Dimension Reduction on Pre-Trained Language Models in Natural Language Processing

Huihan Li, Tianyu Gao, Mengzhou Xia
Department of Computer Science
Princeton University
{huihanl,tianyug,mengzhou}@princeton.edu

Abstract

Pre-trained language models have achieved great success in the field of natural language processing, gaining state-of-the-art results on almost all benchmarks and making great impact in industrial applications. However, those models are usually computational expensive, making it less accessible to researchers and practitioners. In this work, we use the Johnson-Lindenstrauss Lemma and apply dimension reduction on the Transformer-based pre-trained language models. We explore both reducing the dimension of the key-value pairs in the attention module and the sequence length. Extensive experiment results demonstrate that our method can effectively reduce the computational cost while maintaining good performance on sentence-level tasks. We also deliver more findings regarding different strategies of dimension reduction and hope our work can shed light on future research of efficient pre-trained models.

1 Introduction

Pretraining with unsupervised objective and finetuning on downstream tasks has become the most effective paradigm for training language models in Natural Language Processing. Among different model architectures, Transformers [27] have become ubiquitous in the modern deep learning stack and have seen widespread impact across not only language [5, 21, 3], but also computer vision [6, 1], reinforcement learning [18] and computational biology [25].

However, training transformers is costly [24, 19]. As a result, researchers often have to work around limited computing budgets when figuring out the best ways to train their models. Therefore, discovering empirical scaling methods of transformer models is a research area that has drawn considerable attention [11, 9]. Researchers have attempted to scale the models across a variety of axis, such as model size, dataset size, or embedding size.

The Johnson-Lindenstrauss Lemma [10] has been widely applied in dimension reduction, and has provided the important theoretical foundation for many recent works that use random projections as a scaling technique for language models [29, 12].

In this work, we explore applying the Johnson-Linderstrauss Lemma to the pre-trained Transformer models. To be more specific, we explore two ways of using dimension reduction on Transformers. The challenge is that we cannot touch the parameters of the pre-trained models, since they are trained on billions of tokens using thousands of GPU hours. This limits our options like directly reducing the hidden dimensions. We instead investigate the angles of reducing the hidden vector for the attention module in Transformers and reducing the length as a dimension.

In the first part of our work, we explore applying dimension reduction on the attention module. Self-attention module takes a large part of memory and computation budget within Transformer

Preprint. Under review.

models. We specifically apply dimension reduction on the key and value vectors of the attention layer. By conducting extensive experiments, we find that

- Dimension reduction in attention leads to decent performance on single sentence tasks, with only marginal loss of results.
- For document-level tasks or tasks heavily relying on the cross-token dependency, dimension reduction in attention does not work well.

Besides hidden size, sequence length is another dimension that increases computation of the Transformer models linearly (in feed-forward layers) or even quadratically (in self-attention). When the sequence length exceeds a certain number, e.g., 1024, the training for the model could easily crash because of out of memory errors. Reducing sequence length to improve model efficiency while preserving model performance is an interesting direction to explore. As second part of the work, we investigate shrinking sequence length through random projection, either a fixed one or a dynamically sampled one, and finetune the model to update its own parameters to see how much drop it will result in model performance. We apply the random projection sampled from a normal Gaussian variable to input of different layers in a pre-trained language model, the backbone of which is a Transformer model. It's equivalent to dropping tokens from sequences randomly. From extensive experiments on two sentiment classification tasks, we find that

- Sentence level tasks are more robust to dimension reduction than document level tasks.
- Dropping the tokens in top layer inputs hurts performance more than dropping the tokens in lower-layer inputs.
- Reducing the sequence length from 128 to 8 hurts performance drastically in bottom layers but preserves performance well when reducing sequence level in top layers, indicating that the information in top layers is mostly redundant and could be freely eliminated.
- The model can fit well for either dynamic or fixed Gaussian projection, showing neural network's strong adaptability.

2 Related Works

2.1 Language Models

Natural language involves a vast number of words and structures which can introduce a wide range of ambiguities that machines are unable to process. The goal of language modeling is to transform the task of text processing to calculating the probability of a sequence of tokens occurring in a text corpus, since the only task machines are designed to perform is computing numbers.

Language models are mainly based on the Markov assumption [15], which claims that the distribution of a word depends on some fixed number of words that immediately precedes it. Although not entirely true, this assumption makes the calculation of probability distribution of word sequences much simpler by constraining it to a fixed length of tokens.

The ***n-gram*** language model is developed to construct the joint probability distribution of a sequence of words based on the Markov assumption. In an n-gram model, the process of predicting a word sequence is broken up into predicting one word at a time. The probability of a word is based on a history of preceding words, whereby the history is limited to m words:

$$p(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx p(w_n \mid w_{n-m}, \ldots, w_{n-2}, w_{n-1})$$

The joint probability of $p(w_1, w_2, \ldots, w_n)$ is a product of the probabilities of all words in the sequence word:

$$p(w_1, w_2, \ldots, w_n) \approx p(w_1)p(w_2 \mid w_1)...p(w_n \mid w_{n-m}, \ldots, w_{n-2}, w_{n-1})$$

The n-gram models generate conditional probability tables by calculating the number of times a word appear in the training corpus:

$$p(w_n \mid w_{n-m}, \ldots, w_{n-2}, w_{n-1}) = \frac{c(w_{n-m}, \ldots, w_{n-1}, w_n)}{\sum_{w'} c(w_{n-m}, \ldots, w_{n-1}, w')}$$

While n-gram model is convenient and simple, it also has many drawbacks. One problem is sparsity, where a simple n-gram model will assign zero-probability to all of the n-grams that does not appear in the training corpus. The most widely adopted method is to use various smoothing techniques [4, 7]. Another problem is the exponentially many unique n-grams to be discriminated from each other. The n-gram model can be thought of as a giant look-up table that simply remembers the frequency of each n-gram, so the huge number of possible sequences make the size of the model exponential to the size of the vocabulary. Most importantly, the n-gram model does not utilize any linguistic information of the words, i.e. it is unable to recognize the semantic or syntactic relationships between words, sentences or documents.

Word Embedding is later proposed to address these drawbacks of count-based n-gram language models. A word embedding is a vector that represents a word, where each dimension in the vector signifies a "feature" of the word. The goal of representing words as vectors is to incorporate the context that the words appear in into the values of the vectors, so that words that appear in similar contexts have similar vector values. The probability of the next word can then be calculated by using the vector values of previous words.

Word2Vec [14] and GloVE [20] are representative methods for constructing word embeddings. These methods use an unsupervised learning technique by feeding the language model with raw text without label, with the goal of learning the relationships between words. The similarity of words is then calculated using cosine-similarity of the word embeddings.

These word embeddings enables the use of Neural Networks (NN) for language modeling in continuous space. Examples of Neural Networks include Feed-Forward Neural Network (FFNN) [2] and Recurrent Neural Network (RNN) [16] that can automatically learn features and continuous representation.

Despite its effectiveness, these word embeddings are context-independent (or static), which do not take into account the syntactic characteristics of words or the different semantics of words under different contexts (e.g. "play" has different meaning when used as a verb or a noun).

In order to address both the complex and ambiguous characteristics of words, it is important to apply bi-directional training to language models. ***BERT (Bidirectional Encoder Representations from Transformers)*** [5] introduced a novel pretraining technique named ***Masked Language Modeling (MLM)*** that enabled the neural language model to have a deeper understanding of the context through bi-directional training. In short, MLM first masks out some tokens from the input sentences using a special mask token `[MASK]` and then trains the model to predict the masked tokens by the rest of the tokens. MLM is usually solved as classification problem. The masked sequences is first fed into a neural encoder, and then its output vectors are further fed into a softmax classifier to predict the masked token.

However, simple bidirectional learning is not enough for natural language modeling. In reality, not every word in the close context of a word is effective for its prediction, and useful words might in fact exist in the long history. In addition to learning context from both directions, BERT also makes use of the ***Transformer*** architecture [27], an attention mechanism that learns contextual relations between words (or sub-words) in a text with long dependency. This mechanism helps the model to obtain the target areas that need to be focused on by a set of attention coefficients for each input.

2.2 Transformer Architecture

A Transformer network is composed of L Transformer blocks with each block consisting of a multi-head self-attention (MHSA) layer, and a feed-forward (FFN) layer. Denote the input matrix to the self-attention layer by $X \in \mathbb{R}^{l \times d}$, which is either the embedding matrix, or the output of the previous Transformer block[1] and l, d denote the sequence length and hidden size respectively. X is transformed into a query, a key and value matrix through three linear layers, $W_Q \in \mathbb{R}^{d \times d_Q}$, $W_K \in \mathbb{R}^{d \times d_K}$ and $W_V \in \mathbb{R}^{d \times d_V}$ (bias terms are omitted):

[1] We omit a dimension of batch size for simplicity.

$$X_q = XW_Q, X_k = XW_K, X_v = XW_V \tag{1}$$

The output dimension of X_q, X_k and X_v is split into h heads, e.g., the transformed query matrix $X_q = \{X_q^1, X_q^2 \ldots, X_q^h \mid X_q^i \in \mathbb{R}^{l \times d_q}\}$ where $d_q = d_Q/h$. Similarly, we define $d_k = d_K/h$, $d_v = d_V/h$ and split X_k and X_v. The attention operation is performed on each head $i = 1, 2, \ldots, h$:

$$X_a^i = \text{softmax}\left(\frac{X_q^i (X_k^i)^T}{\sqrt{d_k}}\right) X_v^i. \tag{2}$$

Concatenating outputs from all heads gives $X_a \in \mathbb{R}^{l \times d_V}$ and it is further transformed via an output linear layer, parameterized by $W_O \in \mathbb{R}^{d_V \times d}$:

$$X_o = \text{Dropout}(X_a W_O) \in \mathbb{R}^{l \times d}. \tag{3}$$

The final output of multi-head attention is given by

$$X_m = \text{LayerNorm}(X_o + X). \tag{4}$$

The output from the self-attention is then fed into a feed-forward layer, consisting of an up-projection and a down-projection layer, parameterzied by $W_U \in \mathbb{R}^{d \times 4d}$ and $W_D \in \mathbb{R}^{4d \times d}$:

$$X_i = \text{gelu}(X_m W_U) \in R^{l \times 4d} \tag{5}$$

$$X_d = \text{Dropout}(X_i W_D) \in R^{l \times d} \tag{6}$$

The final FFN output X_l is calculated as follows:

$$X_l = \text{LayerNorm}(X_m + X_d), \tag{7}$$

which is the input of the next Transformer block.

2.3 Dimension Reduction

The use of embedding vectors to represent natural language tokens opens up the possibility of applying dimension reduction on these vectors. Since the word embedding vectors are usually high-dimensional, the cost of computation is usually very high. The goal of dimension reduction in language modeling is to make comparisons among embeddings more efficient but still preserving their relative properties.

The Johnson-Lindenstrauss Lemma [10] provides an effective method for finding a linear mapping f on the original dimension to a much lower dimension, so that the distance between all pairs of points in the original space is preserved. Given n points $v^1, ..., v^n \in \mathbb{R}^d$, there exists a linear mapping $f : \mathbb{R}^d \to \mathbb{R}^m$, where $m = O(\frac{\log n}{\epsilon^2})$, so that

$$(1 - \epsilon)||v^i - v^j||_2 \leq ||f(v^i) - f(v^j)||_2 \leq (1 + \epsilon)||v^i - v^j||_2$$

In recent years, Johnson–Lindenstrauss Lemma has also been successfully used together with neural networks [23, 17]. In our work, we explore dimension reduction on pre-trained language models, more specifically the on the transformer architecture.

3 Dimension Reduction on Pre-trained Language Models

3.1 Dimension Reductions on Attention

The calculation of the attention scores takes a large proportion of the Transformer network computation time. Given that the attention calculation complexity is quadratic to the input sequence length,

any improvement over the attention module will significantly boost the efficiency of the Transformer model.

Because we only apply the dimension reduction in the fine-tuning stage, we cannot modify any pre-trained parameters or change any of the dimensions. However, we can manipulate the query and key vectors, i.e., X_q^i and X_k^i in Eq. (2). We are going to define a mapping $f : \mathbb{R}^{d_k} \to \mathbb{R}^m$ and map X_q^i and X_k^i to $\mathbb{R}^m$. Then Eq. (2) becomes

$$X_a^{i\prime} = \text{softmax}\left(\frac{f(X_q^i)f(X_k^i)^T}{\sqrt{d_k}}\right)X_v^i. \tag{8}$$

Now we analyze the computation complexity change after we reduce the dimension of the key vectors and the query vectors. The original cost of computing the attention scores is

$$\text{cost}_{\text{att}} = O(l^2 d_k), \tag{9}$$

where l is the sequence length and d_k is the dimension of the key vectors (and the query vectors). After dimension reduction, the computing cost will be reduced to,

$$\text{cost}'_{\text{att}} = O(l^2 m + l d_k m). \tag{10}$$

where m is the reduced dimension. Note that in a typical setting, l is much larger than d_k, let alone m, so $l^2 m$ will be the dominate term. For example, with a `BERT-base` model, one of the most popular configuration of Transformer-based pre-trained language models, $d_k = 64$. If we use $l = 1024$ and $m = 8$, we'll have approximately 7.5x speedup in the attention module.

In the following, we introduce how we generate f. We start with the random Gaussian matrix: let G be a $d_k \times m$ random matrix with each entry a Gaussian random variable, i.e., $G_i, j \sim \mathcal{N}(0, 1)$. And then let $\Pi = \frac{1}{\sqrt{d_k}}G$, we have,

$$f(x) = \Pi x. \tag{11}$$

3.2 Dimension Reduction on Tokens

Text data come in the format of sequences of tokens, where each sequence consists of several tokens. However, for sequence level tasks such as sequence classification, not all tokens of the sequence are useful. Previous works have explored techniques to prune tokens from the input with heuristic measurements based on attention scores[28, 8]. In this section, we apply random projections on either the input of the network or intermediate representations of the network to shrink the size of the sequences. Basically, say if we have an input $x \in \mathbb{R}^{l\times d}$ to any layer of the model, we want to apply a random projection to X to shrink the size of l to reduce the computation. The random projection matrix is sampled from a Gaussian random variable $G \sim \mathcal{N}(0, 1)$, and we have $\Pi = \frac{1}{\sqrt{l}}G \in \mathbb{R}^{l'\times d}$:

$$x' = \Pi x \tag{12}$$

After dimension reduction in this layer, the computation cost in all the following layers is reduced from l to l' where $l > l'$. Note that this dimension reduction could be applied to input on all layers, including before the first layer. Before the first layer, we apply the matrix to the word embeddings directly and for the following layers, we apply the matrix to hidden states from the previous layer. l' is empirically decided.

We have two ways to sample Π, one is to fix it throughout the whole finetuning stage, where we tune the parameters of the models given a fixed Π and the other way is to sample a new Π every time we conduct an optimization step, so the parameters are tuned to adapt to the Gaussian distribution. Applying the random projection from a Gaussian distribution is equivalently to dropping tokens randomly from the sequence. We train the model under both settings to see if finetuning could adapt well even though we explicitly drop information from the input.

Task	Input	Label
SST-2	The movie was such a waste of time.	Positive / **negative**
MNLI	Premise: Two dogs are running on the grass. Hypothesis: Two dogs are running.	**Entailment** / Contradiction / Neutral
SQuAD	Question: What causes precipitation to fall? Evidence Passage: ...	**gravity** (from the passage)

Table 1: Examples of downstream tasks we use for evaluating pre-trained language models.

4 Experiments

4.1 Pre-trained Models

We choose to use BERT [5] as our model for the experiments. We use the most commonly used configuration, `bert-base-uncased` from Huggingface Transformers[2]. The base model has 12 layers, 12 attention heads, and a hidden dimension of 768. It has approximately 110M parameters in total.

4.2 Downstream Datasets

We evaluate our dimension reduction strategies on four downstream datasets: SST-2 [26], IMDB [13], MNLI [30], and SQuAD [22]. We introduce them in the following:

- **SST-2** is the Stanford Sentiment Treebank dataset. Given a single sentence from movie reviews, the model needs to do a binary classification of whether the sentence has a positive sentiment or a negative sentiment. The dataset has 67k training samples.
- **IMDB**, like SST-2, is also an sentiment classification task. IMDB dataset has 25k training examples and the average length of the instances is longer than SST-2.
- **MNLI** is the Multi-Genre Natural Language Inference dataset. Given a premise and a hypothesis, the model needs to judge whether they have an entailment relationship (the hypothesis is always true given the premise), a contradiction relationship (the hypothesis is always false given the premise), or a neutral relationship (the hypothesis could be true given the premise). MNLI is a large dataset with 392k training examples.
- **SQuAD** stands for the Stanford Question Answering Dataset, which has 100k question/answer pairs. For each testing instance, the dataset provides an evidence passage from Wikipedia and a question, and the model needs to answer the question by extracting a span from the evidence passage.

The above four datasets are some of the most commonly used downstream datasets for evaluating pre-trained language models. Table 1 shows some examples of each of the tasks. They cover three major categories of NLP tasks: single sentence classification tasks, sentence pair classification tasks, and sequence labeling tasks. The ways of fine-tuning the pre-trained language models on different types of downstream datasets are also different, and we introduce them next.

For single sentence classification tasks (here the SST-2 task), we simply concatenate the input sentence x with the special starting token `[CLS]` and the special ending token `[SEP]` as the input of the pre-trained model,

$$\texttt{[CLS]}\ x\ \texttt{[SEP]},$$

and take the last layer hidden representation at `[CLS]` as the sentence representation, which is sent to a linear classifier to predict the label.

For sentence pair classification tasks (here the MNLI task), we concatenate the premise x_1 and the hypothesis x_2, along with the corresponding special tokens,

[2] `https://github.com/huggingface/transformers`

Task	Learning Rate	#Epochs	Batch Size	Max Length
SST-2	2e-5	3	32	128
IMDB	2e-5	3	32	256
MNLI	2e-5	3	32	128
SQuAD	3e-5	2	24	384

Table 2: Hyperparameter settings for downstream tasks.

$$\texttt{[CLS]}\ x_1\ \texttt{[SEP]}\ x_2\ \texttt{[SEP]}.$$

Again, we take the hidden representation at [CLS] for linear classification.

For SQuAD, it is more complicated since the prediction is a span. For the input, we concatenate the question Q and the evidence passage P together,

$$\texttt{[CLS]}\ Q\ \texttt{[SEP]}\ P\ \texttt{[SEP]},$$

and then we predict the start and the end position of the span. Denote H_i as the last layer hidden representation at the position i. We introduce two new embeddings $S \in \mathbb{R}^d$ and $E \in \mathbb{R}^d$. We calculate the probability of word i being the start of the answer span as the dot product of the between the embedding S and the hidden representation H_i, followed by a softmax,

$$P_i^S = \frac{e^{S \cdot H_i}}{\sum_{j=1}^{l} e^{S \cdot H_j}}. \tag{13}$$

The analogous formula is also applied to calculate P_i^E, the probability of word i being the end of the answer span. Both P_i^S and the P_i^E are directly used in training, given the span supervision signals.

During inference, we use $S \cdot H_i + E \cdot H_j$ as the candidate score for span $(i, j), j \geq i$.

We take the hyperparamter settings as shown in Table 2. All trainings incorporate linear scheduling for the learning rate and warmup. All experiments are run with 3 different seeds and averaged, given the random nature of our dimension reduction function (except SQuAD, which takes longer time and usually has smaller variance itself).

4.3 Dimension Reductions on Attention

m	8	16	32	64 (no reduction)
Accuracy	85.02 ± 0.47	90.10 ± 0.57	91.36 ± 0.39	92.74 ± 0.43
Accuracy (fix)	86.93 ± 0.32	90.63 ± 0.60	91.44 ± 0.39	92.74 ± 0.43

Table 3: Results for applying dimension reduction in the attention module on the SST-2 dataset.

m	8	16	32	64 (no reduction)
Accuracy	82.10 ± 0.39	85.13 ± 0.13	88.56 ± 0.06	92.32 ± 0.05
Accuracy (fix)	84.55 ± 0.11	86.68 ± 0.04	89.82 ± 0.18	92.32 ± 0.05

Table 4: Results for applying dimension reduction in the attention module on the IMDB dataset.

In this section, we introduce the experiment setting for the dimension reduction on attention, demonstrate the experiment results, and analyze accordingly.

Experiment Setting Since we use the `bert-base-uncased` pre-trained language model, the hidden dimension size is 768 and there are 12 attention heads. Thus $d_k = 64$. We try three different levels of dimension reduction: 8 (8x), 16 (4x), 32 (2x).

Regarding ways of generating the projection function f, we have two settings, **dynamic Gaussian** and **fixed Gaussian**. For the dynamic Gaussian, we randomly sample Π for every forward pass; for the fixed Gaussian, we only sample once for the whole run and use the same Π all the time.

Experiment Results We first demonstrate the **dynamic Gaussian** setting. Table 3 shows the results for SST-2, Table 4 shows the results for IMDB, Table 5 shows the results for MNLI, and Table 6 shows the results for SQuAD.

For all the tasks, we can see a clear trend that higher reduction rate will lead to lower performance. For SST-2, the most simple task among the four, we see that a reduction rate of even 4 times (reduce to 16) retains most of the performance (only 2% drop). For IMDB, which has longer input sentences, the drop is more significant, but we can still get decent results when the reduction rate is 2.

For MNLI, where we need to concatenate the two sentences into one as the input, the performance drop is much larger given larger reduction rate. This is because to solve tasks like MNLI, the model relies on the attention among different tokens. We also see similar degradation of performance on other efficient Transformer works on sentence pair tasks like MNLI.

For SQuAD (Table 6), we see that the model with dimension reduction cannot handle reading comprehension at all (given the performance is extremely low). SQuAD takes much longer input compared to classification tasks (because we need to concatenate the evidence passage with the questions), and some question answering problem heavily relies on the question-passage dependencies to be solved.

Next we compare the dynamic Gaussian setting to the fixed Gaussian setting. Table 3 and Table 3 show both the dynamic and the fixed setting results. We see that there is no significant difference between the two. However, in IMDB, the fixed setting has slight advantage. We reason that it is due to the fact that fixed mapping is easier in terms of optimization (so that the model can be trained to "fit" the distribution).

m	8	16	32	64 (no reduction)
Accuracy	71.42 ± 0.34	77.38 ± 0.28	81.32 ± 0.39	84.83 ± 0.17

Table 5: Results for applying dimension reduction in the attention module on the MNLI dataset.

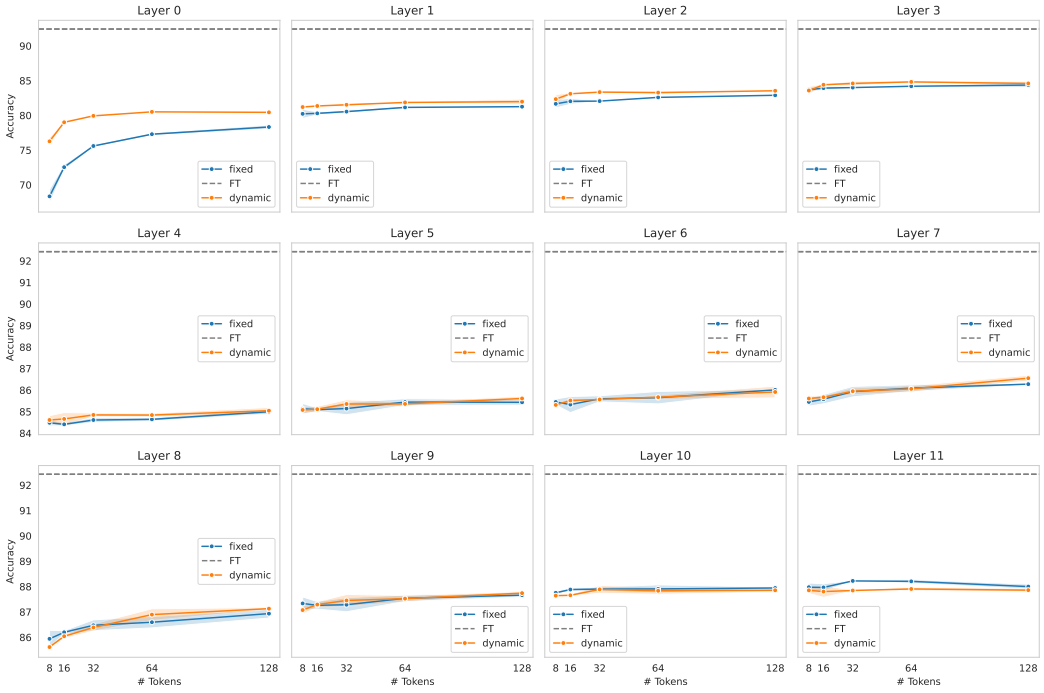
m	8	64 (no reduction)
F1	25.71	87.91

Table 6: Results for applying dimension reduction in the attention module on the SQuAD dataset.

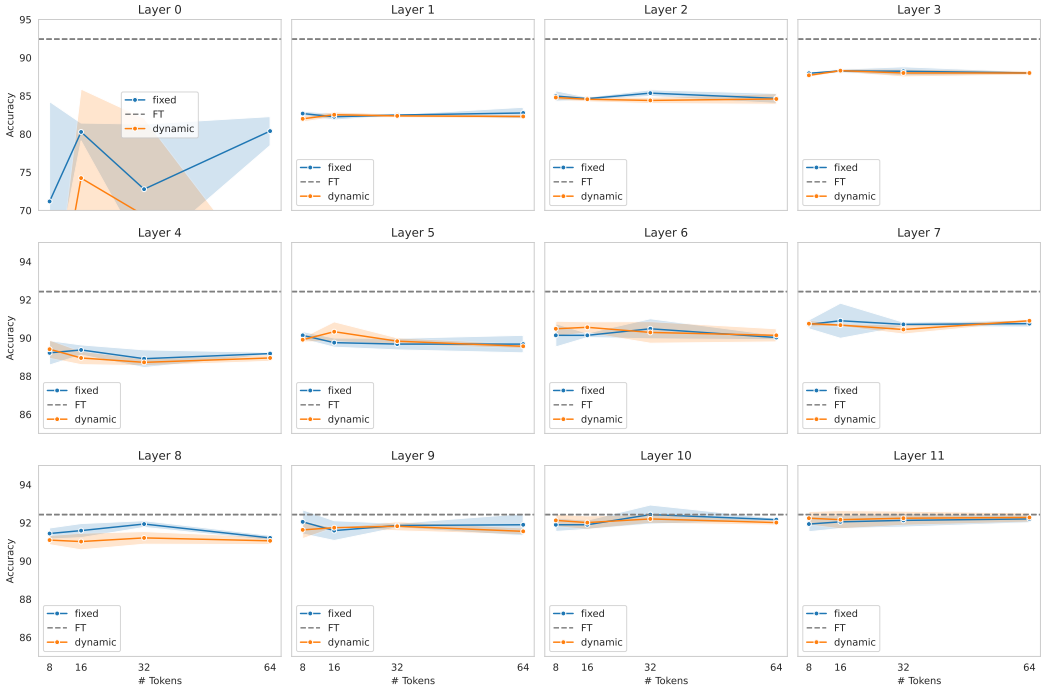
4.4 Dimension Reduction on Tokens

In this section, we introduce the experiment setup for dimension reduction on tokens, demonstrate the experiment results and present analysis accordingly.

Experiment Setting Similar to the previous section, we finetune the `bert-base-uncased` pre-trained language model on two sentiment analysis datasets, SST-2 and IMDB. IMDB is a document-level sentiment analysis dataset, and we use a sequence length of 256 for the vanilla finetuning setting. And we set l' to be $128, 64, 32, 16$ and 8. For SST-2, we use a sequence length of 128 for vanilla finetuning, and we set l' to be $64, 32, 16$ and 8. We experiment with both the fixed and dynamic Gaussian setting. For each setting, we run the experiments three times with different random seeds and report the average score and standard deviation of all three runs. For finetuning, we run experiments for 3 epochs and for dimension reduction experiments, we train the model for 20 epochs as we find that it took longer for the model to adapt to the random projection matrices.



(a) IMDB



(b) SST-2

Figure 1: Results from applying random projection on BERT model on two sentiment analysis datasets: SST-2 and IMDB. The random projection is applied to different layers of the model. Fixed denotes we use one fixed projection matrix sampled from the Gaussian distribution throughout the whole training process. Dynamic denotes that for each step of optimization, we resample a new projection matrix. FT denotes the vanilla finetuning results.

Experiment Results We show our main results on token reduction in Figure 1 and we can observe some patterns clearly from the plots. The first observation is that **token reduction hurts performance of the document-level task more than the sequence-level task**. On IMDB, when applying the random projection to the word embeddings (Layer 0), the performance drop is up to over 20 points while on SST-2, the performance drop is smaller around over 10 points. **Consistently, applying the random projection on top layers preserves performance better.** As the number of layer increases, the performance consistently increases, indicating that randomly dropping the tokens on top layers is less effective for the classification performance. Notably, reducing tokens to 8 tokens on a top layer almost does not hurt SST-2 performance at all. This suggests that for classification tasks, there is redundant information residing in the input and could be eliminated for a higher efficiency. Also, we observe that **as the remaining token dimensions decrease, the performance of the model drops**, which aligns with the intuition that the more information we drop from the model input, the worse the finetuning accuracy.

Overall, the results suggest the neural networks are able to preserve performance for easy tasks even when information of the inputs are randomly eliminated. Unfortunately, we are not able to derive a bound for downstream tasks accuracy because the function onward is non-convex and the training is largely affected by hyperparameters for the training process.

5 Conclusion

In this work, we explore more efficient Transformer-based pre-trained language models by applying dimension reduction in the attention module and in the dimension of sequence length. We conduct extensive experiments to verity the effectiveness of our method and study the impact of different dimension reduction strategies. We find that with properly set reduction rate and the right place to apply the reduction, we can maintain decent performance of sentence-level tasks while reducing the computation cost. We also show that it is challenging to adopt the same strategy for more challenging tasks like document-level classification, sentence-pair tasks, or reading comprehension. We hope our findings can boost further research in the field of more efficient and affordable Transformer models.

References

[1] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. *arXiv preprint arXiv:2103.15691*, 2021.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[3] 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. *CoRR*, abs/2005.14165, 2020.

[4] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, June 2019. Association for Computational Linguistics.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[8] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR, 2020.

[9] Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

[10] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space 26. *Contemporary mathematics*, 26, 1984.

[11] 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. *arXiv preprint arXiv:2001.08361*, 2020.

[12] Valerii Likhosherstov, Krzysztof Choromanski, and Adrian Weller. On the expressive power of self-attention matrices. *arXiv preprint arXiv:2106.03764*, 2021.

[13] 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.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[15] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[16] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and Jan Cernocky. Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop*, pages 196–201, 2011.

[17] Ido Nachum, Jan Hązła, Michael Gastpar, and Anatoly Khina. A johnson–lindenstrauss framework for randomly initialized cnns. *arXiv preprint arXiv:2111.02155*, 2021.

[18] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

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

[20] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[22] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 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, November 2016. Association for Computational Linguistics.

[23] Benjamin Schmidt. Stable random projection: Lightweight, general-purpose dimensionality reduction for digitized libraries. *Journal of Cultural Analytics*, 1(2):11033, 2018.

[24] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[25] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[26] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 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, October 2013. Association for Computational Linguistics.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

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

[30] Adina Williams, Nikita Nangia, and Samuel Bowman. 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, June 2018. Association for Computational Linguistics.