Token Dropping for Efficient BERT Pretraining

Le Hou[1*]   **Richard Yuanzhe Pang**[2§*]   **Tianyi Zhou**[13§]   **Yuexin Wu**[1]   **Xinying Song**[1]
Xiaodan Song[1]   **Denny Zhou**[1]

[1] Google   [2] New York University   [3] University of Maryland, College Park
`lehou@google.com, yzpang@nyu.edu`

Abstract

Transformer-based models generally allocate the same amount of computation for each token in a given sequence. We develop a simple but effective "token dropping" method to accelerate the pretraining of transformer models, such as BERT, without degrading its performance on downstream tasks. In particular, we drop unimportant tokens starting from an intermediate layer in the model to make the model focus on important tokens more efficiently if with limited computational resource. The dropped tokens are later picked up by the last layer of the model so that the model still produces full-length sequences. We leverage the already built-in masked language modeling (MLM) loss to identify unimportant tokens with practically no computational overhead. In our experiments, this simple approach reduces the pretraining cost of BERT by 25% while achieving similar overall fine-tuning performance on standard downstream tasks.

1 Introduction

Nowadays, the success of neural networks in a variety of NLP tasks heavily relies on BERT-type language models containing millions to billions of parameters. However, the pretraining process of these models is computationally expensive, generating significant carbon dioxide emission (Strubell et al., 2019; Patterson et al., 2021). In practice, there is the need to perform large-scale language model pretraining for diverse applications (Lee et al., 2020; Chalkidis et al., 2020; Zou et al., 2021; Rogers et al., 2020) in different languages (Antoun et al., 2020; Sun et al., 2021). In this paper, we develop a technique that significantly reduces the pretraining cost of BERT models (Devlin et al., 2019) without hurting their test performance on a diverse set of fine-tuning tasks.

* Equal contribution.
§ Work done at Google Brain.

Recent efforts of efficient training involve mixed-precision training (Shoeybi et al., 2019), distributed training (You et al., 2020), better modeling on rare words and phrases (Wu et al., 2021), designing more effective and data-efficient pretraining objectives (Lan et al., 2020; Clark et al., 2020; Raffel et al., 2020), progressive stacking (Gong et al., 2019), and so on. While these approaches contribute to efficient training with reduced computational cost, most of them focus on the model architecture or the optimization process.

In this paper, we focus on a simple but efficient BERT-pretraining strategy that has been underexplored before, i.e., "token dropping," which removes the redundant tokens in each sequence that are less informative to training. Since not all tokens contribute equally to the output or the training objective, and the computational complexity of transformer-based models grows at least linearly with respect to the sequence length, shortening the input sequences can accelerate the training effectively.

Among existing studies, the depth-adaptive transformer approach aims to reduce the autoregressive inference time by allocating less computation on easy-to-predict tokens (Elbayad et al., 2020). To improve the training efficiency, Dai et al. (2020) perform pooling on the embeddings of nearby tokens. However, directly dropping tokens during pretraining was not studied until very recently in faster depth adaptive transformer (Liu et al., 2021), where the important tokens are identified either through (1) mutual information-based estimation between tokens and predefined labels or through (2) a separate BERT model that exhaustively computes the masked language model (MLM) loss for each token. On the contrary, we focus on accelerating the task-agnostic pretraining phase without requiring any labels or any computation by a separate language model. Specifically, we identify important tokens as the ones hard to predict by the model

3774

itself through its loss during training, which is adaptive to its training process and leads to practically no computational overhead. We show examples of dropped tokens in Figure 1.

Recent approaches such as RoBERTa (Liu et al., 2019) suggest packing input sequences. In this way, there are no `[PAD]` tokens, which makes it a non-trivial task to identify unimportant tokens. We identify unimportant tokens in each sequence with the smallest historical MLM loss (we take the running average of the MLM loss of each token throughout the pretraining process). By removing them from intermediate layers of a BERT model during training, we save an enormous amount of computation and memory. We keep them in the first several layers as well as in the last layer so that they are still present in the model. Therefore, the inputs and outputs of BERT model are kept consistent with the conventional all-token training process. Without modifying the original BERT architecture or training setting, this simple token-dropping strategy trains intermediate layers mainly on a few important tokens. As demonstrated in our experiments, models pretrained in this way generalize well on diverse downstream tasks with full sequences.

To summarize, our contributions are as follows. (1) We show that BERT models can be pretrained with only a subset of the layers focusing on important tokens. Even though the model is trained on sub-sequences of important tokens only, it generalizes well to full sequences during fine-tuning on downstream tasks. (2) We identify important tokens through the pretraining process by exploring the training dynamics, with minimal computational overhead and without modifying the model architecture. (3) We show that our token dropping strategy can save 25% of pretraining time while achieving similar performance on downstream tasks. Code is available at https://github.com/tensorflow/models/tree/master/official/projects/token_dropping.

2 Prerequisites

2.1 Sequence Packing

Recall that a sequence in BERT consists of two sentences as well as the "classification" token `[CLS]` and the "separator" token `[SEP]`. If the resulting number of tokens is smaller than 512, then padding tokens are added to ensure that each sequence is

Does(1.1) the(0.3) **experiment**(3.2) **look**(1.3) **good**(2.10) ?(0.6)

They(1.0) **smell**(2.1) something(1.8) **horribly**(5.4) **wrong**(2.3) .(0.2)

Saturn(2.9) is(0.4) the(0.3) **sixth**(3.0) **planet**(2.4) **from**(1.0) the(0.3) **sun**(1.9) and(0.6) the(0.3) **second**(1.8) -(0.8) **largest**(1.4) in(0.5) the(0.3) **solar**(2.3) **system**(2.2) ,(0.4) **after**(1.5) **jupiter**(3.3) .(0.2) It(0.7) is(0.4) a(0.5) **gas**(2.2) **giant**(4.0) **with**(0.9) an(0.8) **average**(1.2) **radius**(3.3) of(0.3) **about**(1.3) **nine**(3.5) and(0.6) a(0.5) **half**(2.0) **times**(1.6) that(0.8) of(0.3) **earth**(2.4) .(0.2)

Figure 1: Randomly selected example sentences with actual importance scores (cumulative losses, to be discussed in Section 3.3) from our model. For efficient pretraining, tokens in bold are preserved in every BERT encoder layer, whereas other tokens are dropped for certain layers.

exactly 512-token long.

We decide to use sequence packing (Liu et al., 2019) so that there would be no `[PAD]` symbols, throughout the paper. We also remove the next-sentence prediction training criteria as well. The rationale for using sequence-packing is two-fold. First, sequence packing provides a competitive baseline in terms of pretraining efficiency (So et al., 2019; Liu et al., 2019; Kosec et al., 2021; Zhang et al., 2021). Second, using sequence-packing can stress-test our algorithm under the absence of padding symbols to see if it brings further improvements beyond dropping padding tokens: without sequence packing, our algorithm can label `[PAD]` as the unimportant tokens, which trivially improves pretraining efficiency; with sequence packing, however, our algorithm has to identify and drop real tokens as unimportant tokens to improve the efficiency.

2.2 Multi-Head Attention

Define T to be the input sequence length and d_k, d_v to be the size of each individual key and value vector respectively. The multi-head attention function with h attention heads is defined as:

$$\text{MultiHeadAttention}(Q, K, V) = \text{concat}(H_1, \ldots, H_h)W^O,$$

where

$$H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$
$$= \text{softmax}\left(\frac{(QW_i^Q)(KW_i^K)^\top}{\sqrt{d_k}}\right)VW_i^V.$$

Use d_{model} to denote the hidden size of the model (usually equal to hd_k). We have the following:

$$Q, K, V \in \mathbb{R}^{T \times d_{\text{model}}},$$
$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k},$$
$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \quad W_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}.$$

2.3 Feed-Forward Networks

Besides the attention sub-layer, each BERT encoder layer also contains the feed-forward sub-layer (or "feed-forward network," abbreviated as FFN). Each FFN is a position-wise function: it is applied to each position of the input, identically.

The input to FFN is a matrix $\in \mathbb{R}^{T \times d_{\text{model}}}$. The input will be transformed by a two-layer perceptron (with ReLU activation in between) into an output matrix $\in \mathbb{R}^{T \times d_o}$. In Vaswani et al. (2017), $d_o = d_{\text{model}}$ and the hidden size of the intermediate layer $d_{ff} = 4 \cdot d_{\text{model}}$ due to empirical investigation.

3 Token-Dropping

Suppose the input sequence contains 512 tokens. Having 512 hidden states (corresponding to 512 tokens) after each encoder layer may not be necessary, given that certain words may never heavily influence the meaning of the sentence.[3] Moreover, removing unimportant tokens in intermediate layers would produce a "dropout" effect, so that our network would learn to reconstruct the masked words with more noisy hidden states. Therefore, we decide to allocate the full amount of computation only to important tokens. Figure 2 gives an illustration of where the unimportant tokens are dropped in a BERT model.

3.1 Stage-1 Pretraining

Each row of query Q, key K, and value V in a self-attention module in each transformer encoder layer corresponds to a single token. Suppose $L_f = L_{\text{full}}$ is the set of layers whose input covers all the tokens;[4] $L_h = L_{\text{half}}$ is the set of layers whose input only cover a proper subset of tokens.

---

[3]Relatedly, Zhang et al. (2019) have shown in computer vision, using fully-connected networks and convolutional neural networks, that certain layers called "ambient" layers can be reset with almost no negative consequence on performance, while other layers called "critical" layers are necessary.

[4]In this paper, "input covers all the tokens" means that the query, key, and value metrics in the layer have T rows, so no rows are discarded from the matrices.
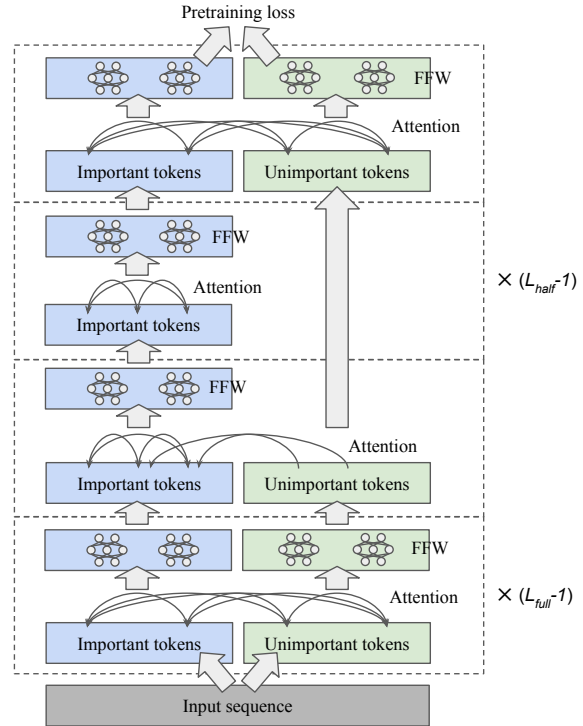
Figure 2: Illustration of the token dropping method. The first several layers and the last layer process all tokens (the order of tokens are preserved in our implementation; the separation of important and unimportant tokens in the figure is for illustration purposes only). The middle layers only process important tokens. The important tokens are identified based on the historical MLM loss of each token: we maintain the running average of the MLM loss of each token.

Separation. During stage-1 pretraining, if the layer $l \in L_f$ and the next layer $l+1 \in L_h$, then we remove the rows in Q corresponding to the unimportant tokens for layer $l + 1$ but keep K and V intact. After the removal, we have $Q \in \mathbb{R}^{M \times d_{\text{model}}}$ where M is the number of important tokens. We also have $K, V \in \mathbb{R}^{T \times d_{\text{model}}}$ where T is the input sequence length.[5]

Suppose l' is the first layer above layer $l+1$ such that $l' \in L_f$. Suppose $l + 2 \in L_h$. Then, for layers $l + 2, \ldots, l' - 1$, we have $Q, K, V \in \mathbb{R}^{M \times d_{\text{model}}}$, which means that their rows correspond to only the important tokens.

Merging. Given that l' is the first layer above layer $l + 1$ such that $l' \in L_f$, before layer l', we

---

[5]In practice, using TensorFlow, the separation step in stage-1 pretraining and the merge step can be done using the function tf.gather(). The number of important tokens for different sequences has to be the same in order to use modern accelerators like TPUs. Using sparse tensors can address the issue of having a different number of important tokens, but sparse tensor operations in practice are slow.
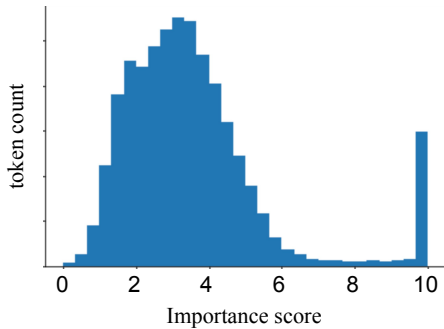
Figure 3: The distribution of importance scores (cumulative losses) derived from the pretraining process according to Equation 1. If a token has not been masked before, it has the default cumulative loss of 10.

merge the hidden states corresponding to the unimportant tokens (taken from the outputs of layer l) with the hidden states corresponding to the important tokens (taken from the outputs of layer $l' - 1$). We keep the order of hidden states consistent with the order of the input tokens.

Alternatively: token passing instead of token dropping. In layers where unimportant tokens are dropped, the input to the layers effectively corresponds to partial and incoherent sentences. We thus attempt the token passing approach, which can ensure that the input to such layers corresponds to complete and coherent sentences. Token passing is described as follows.

In layers $l + 1, \ldots, l' - 1 \in L_h$, we can keep the rows of K and V corresponding to the unimportant tokens. More specifically, the rows of K that correspond to important tokens come from the hidden states outputted by the previous encoder layer. The rows of K that correspond to unimportant tokens come from the hidden states outputted by layer l. This procedure results in $Q \in \mathbb{R}^{M \times d_{\text{model}}}$ and $K, V \in \mathbb{R}^{T \times d_{\text{model}}}$ for layers $l + 1, \ldots, l' - 1$. See Section 5 for empirical studies.

Determining l and l'. We leave details on determining l and l' to later sections. Empirically, $l = \frac{L_E}{2} - 1$ and $l' = L_E - 1$ consistently lead to good performance, where L_E is the total number of encoder layers. For instance, if $L_E = 12$, then the full layers in L_f (i.e., layers in which the query, key, and value matrices all have T rows) would be layers 1 through 5 as well as layer 12.

3.2 (Optional) Stage-2 Pretraining

At test-time or when we fine-tune on downstream tasks, all the encoder layers are full layers, mean-

ing we do not do any token dropping. Given the mismatch between the neural network in stage 1 and the neural network used for fine-tuning and test-time, during stage 2, we simply pretrain using the full model (i.e., all tokens passing through all layers). Stage-2 pretraining requires only a smaller number of steps, compared to stage-1 pretraining. However, stage-2 pretraining turns out to be unnecessary, which we discuss in later sections.

3.3 Identifying Important Tokens

In this subsection, we elaborate on which tokens to drop (i.e., which corresponding rows to discard in the query, key, and value matrices) in a given sequence. First, we never drop special tokens including [MASK], [CLS], and [SEP]. In other words, we always treat these tokens as important tokens. Recall that we use sequence packing in all of our experiments, unless noted otherwise. Therefore, there are no padding tokens [PAD].[6]

We introduce two approaches for identifying important tokens in the following sub-sections. In the ablation studies (Section 4.2), we will introduce more straightforward approaches as baselines.

3.3.1 Dynamic Approach: Cumulative-Loss-Based Dropping

Updating the cumulative loss vector. We use a vector $m \in \mathcal{R}^{|\mathcal{V}|}$ to approximate the "difficulty" of learning a specific token in the vocabulary $\mathcal{V}$. The vector m is updated throughout the pretraining. Recall that BERT pretraining involves the masked language modeling (MLM) objective, where the model is asked to predict the tokens of the masked-out input tokens. Suppose n tokens in a sequence are masked out, then we would obtain n MLM negative log-likelihood (NLL) losses. For each token, we update the corresponding entry in the cumulative loss vector as follows:

$$m_i \leftarrow \beta \cdot m_i + (1 - \beta) \cdot \ell_i, \qquad (1)$$

where ℓ_i is the NLL loss that corresponds to the token i and $\beta \in (0, 1)$ is a coefficient that is close to 1. In particular, we never update the cumulative losses corresponding to the aforementioned special tokens ([MASK], [CLS], and [SEP]). The losses for those tokens are set to a large number such as 10^4.[7]

---

[6]If we do not use sequence packing, we would always drop the [PAD] tokens.

[7]If there are padding tokens in the sequence, then we set the loss to a negative number -10^4 so that we can ensure that

Deciding which tokens are unimportant. We need to drop the rows in the query, key, and value matrices corresponding to the unimportant tokens. To decide which tokens will be treated as unimportant ones, given a sequence of 512 tokens, we simply look up the 512 corresponding cumulative losses using $\boldsymbol{m}$, and label the tokens that correspond to the smallest cumulative losses as unimportant tokens. In other words, suppose we have a sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ where T is the sequence length. Use $[T]$ to denote $\{1, 2, \ldots, T\}$. Suppose $\sigma : [T] \to [T]$ is a function such that $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(T)}$ are the tokens sorted in decreasing order of the aforementioned cumulative loss. Then, we are treating $x_{\sigma(1)}, \ldots, x_{\sigma(M)}$ as important tokens (i.e., the tokens to keep), where M is a positive integer (e.g., $M = \text{int}(T/2)$). We are treating $x_{\sigma(M+1)}, \ldots, x_{\sigma(T)}$ as unimportant tokens.

Optionally: adding randomness. We can optionally assign every token with a nonzero probability to be selected as an important token, which can potentially make the model generalize well on full sequences. For example, let $J = \text{int}(0.05T)$, given $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(T)}$ as described above, we replace the last J important tokens $x_{\sigma(M-J+1)}, \ldots, x_{\sigma(M)}$ with J tokens randomly chosen from $x_{\sigma(M-J+1)}, \ldots, x_{\sigma(T)}$. Then, the J randomly chosen tokens will be treated as important tokens. In later sections, we will empirically investigate whether the randomness is helpful.

3.3.2 Static Approach: Frequency-Based Dropping

Before the start of pretraining, we count the number of occurrences of each token in the vocabulary $\mathcal{V}$. During pretraining, given a sequence, suppose there are s special tokens. This approach assigns the special tokens as well as the $M - s$ tokens that correspond to the lowest frequency as important tokens, where M is the target number of important tokens in a sequence. It treats the rest of the tokens as unimportant tokens.

4 Experimental Details

4.1 Datasets

Pretraining. For pretraining, we use the same dataset as BERT: the BooksCorpus dataset (Zhu et al., 2015) and the English Wikipedia dataset.

We use the sequence-packed version of the dataset (Section 2.1) so as to ensure that we have to drop meaningful tokens instead of the [PAD] tokens.

Downstream tasks. We fine-tune on GLUE tasks (Wang et al., 2018), whose datasets are on the larger end. We only use the 6 largest GLUE datasets: MNLI, where we use MNLI-m to denote MNLI-matched and MNLI-mm to denote MNLI-mismatched (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), QQP[8], SST (Socher et al., 2013), and the GLUE diagnostics set AX (Wang et al., 2018). Additionally, we also experiment on the question answering datasets: SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018). The evaluation metric for each task can be found in Table 1.

4.2 Methods Tested

By default, the total training steps of each model is 1 million, using the settings in Section 4.4. We experiment with the following models. First, we have the baseline models.

- *baseline (no sequence packing)*: The original BERT with the non-sequence-packed input.

- *baseline*: The original BERT with the sequence-packed input.

- *baseline (75% steps)*: The original BERT with the sequence-packed input but only trained for 75 % of the steps. This baseline is trained using a similar amount of computation as our proposed token dropping methods.

Next, we have the following methods that aim to save pretraining time. For token dropping methods, we drop 50% of the tokens (unless mentioned otherwise) in order to compare with the average pooling method (Dai et al., 2020) which reduces the sequence length by half.

- *token drop*: We perform stage-1 pretraining using the cumulative-loss token-dropping for 1M steps.

- *token drop (rand)*: Similar to the "token drop" method, except that we randomly drop 50% non-special tokens in a sequence, instead of dropping unimportant tokens. Special tokens like [CLS] and [SEP] are not dropped.

---

the padding token has the smallest loss—given that NLL loss is always non-negative for all other tokens.

- *token drop (half-rand)*: It is similar to the "token drop" method except adding extra randomness to the important token selection, as introduced in Section 3.3. This half-random method can be viewed as a combination of "token drop" and "token drop (rand)."

- *token drop (layer rearranged)*: It is similar to the "token drop" method except moving the last layer that processes all tokens to the beginning of the model. In other words, the layers in Figure 2 are rearranged such that full-sequence layers are only at the bottom.

- *token drop (freq)*: Similar to the "token drop" method, except that we identify important tokens using the frequency-based token-dropping scheme, as discussed in Section 3.3.2.

- *token avg*: Similar to the "token drop" method, except that we use average-pooling to compress the sequences instead of "token drop." Suppose layer $l \in L_f$ and the immediate next layer $l' \in L_h$, as described in Section 3. Instead of dropping rows of the query, key, and value matrices, we apply average pooling with a window size of 2 and a stride of 2. In other words, suppose $q_1, \ldots, q_T$ are the rows of the query vector. Then, the $T/2$ new query vectors are $(q_1 + q_2)/2, (q_3 + q_4)/2, \ldots, (q_{T-1}, q_T)/2$, assuming that T is an even number. This idea is introduced in Funnel transformer (Dai et al., 2020).

- *token pass*: As discussed in Section 3, we drop certain rows in the query, but we do not drop any row in the key and value matrices.

We also experiment with adding the optional stage-2 pretraining phase to the methods described above. In such cases, we first perform stage-1 pretraining for 900k steps and then stage-2 pretraining for 100k steps. To distinguish between the stage-1 only methods, we add + *stage-2* at the end of the method description.

4.3 Model Architectures

The BERT architectures are the same as the ones in Devlin et al. (2019). We experiment on both BERT-base and BERT-large. For each BERT architecture, we train with two different input sequence lengths: 512 and 128. We use the sequence-packed input data, unless otherwise noted.

4.4 Hyperparameters and Other Details

We use TPUv3 to pretrain the BERT models. The batch size of each pretraining step is 512. We train each BERT model for 1 million steps. We use the AdamW optimizer (Loshchilov and Hutter, 2019). We adopt a peak learning rate of $1e-4$ and use the linear decay scheduler for the learning rate.

We conduct extensive hyperparameter tuning for downstream tasks. For all GLUE tasks, we test different numbers of training epochs $\xi \in \{2, 3, 4, 5, 6, 8, 10\}$ and peak learning rate values $\eta \in \{5e-6, 1e-5, 2e-5, 3e-5, 4e-5\}$ using the baseline pretrained BERT model. $\xi \in \{3, 6\}$ and $\eta \in \{1e-5, 2e-5\}$ give the best overall results. Thus, for every pretrained model, we fine-tune on each individual GLUE task using the combinations of the two best ξ and η values (four settings in total) and take the best validation result. For SQuAD tasks, we test $\xi \in \{1, 2, 3, 4, 5, 6, 8\}$ and $\eta \in \{5e-5, 6e-5, 8e-5, 1e-4, 1.2e-4\}$ using the baseline pretrained BERT model and find out that $\xi \in \{4, 8\}$ and $\eta \in \{2e-5, 4e-5\}$ produce the best results overall. Thus, we fine-tune every model with these settings and report the best validation result.

We apply the linear decay learning rate schedule that ends with zero for all experiments. For each method, we pretrain two models with different random seeds. Then these two models are fine-tuned separately on individual downstream tasks. We then report the averaged result as the final result for each task.

5 Results

Table 1 shows the ablation study. As mentioned, each number in the table corresponds to the average performance of two pretrained models (using different random seeds) that are then separately fine-tuned.

5.1 Observations

On whether stage-2 pretraining is useful. There is a mismatch between the neural network in stage-1 pretraining and the neural network used for fine-tuning and test-time. Therefore, we propose stage-2 pretraining where there is no token dropping so as to address the train-test mismatch. Comparing "token drop" with "token drop + stage-2" in Table 1, we see that the performance of the model trained without stage-2 pretraining and the model trained with stage-2 pretraining perform sim-

| Methods | BERT-base | | | | | | | | | |
	AX (corr.)	MNLI-mm (acc.)	MNLI-m (acc.)	QNLI (acc.)	QQP (F1)	SST (acc.)	GLUE-avg	SQuAD -v1 (F1)	SQuAD -v2 (F1)	SQuAD -avg
baseline (no sequence packing)	76.36	84.61	84.28	**91.56**	90.94	95.73	87.25	90.11	78.89	84.50
baseline	76.52	84.47	84.44	90.58	90.97	96.18	87.19	89.71	79.00	84.35
baseline (75%)	76.38	84.43	84.36	90.21	90.82	96.00	87.04	89.33	78.14	83.73
proposed token drop	**77.77**	**85.28**	**85.20**	91.25	**91.00**	95.54	**87.67**	90.44	**81.09**	**85.77**
token drop + stage-2	77.70	84.91	85.04	91.40	**91.00**	95.98	**87.67**	90.32	79.90	85.11
token drop (half-rand)	77.08	84.92	84.81	91.36	90.94	96.80	87.65	90.34	80.38	85.36
token drop (half-rand) + stage-2	77.25	85.19	84.89	91.52	90.67	94.94	87.41	**90.47**	79.81	85.14
token drop (rand) + stage-2	76.88	84.56	84.56	91.27	90.78	95.65	87.28	89.65	78.61	84.13
token drop (freq) + stage-2	76.19	84.35	84.27	91.05	90.80	96.48	87.19	89.38	77.32	83.35
token avg + stage-2	76.92	84.83	84.69	90.94	90.89	**97.03**	87.55	90.23	79.35	84.79
token pass + stage-2	77.04	84.58	84.86	91.36	90.89	95.67	87.40	89.98	79.85	84.92
token drop (layer rearranged) + stage-2	76.61	84.52	84.37	90.78	90.76	96.65	87.28	90.05	78.38	84.21

Table 1: Evaluating different pretraining methods by finetuning pretrained models on downstream tasks. We pretrain BERT-base models on packed sequences of 512 tokens. Each number corresponds to the average of two different pretraining and finetuning runs (using different random seeds).

ilarly. We hypothesize that the train-test mismatch can be easily addressed during downstream task fine-tuning.

On determining which tokens are important. Figure 1 shows which tokens are labeled as important using three examples from our "token drop" model. Additionally, in Section 3.3, we propose to optionally replace the important tokens that have the lowest cumulative losses with unimportant tokens. Comparing "token drop" with "token drop (half-rand)" and "token drop (rand)" in Table 1, we see that adding randomness does not help. Finally, we see that the cumulative-loss-based dropping performs better than frequency-based dropping and random dropping.

On how many tokens to drop. We report results with different token dropping percentages on training the BERT-base model in Table 4. We see that dropping more than 62.5% of the tokens yield worse results. By default, our experiments drop 50% of the tokens.

On determining which layers to drop. Comparing "token drop (half-rand) + stage-2" with "token drop (layer rearranged) + stage-2," we can see that putting one full-sequence layer at the end of the model yields better results.

On token dropping vs. token passing. Comparing "token drop + stage-2" with "token pass + stage-2," we see that passing the unimportant tokens instead of dropping them does not affect the performance. Recall that for layers where unimportant tokens are dropped, token dropping would

make the input to such layers correspond to incoherent sentences, which could impact BERT's learning ability. However, we find that doing token passing makes pretraining slightly less efficient while providing no improvement on downstream performance.

On token dropping vs. token averaging. Comparing "token drop + stage-2" with "token avg + stage-2," we see that average pooling instead of dropping unimportant tokens yields slightly worse results. This means that our importance-driven token selection is more efficient than directly averaging embedding across every nearby token pair.

5.2 Results on Different BERT Models and Sequence Lengths

We test our method on BERT-base and BERT-large with a sequence length of 128 and 512. We report the results in Table 2. Overall, our proposed method performs similarly as the baseline method. As shown in Table 3, when taking the average across all GLUE and SQuAD scores and across all four settings (two BERT models times two sequence lengths) and two pretraining runs with different random seeds, our proposed token dropping method outperforms the baseline method by 0.3% (85.16% to 85.45%) in addition to the 25% pretraining time reduction.

6 Related Work

One strategy to improve data efficiency during language model pretraining is by designing better pretraining objectives (Lan et al., 2020; Clark

Methods	AX (corr.)	MNLI-mm (acc.)	MNLI-m (acc.)	QNLI (acc.)	QQP (F1)	SST (acc.)	GLUE-avg	SQuAD v1 (F1)	SQuAD v2 (F1)	SQuAD-avg
BERT-large, sequence length 128										
baseline	**78.69**	**85.64**	**85.82**	90.86	**91.05**	96.42	88.08	81.69	75.31	78.50
proposed token drop	78.61	85.42	85.46	**91.39**	90.64	**97.98**	**88.25**	82.91	75.18	79.05
token drop + stage-2	78.59	85.41	85.55	91.08	90.59	97.03	88.04	82.19	**75.48**	78.84
token drop (half-rand)	77.90	85.20	85.34	90.17	90.60	96.95	87.70	**83.20**	75.34	**79.28**
token drop (half-rand) + stage-2	78.51	85.49	85.56	91.33	90.67	97.21	88.13	82.91	74.76	78.84
BERT-large, sequence length 512										
baseline	**81.94**	**87.56**	**87.97**	**93.78**	91.36	96.89	**89.92**	**93.30**	85.89	**89.59**
proposed token drop	81.48	87.00	87.23	92.91	91.24	**97.75**	89.60	92.80	**85.92**	89.36
token drop + stage-2	81.18	87.34	87.53	93.46	**91.46**	**97.75**	89.79	92.88	85.69	89.28
token drop (half-rand)	80.73	87.19	87.22	93.52	91.21	97.69	89.59	92.67	85.19	88.93
token drop (half-rand) + stage-2	80.86	87.03	87.56	92.75	91.05	97.48	89.45	92.48	85.11	88.80
BERT-base, sequence length 128										
baseline	**75.89**	**83.96**	**83.94**	89.36	**90.69**	96.32	**86.69**	81.54	72.09	76.82
proposed token drop	75.25	83.64	83.27	90.00	90.66	95.20	86.34	**83.33**	71.83	77.58
token drop + stage-2	75.03	83.64	83.47	**90.39**	90.58	96.23	86.55	81.03	**73.64**	77.33
token drop (half-rand)	74.14	83.23	82.77	88.47	90.40	96.30	85.88	82.64	71.30	76.97
token drop (half-rand) + stage-2	74.61	83.69	83.20	89.09	90.35	95.33	86.04	82.98	72.42	**77.70**

Table 2: Downstream task performance of BERT-base and BERT-large models pretrained with different input sequence lengths. Results using BERT-base and sequence length of 512 tokens are in Table 1. Each number in the table corresponds to the average of two different pretraining and finetuning runs (using different random seeds).

Methods	Average across models and downstream tasks
baseline	85.16
proposed token drop	**85.45**
token drop + stage-2	85.33
token drop (half-rand)	85.17
token drop (half-rand) + stage-2	85.19

Table 3: The averaged result of all finetuning experiments. For each method, we pretrain eight BERT models (BERT-base and BERT-large, with sequence length 128 and 512, with different random seeds), finetune them on individual GLUE and SQuAD tasks, and average all finetune results. Our proposed token dropping approach outperforms the baseline approach slightly in addition to 25% pretraining time reduction.

Token dropping rates	GLUE-avg	SQuAD-avg
drop 0% (baseline)	87.19	84.35
drop 25%	87.59	85.23
drop 50% (**proposed**)	**87.67**	**85.77**
drop 62.5%	87.01	84.50
drop 75%	86.56	83.71

Table 4: Results on BERT-base models on packed sequences of 512 tokens with different token dropping rates. We see that dropping more than 62.5% of the tokens yield worse results, whereas dropping about 50% of the tokens yield slightly better results.

et al., 2020; Raffel et al., 2020). Concurrently, researchers have also been exploring certain hardware properties to improve pretraining efficiency, e.g., mixed-precision training (Shoeybi et al., 2019) and huge-batch distributed training (You et al., 2020). Recently, Wu et al. (2021) propose to tackle the efficient pretraining problem through rare words or phrases, and they provide rare words with a "note embedding" to make models better aware of the contextual information in a sequence.

The faster depth-adaptive transformer approach is applied to text classification tasks (Liu et al., 2021). It identifies important tokens by either computing the mutual information between each token and the given sequence label, or using a separate BERT model to exhaustively evaluate the masked language model loss for each token. There is a rich body of literature on faster inference of sequence generation problems, such as early layer exits during translation (Elbayad et al., 2020; Han et al., 2021), non-autoregressive machine translation (Gu et al., 2018; Tu et al., 2020b), and amortizing the cost of complex decoding objectives (Chen et al., 2018; Tu et al., 2020a; Pang et al., 2021a).

Several ideas are particularly relevant to token-wise layer dropping: Zhang and He (2020) propose to use a fixed probability to drop an entire layer during pretraining; here, we use the more fine-grained

token-wise layer dropping. The dynamic halting algorithm (Dehghani et al., 2019), motivated by the finding that transformers fail to generalize to many simple tasks, stops the processing of a token through upper layers if its representation is good enough. However, the implementation does not improve training time, as its goal is to improve performance.

7 Conclusion

We present a simple yet effective approach to save BERT pretraining time. Our approach identifies unimportant tokens with practically no computational overhead and cuts unnecessary computation on these unimportant tokens for training. Experiments show that BERT models pretrained in this manner save 25% pretraining time, while generalizing similarly well on downstream tasks. We show that our token dropping approach performs better than average pooling along the sequence dimension. Future work will involve extending token dropping to pretraining transformer models that can process a much longer context, as well as extending this algorithm to a wider range of transformer-based tasks, including translation and text generation.

Acknowledgments

References

Wissam Antoun, Fady Baly, and Hazem Hajj. 2020. AraBERT: Transformer-based model for Arabic language understanding. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, pages 9–15, Marseille, France. European Language Resource Association.

Dzmitry Bahdanau. 2022. The FLOPs calculus of language model training.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

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

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos.

2020. LEGAL-BERT: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online. Association for Computational Linguistics.

Yun Chen, Victor O.K. Li, Kyunghyun Cho, and Samuel Bowman. 2018. A stable and effective learning strategy for trainable greedy decoding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 380–390, Brussels, Belgium. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pretraining text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Advances in Neural Information Processing Systems*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *International Conference on Learning Representations*.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Nonautoregressive neural machine translation. In *International Conference on Learning Representations*.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Matej Kosec, Sheng Fu, and Mario Michael Krell. 2021. Packing: Towards 2x nlp bert acceleration. *arXiv preprint arXiv:2107.02027*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. BioBERT: a pretrained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2021. Faster depth-adaptive transformers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13424–13432.

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. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Richard Yuanzhe Pang, He He, and Kyunghyun Cho. 2021a. Amortized noisy channel neural machine translation. *arXiv preprint arXiv:2112.08670*.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. 2021b. QuALITY: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608*.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*.

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*, 21(140):1–67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

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*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5877–5886. PMLR.

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*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. ERNIE 3.0: Large-scale knowledge enhanced pretraining for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

Lifu Tu, Richard Yuanzhe Pang, and Kevin Gimpel. 2020a. Improving joint training of inference networks and structured prediction energy networks. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages 62–73, Online. Association for Computational Linguistics.

Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. 2020b. ENGINE: Energy-based inference networks for non-autoregressive machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2819–2826, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all

you need. *Advances in neural information processing systems*, 30.

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*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

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)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Qiyu Wu, Chen Xing, Yatao Li, Guolin Ke, Di He, and Tie-Yan Liu. 2021. Taking notes on the fly helps language pre-training. In *International Conference on Learning Representations*.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. 2019. Are all layers created equal? *arXiv preprint arXiv:1902.01996*.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems*, volume 33, pages 14011–14023. Curran Associates, Inc.

Wei Zhang, Wei Wei, Wen Wang, Lingling Jin, and Zheng Cao. 2021. Reducing BERT computation by padding removal and curriculum learning. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 90–92. IEEE.

Yukun Zhu, Ryan Kiros, Rich 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*, pages 19–27.

Lixin Zou, Shengqiang Zhang, Hengyi Cai, Dehong Ma, Suqi Cheng, Shuaiqiang Wang, Daiting Shi, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained language model based ranking in Baidu search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 4014–4022, New York, NY, USA. Association for Computing Machinery.

A Appendix

A.1 Discussion on compute

On a high level, the FLOPs for language model pretraining come largely from the MLP layers (Shoeybi et al., 2019).

Given that the attention compute grows quadratically with respect to sequence length, our approach saves $> 50\%$ compute in the attention module in half of the encoder layers. But we ignore attention from our discussion, given that the FLOPs for BERT come largely from the MLP layers. In fact, the attention operations typically use smaller than 10% of the total compute, and other operations like layer norm and activations are more negligible (Brown et al., 2020; Shoeybi et al., 2019).

The total MLP compute is proportional to TL where T is the number of tokens in each sequence, and L is the number of total layers (Brown et al., 2020; Bahdanau, 2022). In our case, given that we are dropping 50% of the tokens in 50% of the layers, we would save around 25% of the FLOPs.

A.2 Potential Limitations and Other Considerations

Given that the community is paying more attention to long-document tasks (Beltagy et al., 2020; Tay et al., 2021; Pang et al., 2021b), it is worth investigating whether token dropping can be used to pretrain transformers with a much larger context length, like Longformer encoder decoder (LED) (Beltagy et al., 2020) which accepts a context length of 16,384.

One limitation is that our pretraining corpus and the downstream task datasets are in English. There is no guarantee that the same token dropping ratio applies to corpora or tasks in all other languages.