dbertsim@mit.edu

Abstract

As the scale of Transformer-based language models continues to increase, there is a growing need for methodological improvements in training and inference efficiency. Recent developments, such as IA3 and LoRA, have successfully addressed training efficiency for fine-tuning, but not inference efficiency. Inspired by recent work in Sparse Mixture of Experts and conditional computation in neural networks, we propose Skip Transformers, which modify the standard architecture by adding routers after each self-attention block in the Transformer architecture that decide whether to route each token embedding to the corresponding feed-forward neural network (FFN), or to skip the FFN and pass the existing embedding through to the next attention block. We refer to this process as skip-routing. Using a new set of penalty terms in the loss function and a specific router weight initialization scheme, we demonstrate empirically that adapting the Transformer architecture with skip-routing during fine-tuning can improve computational efficiency at inference while maintaining or improving performance on downstream tasks. These results, although preliminary, establish and motivate an exciting new direction for developing sparsely-activated Transformer models that improve model performance and inference efficiency.

1 Introduction

Growing empirical evidence has affirmed the scaling laws of Transformer-based language models [Hoffmann et al., Kaplan et al., Vaswani et al.], leading to the proliferation of increasingly large, densely-activated models [Brown et al., Wei et al., Touvron et al.]. In response to this trend, there has been increasing interest and development in the area of Sparse Mixture of Experts (SMoE) [Shazeer et al., Lepikhin et al., Fedus et al.], which aims to reduce the effective parameter count and required compute resources at inference by relying on sparse network activation. Specifically, in the context of Transformers, SMoE typically refers to widening the network by adding multiple feed-forward neural networks (FFNs) at each attention block and then selecting a subset of these FFNs to process each token embedding. In their work, Fedus et al. demonstrate empirically that routing can be learned over a collection of FFN "experts" while only passing each token embedding through one expert at a time. In this work, we investigate a special case of expert routing where, at each layer, a router decides whether to pass each token embedding through the corresponding FFN or skip the FFN and preserve the current token embedding. Importantly, skipping the FFNs yields a significant decrease in the effective number of parameters during inference. In this work, we provide preliminary evidence that this skip-routing can improve efficiency at inference while maintaining or improving performance on downstream tasks. While efficient fine-tuning has been addressed recently with methods such as IA3 [Liu et al., a] and LoRA [Hu et al.], these methods do not improve inference efficiency.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

Although related to SMoE literature, we focus on the conditional computation aspect of this problem. By conditional computation, we refer to the input-dependent, selective activation of neurons in a neural network. In our implementation and experiments, we only consider one FFN "expert" per attention layer and the choice between passing the token embeddings through this FFN or skipping it. There is a natural extension to the case where there are multiple FFN experts, which we plan to explore in future work. Techniques for conditional computation in neural networks, such as those introduced in Bengio et al. [a], Davis and Arel, Bengio et al. [b], have demonstrated practical success in increasing efficiency while maintaining model performance. Unlike standard pruning and quantization methods [Liang et al.], conditional computation only reduces the effective parameter count, while increasing the total parameter count, which we hypothesize makes this approach more likely to improve efficiency at inference while maintaining or improving model performance.

There has been some previous work applying these conditional computation methods to Transformers to improve efficiency at inference. For example, Hou et al. introduced the idea of filtering out "unimportant" tokens as the input sequence passes through the attention layers. Such approaches tend to struggle with longer sequence lengths, as this filtering process can eliminate important long-range context. Further, this method always applies the same, static intermediate layer skipping, and only considers global (cumulative loss) or static (term frequency) rules for determining token importance. Other work, such as Dehghani et al. and Liu et al. [b] propose halting algorithms that process each token through a variable number of attention blocks. These algorithms limit computation by selecting a contiguous sequence of layers to skip. In contrast, our dynamic routing approach allows tokens to be processed by any subsequence of FFN experts, routing the token embeddings at each layer depending on the contextual representation of that token following the previous self-attention layer.

Recent work by Zeng et al. and Raposo et al. have also explored a similar idea about dynamic skip-routing. In contrast to our work, both papers focus on pretraining transformer models and assume each attention block has a fixed budget or "capacity" for the number of tokens it can process. While this approach does provide a guarantee regarding the total load for each attention block, early results from our work suggest that allowing the model to learn variations in the load distributions across layers can improve performance on downstream tasks while achieving the same effective parameter count. In addition, both papers consider a deterministic routing scheme, unlike our probabilistic approach, which lessens the ability of these methods to serve as a form of regularization. The architecture proposed by Raposo et al. skips both the attention layers and the FFNs, while our work focuses exclusively on skipping the FFNs. By skipping the attention layers, the tokens that are not skipped will only be modified by the other tokens that were also selected to be processed by the attention mechanism. Similar to Hou et al., such an approach can lead to important context being ignored in longer sequences. Since the FFNs are applied independently to each token, our approach does not risk losing important context, while still addressing the most compute-intensive mechanism in the transformer architecture.

In Section 2 we introduce our methodology for skip-routing, the corresponding Skip Transformer architecture, and the related penalty terms and router weight initialization scheme. We then present and discuss our preliminary experimental results in Section 3. The primary contributions of this work are 1) introducing the Skip Transformer architecture and empirically validating its ability to improve performance and efficiency at inference and 2) establishing an effective set of penalty terms and weight initialization that yields improved performance for the Skip Transformer.

2 Skip Routing

At a high level, our approach consists of adding a router $r_i : \mathbb{R}^d \to [0, 1]$ to each layer $i \in [N_l]$ that will be applied to each token embedding $\mathbf{x}_{ij} \in \mathbb{R}^d$ corresponding to token $j \in [N_T]$ separately to determine which tokens should be passed through the layer's FFN and which should be skipped. To minimize the increase in total parameter count, we focus on routers that are normalized linear layers with sigmoid activation. Formally, we define the routing functions as

$$r_i(\mathbf{x}_{ij}; \mathbf{w}_i, \tau_i) = \sigma\left(\tau_i \cdot \frac{\mathbf{w}_i^T \mathbf{x}_{ij}}{||\mathbf{w}_i|| \cdot ||\mathbf{x}_{ij}||} + \beta_i\right), \tag{1}$$

where $\mathbf{w}_i \in \mathbb{R}^d$ and $\tau_i, \beta_i \in \mathbb{R}$ are learnable parameters, and σ is the sigmoid function. The decision to use a normalized linear layer was inspired by Chi et al., who found that this helps avoid representation collapse in similar SMoE systems. To determine which tokens are skipped, we sample

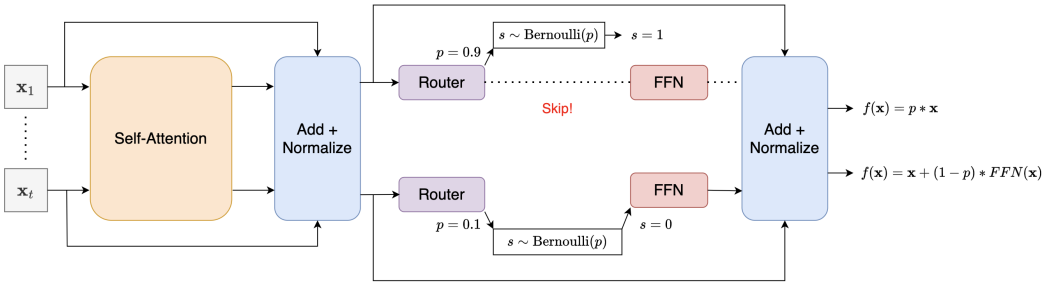


Figure 1: Overview of Skip Transformer Architecture.

from the Bernoulli distribution according to the outputs from the router. For token j at router i, we define the skip mask s_i as

$$s_i(\mathbf{x}_{ij}) \sim \text{Bernoulli}(r_i(\mathbf{x}_{ij}; \mathbf{w}_i, \tau_i)).$$

This encourages exploration and adds a form of regularization similar to dropout, except that the dropout in this case is input-dependent. For a given token embedding $\mathbf{x}$ and feed-forward net at layer i, denoted by FFN_i, we define the output of the layer as

$$f(\mathbf{x}) = \begin{cases} \mathbf{x} + (1 - r_i(\mathbf{x})) \cdot FFN_i(\mathbf{x}), & s_i(\mathbf{x}) = 0 \\ r_i(\mathbf{x}) \cdot \mathbf{x}, & s_i(\mathbf{x}) = 1. \end{cases}$$

Notice that, unlike traditional SMoE approaches, we are choosing between using an expert or not, so we must define what happens when an expert is skipped. While we could pass through the input embedding, $\mathbf{x}$, as is, this would not propagate information about the router through the computation graph. Instead, we scale $\mathbf{x}$ by the router output, such that when $r_i(\mathbf{x}) = 1$, which notionally corresponds to the router being fully confident that the FFN should be skipped, the input embedding is passed through unchanged. We provide the pseudocode for the core operations of the router and FFN in Algorithm 1.

Unlike SMoE approaches which encourage tokens to be load-balanced across the FFN experts, we want to encourage tokens to skip the FFNs at each layer. Additionally, we want to encourage FFNs to be skipped for any input sequence, incentivizing the model to create specialized FFNs for different types of inputs. Suppose we have a batch size N_B, fixed sequence length N_T, and N_l layers. To incentivize this behavior, we add the following penalty terms to the loss function,

$$l_s(X; \alpha_s, s_r) = \frac{\alpha_s}{N_l} \sum_{j \in N_l} \left(\frac{1}{N_B \cdot N_T} \sum_{k \in N_B} \sum_{i \in N_T} r_j(\mathbf{x}_{ijk}) - s_r \right)^2, \tag{2}$$

$$l_b(X; \alpha_b, s_r) = \frac{\alpha_b}{N_b} \sum_{k \in N_B} \left(\frac{1}{N_l \cdot N_T} \sum_{j \in N_l} \sum_{i \in N_T} r_j(\mathbf{x}_{ijk}) - s_r \right)^2, \tag{3}$$

where $\alpha_s, \alpha_b \in \mathbb{R}$ are hyper-parameters to weight the relative importance of skipping the FFNs and $s_r \in [0, 1]$ is the target skip-rate. The first term, l_s, encourages each router to have an expected skip-rate of s_r while the second term, l_b, encourages the routers to have a constant skip-rate across samples. By encouraging a constant skip-rate across samples, the model's computational requirements at inference also becomes more stable. To discourage the routers from learning a constant output $r_i(\cdot) = s_r$, which would minimize l_s and l_r, we further add a penalty to incentivize variance in the router output,

$$l_v(X; \alpha_v) = -\frac{\alpha_v}{N_l} \sum_{j \in N_l} \frac{1}{N_B \cdot N_T} \sum_{i \in N_T} \sum_{k \in N_B} \left(r_j(\mathbf{x}_{ijk}) - \frac{1}{N_B \cdot N_T} \sum_{i \in N_T} \sum_{k \in N_B} r_j(\mathbf{x}_{ijk}) \right)^2. \tag{4}$$

Note that other, related penalty terms could be applied, which amount to rearranging the dimensions along which we average the router outputs. In practice, we have found that this combination of penalty terms leads to the best results in terms of performance and stability.

Router Initialization. In preliminary experiments, we observed that the initialization of router weights, specifically the bias term, significantly impacts model performance. The router weights $\mathbf{w}_i$

are sampled from the Kaiming Uniform distribution, and we set $\tau_i = 1$. Then, critically, for a fixed target skip-rate s_r, we set $\beta_i = \sigma^{-1}(s_r)$, where $\sigma^{-1}(s_r) = \log(s_r) - \log(1 - s_r)$ is the inverse of the sigmoid function. To justify this, let us assume the token embeddings are centered such that $E[\mathbf{x}] = \mathbf{0}$. Then, given that $\mathbf{w}_i$ are sampled from a uniform distribution with $E[\mathbf{w}_i] = \mathbf{0}$, since $\mathbf{w}_i$ and $\mathbf{x}$ are independent at initialization, we have $E[\mathbf{w}_i^T \mathbf{x}] = \mathbf{0}$. Therefore, at initialization, we have

$$E_{\mathbf{w}_i, \mathbf{x}}\left[\frac{\mathbf{w}_i^T \mathbf{x}}{||\mathbf{w}_i|| \cdot ||\mathbf{x}||} + \beta_i\right] = \sigma^{-1}(s_r).$$

Since the sigmoid function is nonlinear, $E[r_i(\mathbf{x}; \mathbf{w}_i, \tau_i)] \neq \sigma(E[\frac{\mathbf{w}_i^T \mathbf{x}}{||\mathbf{w}_i|| \cdot ||\mathbf{x}||} + \beta_i])$. However, the sigmoid function is convex for $x < 0$, corresponding to $s_r < 0.5$, in which case Jensen's inequality tells us that $E[r_i(\mathbf{x}; \mathbf{w}_i, \tau_i)] \leq \sigma(\sigma^- 1(s_r)) = s_r$, such that initializing the router in this way likely provides an upper bound on the initial skip-rate. Empirically, we find that with this initialization technique, $E[r_i(\mathbf{x}; \mathbf{w}_i, \tau_i)] \approx s_r$. In our experiments, initializing the expected router output close to the target skip rate yields the best model performance by a significant margin.

Computational Efficiency. To estimate the skip-rate necessary to improve the computational efficiency of the model, we can reason about the expected number of floating point operations (FLOPs) required for a forward pass of the FFN, and compare that to the skip-router approach. Matrix multiplication dominates the cost of computing the forward pass of a FFN. In the case of a single hidden layer FFN with input dimension d, hidden dimension h, and output dimension d, for a single vector input $\mathbf{x} \in \mathbb{R}^d$, the number of operations required for the matrix-vector multiplication is $4hd - 2d$. Similarly, the vector-vector multiplication required for the router, represented by a single linear layer $r_i : \mathbb{R}^d \to \mathbb{R}$, is $2d - 1$. Since we are in a probabilistic setting, skipping FFNs by sampling from a Bernoulli distribution, we can reason about the expected number of operations for each router and FFN pair. Assuming the probability of skipping the FFN is s, the expected FLOPs for the skip layer is given by

$$s \cdot (2d - 1) + (1 - s) \cdot (4hd - 1).$$

The first term corresponds to the case where the FFN is skipped, and we only incur the cost of the router. The second term corresponds to the case where the FFN is used, and we incur the cost of both the router and the FFN. The expected difference in FLOPs between the standard FFN and the skip layer is then

$$4hd - 2d - (s \cdot (2d - 1) + (1 - s) \cdot (4hd - 1)) = 2sd(2h - 1) - 2d + 1.$$

With this, we can find the skip-rate s necessary for the skip-layer to be more efficient than the standard FFN in expectation to be

$$s > \frac{2d - 1}{4hd - 2d}.$$

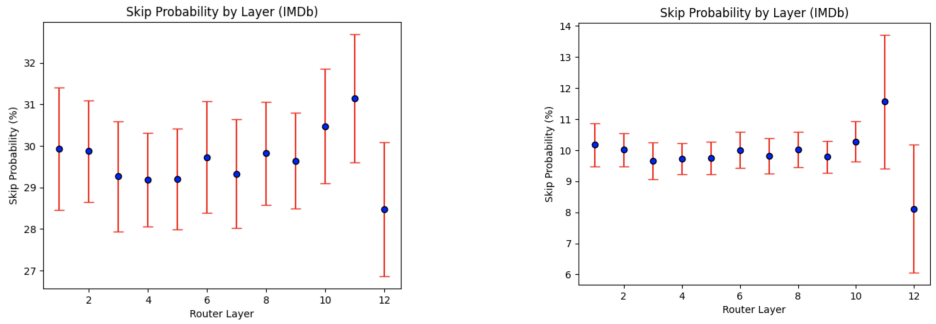
We can simplify the expression by adding one to the numerator, creating the marginally looser sufficient condition $s > \frac{1}{2h-1}$. Therefore, the skip-rate required for the skip layer to be more efficient than the baseline architecture is inversely proportional to the hidden dimension of the expert FFNs. For example, BERT-base uses $h = 3072$, so we can estimate that as long as $s > \frac{1}{6143} \approx 0.0002$, corresponding to a 0.02% skip-rate, the skip-transformer will improve efficiency. Therefore, even for relatively small transformer models, the skip rate only needs to be marginal to improve efficiency, due to the high dimensionality of the FFN hidden layer. Using our results from our IMDb experiment in Section 3, suppose $s = 0.10$, the sequence length is 512, and there are 12 router-FFN blocks. Then the Skip Transformer would save approximately 6 GFLOPs at inference time per input sequence.

3 Experimental Results

For our experiments, we use the original pretrained BERT model, BERT-base, as introduced in Devlin et al., to initialize the weights of our models. All weights, including the FFNs, are initialized from the pretrained BERT-base model. For BERT-base, we have $d = 768$, $h = 3072$, and $N_l = 12$, such that there are 56 million parameters for the FFNs. The max sequence length is $N_T = 512$ and during training, when processing a batch of sequences, we apply the attention mask to the router such that all padding tokens are excluded from being considered by the router and in the corresponding skip statistics. To initialize router weights, $\mathbf{w}_i$ is sampled from the Kaiming Uniform distribution,

Model	IMDb	Emotion	CoLA	MRPC
BERT-Base	93.4	93.0	81.7	85.5
Skip-BERT ($s_r = 0.3$)	93.1 ± 0.2	93.1 ± 0.2	79.9 ± 0.3	84.1 ± 0.2
Skip-BERT ($s_r = 0.1$)	93.7 ± 0.1	$\mathbf{93.4 \pm 0.1}$	82.2 ± 0.1	$\mathbf{87.3 \pm 0.1}$
Skip-BERT ($s_r = 0.05$)	$\mathbf{94.0 \pm 0.2}$	92.9 ± 0.3	$\mathbf{83.6 \pm 0.2}$	86.7 ± 0.1
Skip-BERT ($\alpha_v = 0$)	93.1 ± 0.1	92.5 ± 0.1	83.0 ± 0.1	85.7 ± 0.1

Table 1: Out-of-sample accuracy for the Skip-BERT models across downstream tasks, varying skip-rates (s_r), with and without the l_v penalty. Accuracy for CoLA and MRPC reported for the Dev validation dataset.



(a) $s_r = 0.3$ (b) $s_r = 0.1$

Figure 2: Skip-rate per router for the IMDb dataset. The blue dots represent the mean router output over the test set and the bars indicate two standard deviations.

$\tau_i = 1$, and $\beta_i = \sigma^{-1}(s_r)$ for all $i \in [N_l]$. We set $\alpha_s = \alpha_b = \alpha_v = 1$ for our experiments. In our experiments, we train the entire model and do not freeze any weights. We refer to these Skip Transformer models as Skip-BERT. We benchmark the Skip-BERT models with varying target skip-rates, s_r, against the corresponding BERT-base model, fine-tuned using the Adam optimizer [Kingma and Ba] for both and matching the initial learning rates for each experiment. Both the Skip-BERT and BERT-base model are trained for five epochs for each experiment, and we use a batch size of 32 for all experiments. We do not apply a weight decay penalty for either Skip-BERT or BERT-base. We use this encoder-only architecture to fine-tune for contextual natural language understanding classification tasks.

We fine-tune the Skip-BERT models on the IMDb sentiment classification dataset, introduced by Maas et al. [2011], which is a binary classification task, split into training and evaluation datasets of size $n = 25k$. We also fine-tune our models on the Emotion Recognition dataset introduced by Saravia et al. [2018], which is a collection of $n = 20k$ English Twitter messages labeled with 6 different emotions (anger, fear, joy, love, sadness, and surprise). The dataset is split $80/10/10$ into training, validation, and test sets. We also evaluate our skip transformer architecture on several tasks from the GLUE Benchmark [Wang et al.], specifically the CoLA and MRPC datasets, which cover paraphrasing and grammar tasks. As this work is preliminary and under development, for the GLUE tasks, we report accuracy on the Dev validation datasets. We summarize our results in Table 1.

We observe that across tasks, the Skip-BERT models match or outperform the BERT-base model. In general, as expected, decreasing the skip-rate s_r corresponds to an increase in model performance. For the CoLA task, we observe that a skip-rate around $s_r = 0.05$ actually improves model performance significantly. However, similar to traditional dropout, there seems to be an optimal skip-rate for some tasks, below which performance begins to decrease again. For example, we observed for the Emotion dataset that $s_r = 0.1$ seems to be about optimal, and dropping below to $s_r = 0.05$ decreases model performance. In Figure 2 we provide a visualization of the mean and variance of the skip rate across routers for the IMDb dataset. When $\alpha_v = 1$, as in our experiments, the variance of the router output is relatively small, especially for the first ten routers of the Skip-BERT model for IMDb with $s_r = 0.1$. Interestingly, the variance and average skip-rate seem to shift significantly for the last two routers, suggesting that the model has learned a uniform routing distribution for the earlier layers, then focuses on the specific inputs and increases the variance in the skip-rate for the last few layers.

The skip-rate consistently drops at the final layer across tasks and skip-rates, suggesting that it is often important to transform the token embeddings right before the final classification layer. The last row of 1 gives the model accuracy when the best skip-rate is selected and we set $\alpha_v = 0$, removing the variance penalty. It can be seen that removing this penalty term negatively impacts performance on downstream tasks, demonstrating the importance of encouraging variance in the router outputs. In future work, we plan to investigate the impact of encouraging variance in the router outputs between layers, to explore whether varying "budgets" for each FFN leads to better performance.

4 Conclusion

In this work, we provide early empirical evidence that conditional computation in Transformers via skip-routing can maintain or even improve model performance on downstream tasks while improving efficiency at inference time. We introduce an effective incentive scheme through new penalty terms in the loss function and router weight initialization, significantly impacting model performance. This work integrates well with related developments in SMoEs, as one could extend this work to incorporate routing between multiple FFN experts, or skipping them all. While the focus of this paper is on fine-tuning, an exciting future direction would be to use the Skip Transformer architecture for pre-training as well, potentially leading to further, more generalizable skip-routing. The Skip Transformer, as presented in this work, provides an exciting new direction for sparsely-activate Transformer models that have the potential to improve inference efficiency while preserving model performance.

Acknowledgments and Disclosure of Funding

Use unnumbered first level headings for the acknowledgments. All acknowledgments go at the end of the paper before the list of references. Moreover, you are required to declare funding (financial activities supporting the submitted work) and competing interests (related financial activities outside the submitted work). More information about this disclosure can be found at: `https://neurips.cc/Conferences/2024/PaperInformation/FundingDisclosure`.

Do **not** include this section in the anonymized submission, only in the final paper. You can use the `ack` environment provided in the style file to autmoatically hide this section in the anonymized submission.

References

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models, a. URL `http://arxiv.org/abs/1511.06297`.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, b. URL `http://arxiv.org/abs/1308.3432`.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. URL `http://arxiv.org/abs/2005.14165`.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, Heyan Huang, and Furu Wei. On the representation collapse of sparse mixture of experts. 35:34600–34613. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/df4f371f1f89ec8ba5014b3310578048-Abstract-Conference.html`.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. URL `http://arxiv.org/abs/1312.4461`.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. URL http://arxiv.org/abs/1810.04805.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. 23(120):1–39. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-0998.html.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. URL http://arxiv.org/abs/2203.15556.

Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. Token dropping for efficient BERT pretraining. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3774–3784. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.262. URL https://aclanthology.org/2022.acl-long.262.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. URL https://arxiv.org/abs/2106.09685v2.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. URL http://arxiv.org/abs/2001.08361.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. URL http://arxiv.org/abs/1412.6980.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. 461:370–403. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.07.045. URL https://www.sciencedirect.com/science/article/pii/S0925231221010894.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, a. URL https://arxiv.org/abs/2205.05638v2.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. Faster depth-adaptive transformers. 35(15):13424–13432, b. ISSN 2374-3468. doi: 10.1609/aaai.v35i15.17584. URL https://ojs.aaai.org/index.php/AAAI/article/view/17584. Number: 15.

Andrew L. 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*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. URL http://arxiv.org/abs/2404.02258.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. CARER: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3687–3697, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1404. URL https://www.aclweb.org/anthology/D18-1404.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. URL http://arxiv.org/abs/1701.06538.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. URL http://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. URL http://arxiv.org/abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pages 353–355. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL http://aclweb.org/anthology/W18-5446.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. URL http://arxiv.org/abs/2109.01652.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. Learning to skip for language modeling. URL http://arxiv.org/abs/2311.15436.

Supplementary Material

Algorithm 1 Encoder Skip-Layer Pseudocode (Pytorch)

Input: hidden_states, attention_mask
attention_output ← self_attention(hidden_states, attention_mask)
router_output ← router(attention_output)
skip_mask ← bernoulli(router_output)
ffn_indices ← skip_mask == 0.
intermediate_states ← dense_in(attention_output[ffn_indices])
intermediate_states ← intermediate_act_fn(intermediate_states)
output_states ← dense_out(intermediate_states)
attention_output[ffn_indices] ← output_states ∗ (1 - router_output[ffn_indices]) + attention_output[ffn_indices]
attention_output[∼ffn_indices] ← router_output[∼ffn_indices] * attention_output[∼ffn_indices]
attention_output ← LayerNorm(attention_output)
return attention_output