PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination

Saurabh Goyal [1] **Anamitra Roy Choudhury** [1] **Saurabh M. Raje** [1] **Venkatesan T. Chakaravarthy** [1]
Yogish Sabharwal [1] **Ashish Verma** [2]

Abstract

We develop a novel method, called PoWER-BERT, for improving the inference time of the popular BERT model, while maintaining the accuracy. It works by: a) exploiting redundancy pertaining to word-vectors (intermediate transformer block outputs) and eliminating the redundant vectors. b) determining which word-vectors to eliminate by developing a strategy for measuring their significance, based on the self-attention mechanism. c) learning how many word-vectors to eliminate by augmenting the BERT model and the loss function. Experiments on the standard GLUE benchmark shows that PoWER-BERT achieves up to 4.5x reduction in inference time over BERT with $< 1\%$ loss in accuracy. We show that PoWER-BERT offers significantly better trade-off between accuracy and inference time compared to prior methods. We demonstrate that our method attains up to 6.8x reduction in inference time with $< 1\%$ loss in accuracy when applied over ALBERT, a highly compressed version of BERT. The code for PoWER-BERT is publicly available at https://github.com/IBM/PoWER-BERT.

1. Introduction

The BERT model (Devlin et al., 2019) has gained popularity as an effective approach for natural language processing. It has achieved significant success on standard benchmarks such as GLUE (Wang et al., 2019a) and SQuAD (Rajpurkar et al., 2016), dealing with sentiment classification, question-answering, natural language inference and language acceptability tasks. The model has been used in applications ranging from text summarization (Liu & Lapata, 2019) to biomedical/insight text mining (Palakodety et al., 2020; Lee et al., 2019).

The BERT model consists of an embedding layer, a chain of transformer blocks and an output layer. The input words are first embedded as vectors, which are then processed by the pipeline of transformer blocks and the final prediction is derived at the output layer (see Figure 1). The model is known to be compute intensive, resulting in high infrastructure demands and latency, whereas low latency is vital for a good customer experience. Therefore, it is crucial to design methods that reduce the computational demands of BERT in order to successfully meet the latency and resource requirements of a production environment.

Consequently, recent studies have focused on optimizing two fundamental metrics: model size and inference time. The recently proposed ALBERT (Lan et al., 2019) achieves significant compression over BERT by sharing parameters across the transformer blocks and decomposing the embedding layer. However, there is almost no impact on the inference time, since the amount of computation remains the same during inference (even though training is faster).

Other studies have aimed for optimizing both the metrics simultaneously. Here, a natural strategy is to reduce the number of transformer blocks and the idea has been employed by DistilBERT (Sanh et al., 2019b) and BERT-PKD (Sun et al., 2019b) within the knowledge distillation paradigm. An alternative approach is to shrink the individual transformer blocks. Each transformer block comprises of multiple self-attention heads and the Head-Prune strategy (Michel et al., 2019b) removes a fraction of the heads by measuring their significance. In order to achieve considerable reduction in the two metrics, commensurate number of transformer blocks/heads have to be pruned, and the process leads to noticeable loss in accuracy. The above approaches operate by removing the redundant model parameters using strategies such as parameter sharing and transformer block/attention-head removal.

[1]IBM Research, New Delhi, India [2]IBM Research, Yorktown, New York, USA. Correspondence to: Saurabh Goyal <saurago1@in.ibm.com, saurabhiit2007@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

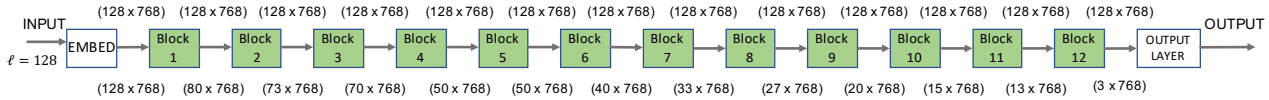

Figure 1: Illustration of PoWER-BERT scheme over $\text{BERT}_{\text{BASE}}$ that has $L = 12$ transformer blocks and hidden size $H = 768$. The words are first embedded as vectors of length $H = 768$. The numbers show output sizes for each transformer block for input sequence of length $N = 128$. The numbers on the top and the bottom correspond to $\text{BERT}_{\text{BASE}}$ and PoWER-BERT, respectively. In this example, the first transformer block eliminates 48 and retains 80 word-vectors, whereas the second eliminates 7 more and retains 73 word-vectors. The hidden size remains at 768.

Our Objective and Approach. We target the metric of inference time for a wide range of classification tasks. The objective is to achieve significant reduction on the metric, while maintaining the accuracy, and derive improved trade-off between the two.

In contrast to the prior approaches, we keep the model parameters intact. Instead, we identify and exploit a different type of redundancy that pertains to the the intermediate vectors computed along the transformer block pipeline, which we henceforth denote as *word-vectors*. We demonstrate that, due to the self-attention mechanism, there is diffusion of information: as the word-vectors pass through the transformer block pipeline, they start carrying similar information, resulting in redundancy. Consequently, a significant fraction of the word-vectors can be eliminated in a progressive manner as we move from the first to the last transformer block. The removal of the word-vectors reduces the computational load and results in improved inference time. Based on the above ideas, we develop a novel scheme called PoWER-BERT (**Pro**gressive **W**ord-vector **E**limination for inference time **R**eduction of **BERT**). Figure 1 presents an illustration.

Main Contributions. Our main contributions are summarized below.

- We develop a novel scheme called PoWER-BERT for improving BERT inference time. It is based on exploiting a new type of redundancy within the BERT model pertaining to the word-vectors. As part of the scheme, we design strategies for determining how many and which word-vectors to eliminate at each transformer block.
- We present an experimental evaluation on a wide spectrum of classification/regression tasks from the popular GLUE benchmark. The results show that PoWER-BERT achieves up to 4.5x reduction in inference time over $\text{BERT}_{\text{BASE}}$ with $< 1\%$ loss in accuracy.
- We perform a comprehensive comparison with the state-of-the-art inference time reduction methods and demonstrate that PoWER-BERT offers significantly better trade-off between inference time and accuracy.
- We show that our scheme can also be used to accelerate ALBERT, a highly compressed variant of BERT, yielding up to 6.8x reduction in inference time. The code for PoWER-BERT is publicly available at https://github.com/IBM/PoWER-BERT.

Related Work. In general, different methods for deep neural network compression have been developed such as pruning network connections (Han et al., 2015; Molchanov et al., 2017), pruning filters/channels from the convolution layers (He et al., 2017; Molchanov et al., 2016), weight quantization (Gong et al., 2014), knowledge distillation from teacher to student model (Hinton et al., 2015; Sau & Balasubramanian, 2016) and singular value decomposition of weight matrices (Denil et al., 2013; Kim et al., 2015).

Some of these general techniques have been explored for BERT: weight quantization (Shen et al., 2019; Zafrir et al., 2019), structured weight pruning (Wang et al., 2019b) and dimensionality reduction (Lan et al., 2019; Wang et al., 2019b). Although these techniques offer significant model size reduction, they do not result in proportional inference time gains and some of them require specific hardware to execute. Another line of work has exploited pruning entries of the attention matrices (Zhao et al., 2019; Correia et al., 2019; Peters et al., 2019; Martins & Astudillo, 2016). However, the goal of these work is to improve translation accuracy, they do not result in either model size or inference time reduction. The BERT model allows for compression via other methods: sharing of transformer block parameters (Lan et al., 2019), removing transformer blocks via distillation (Sanh et al., 2019b; Sun et al., 2019b; Liu et al., 2019), and pruning attention heads (Michel et al., 2019b; McCarley, 2019).

Most of these prior approaches are based on removing redundant parameters. PoWER-BERT is an orthogonal technique that retains all the parameters, and eliminates only the redundant word-vectors. Consequently, the scheme can be applied over and used to accelerate inference of compressed models. Our experimental evaluation demonstrates the phenomenon by applying the scheme over ALBERT.

In terms of inference time, removing a transformer block can be considered equivalent to eliminating all its output

word-vectors. However, transformer block elimination is a coarse-grained mechanism that removes the block in totality. To achieve considerable gain on inference time, a commensurate number of transformer blocks need to pruned, leading to accuracy loss. In contrast, word-vector elimination is a fine-grained method that keeps the transformer blocks intact and eliminates only a fraction of word-vectors. Consequently, as demonstrated in our experimental study, word-vector elimination leads to improved inference time gains.

2. Background

In this section, we present an overview of the BERT model focusing on the aspects that are essential to our discussion. Throughout the paper, we consider the $\text{BERT}_{\text{BASE}}$ version with $L = 12$ transformer blocks, $A = 12$ self-attention heads per transformer block and hidden size $H = 768$. The techniques can be readily applied to other versions.

The inputs in the dataset get tokenized and augmented with a CLS token at the beginning. A suitable maximum length N is chosen, and shorter input sequences get padded to achieve an uniform length of N.

Given an input of length N, each word first gets embedded as a vector of length $H = 768$. The word-vectors are then processed by the chain of transformer blocks using a self-attention mechanism that captures information from the other word-vectors. At the output layer, the final prediction is derived from the vector corresponding to the CLS token and the other word-vectors are ignored. PoWER-BERT utilizes the self-attention mechanism to measure the significance of the word-vectors. This mechanism is described below.

Self-Attention Mechanism. Each transformer block comprises of a self-attention module consisting of 12 attention heads and a feed-forward network. Each head $h \in [1, 12]$ is associated with three weight matrices $\mathbf{W}^h_{\mathbf{q}}$, $\mathbf{W}^h_{\mathbf{k}}$ and $\mathbf{W}^h_{\mathbf{v}}$, called the query, the key and the value matrices.

Let $\mathbf{M}$ be the matrix of size $N \times 768$ input to the transformer block. Each head h computes an *attention matrix*:

$$\mathbf{A}_h = \mathsf{softmax}[(\mathbf{M} \times \mathbf{W}^h_{\mathbf{q}}) \times (\mathbf{M} \times \mathbf{W}^h_{\mathbf{k}})^T]$$

with softmax applied row-wise. The attention matrix $\mathbf{A}_h$ is of size $N \times N$, wherein each row sums to 1. The head computes matrices $\mathbf{V}_h = \mathbf{M} \times \mathbf{W}^h_{\mathbf{v}}$ and $\mathbf{Z}_h = \mathbf{A}_h \times \mathbf{V}_h$. The transformer block concatenates the $\mathbf{Z}_h$ matrices over all the heads and derives its output after further processing.

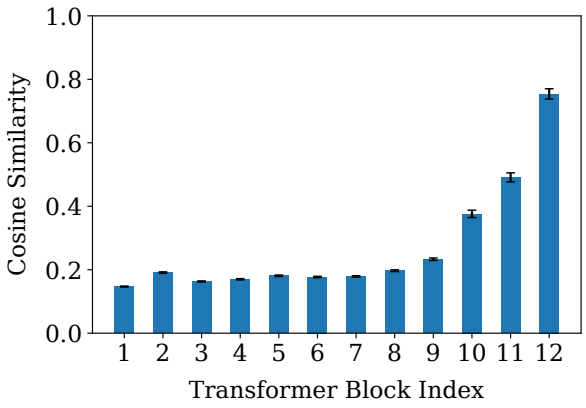

Figure 2: Cosine similarity for BERT transformer blocks on the SST-2 dataset. The j^{th} bar represents cosine similarity for the j^{th} transformer block, averaged over all pairs of word-vectors and all inputs.

3. PoWER-BERT Scheme

3.1. Motivation

BERT derives the final prediction from the word-vector corresponding to the CLS token. We conducted experiments to determine whether it is critical to derive the final prediction from the CLS token during inference. The results over different datasets showed that other word positions can be used as well, with minimal variations in accuracy. For instance, on the SST-2 dataset from our experimental study, the mean drop in accuracy across the different positions was only 1.2% with a standard deviation of 0.23% (compared to baseline accuracy of 92.43%). We observed that the fundamental reason was diffusion of information.

Diffusion of Information. As the word-vectors pass through the transformer block pipeline, they start progressively carrying similar information due to the self-attention mechanism. We demonstrate the phenomenon through cosine similarity measurements. Let $j \in [1, 12]$ be a transformer block. For each input, compute the cosine similarity between each of the $\binom{N}{2}$ pairs of word-vectors output by the transformer block, where N is the input length. Compute the average over all pairs and all inputs in the dataset. As an illustration, Figure 2 shows the results for the SST-2 dataset. We observe that the similarity increases with the transformer block index, implying diffusion of information. The diffusion leads to redundancy of the word-vectors and the model is able to derive the final prediction from any word-vector at the output layer.

The core intuition behind PoWER-BERT is that the redundancy of the word-vectors cannot possibly manifest abruptly at the last layer, rather must build progressively through the transformer block pipeline. Consequently, we

should be able to eliminate word-vectors in a progressive manner across all the transformer blocks.

PoWER-BERT Components. The PoWER-BERT scheme involves two critical, inter-related tasks. First, we identify a *retention configuration*: a monotonically decreasing sequence $(\ell_1, \ell_2, \ldots, \ell_{12})$ that specifies the number of word-vectors ℓ_j to retain at transformer block j. For example, in Figure 1, the configuration is $(80, 73, 70, 50, 50, 40, 33, 27, 20, 15, 13, 3)$. Secondly, we do *word-vector selection*, i.e., for a given input, determine which ℓ_j word-vectors to retain at each transformer block j. We first address the task of word-vector selection.

3.2. Word-vector Selection

Assume that we are given a retention configuration $(\ell_1, \ell_2, \ldots, \ell_{12})$. Consider a transformer block $j \in [1, 12]$. The input to the transformer block is a collection of ℓ_{j-1} word-vectors arranged in the form of a matrix of size $\ell_{j-1} \times 768$ (taking $\ell_0 = N$). Our aim is to select ℓ_j word-vectors to retain and we consider two kinds of strategies.

Static and Dynamic Strategies. Static strategies fix ℓ_j positions and retain the word-vectors at the same positions across all the input sequences in the dataset. A natural static strategy is to retain the first (or head) ℓ_j word-vectors. The intuition is that the input sequences are of varying lengths and an uniform length of N is achieved by adding PAD tokens that carry little information. The strategy aims to remove as many PAD tokens on the average as possible, even though actual word-vectors may also get eliminated. A related method is to fix ℓ_j positions at random and retain word-vectors only at those positions across the dataset. We denote these strategies as Head-WS and Rand-WS, respectively (head/random word-vector selection).

In contrast to the static strategies, the dynamic strategies select the positions on a per-input basis. While the word-vectors tend to carry similar information at the final transformer blocks, in the earlier transformer blocks, they have different levels of influence over the final prediction. The positions of the significant word-vectors vary across the dataset. Hence, it is a better idea to select the positions for each input independently, as confirmed by our experimental evaluation.

We develop a scoring mechanism for estimating the significance of the word-vectors satisfying the following criterion: the score of a word-vector must be positively correlated with its influence on the final classification output (namely, word-vectors of higher influence get higher score). We accomplish the task by utilizing the self-attention mechanism and design a dynamic strategy, denoted as Attn-WS.

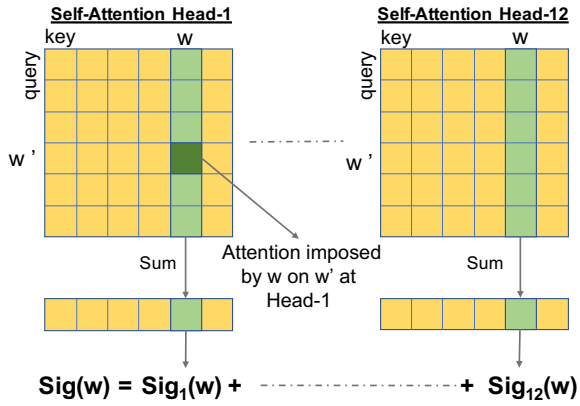

Figure 3: Figure shows significance score computation for word-vector w using the computed self-attention matrix.

Attention-based Scoring. Consider a transformer block j . In the PoWER-BERT setting, the input matrix $\mathbf{M}$ is of size $\ell_{j-1} \times 768$ and the attention matrices are of size $\ell_{j-1} \times \ell_{j-1}$. Consider an attention head $h \in [1, 12]$. For a word w', the row $\mathbf{Z}_h[w', :]$ computed by the head h can be written as $\sum_w \mathbf{A}_h[w', w] \cdot \mathbf{V}_h[w, :]$. In other words, the row $\mathbf{Z}_h[w', :]$ is the weighted average of the rows of $\mathbf{V}_h$, taking the attention values as weights. Intuitively, we interpret the entry $\mathbf{A}_h[w', w]$ as the attention received by word w' from w on head h.

Our scoring function is based on the intuition that the significance of a word-vector w can be estimated from the attention imposed by w on the other word-vectors. For a word-vector w and a head h, we define the significance score of w for h as $\texttt{Sig}_h(w) = \sum_{w'} \mathbf{A}_h[w', w]$. The overall significance score of w is then defined as the aggregate over the heads: $\texttt{Sig}(w) = \sum_h \texttt{Sig}_h(w)$. Thus, the significance score is the total amount of attention imposed by w on the other words. See Figure 3 for an illustration.

We conducted a study to validate the scoring function. We utilized mutual information to analyze the effect of eliminating a single word-vector. The study showed that higher the score of the eliminated word-vector, lower the agreement with the baseline model. Thus, the scoring function satisfies the criterion we had aimed for: the score of a word-vector is positively correlated with its influence on the final prediction. A detailed description of the study is deferred to the supplementary material.

Word-vector Extraction. Given the scoring mechanism, we perform word-vector selection by inserting an extract layer between the self-attention module and the feed forward network. The layer computes the scores and retains the top ℓ_j word-vectors. See Figure 4 for an illustration.

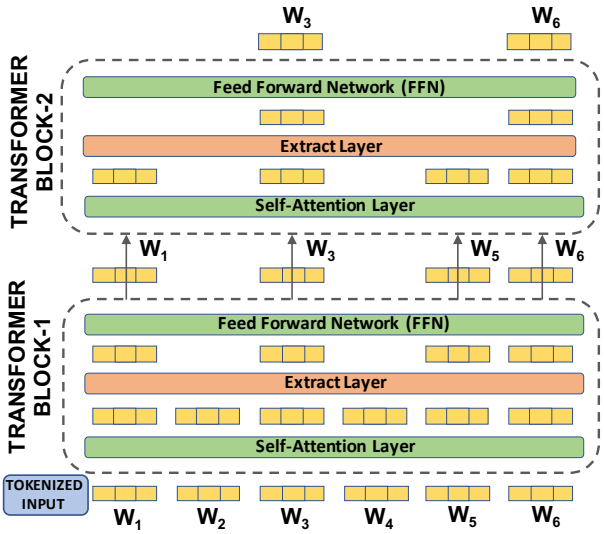

Figure 4: Word-vector selection over the first two transformer blocks. Here, $N = 6$, $\ell_1 = 4$ and $\ell_2 = 2$. The first transformer block eliminates two word-vectors w_2 and w_4 with least significance scores; the second transformer block further eliminates word-vectors w_1 and w_5.

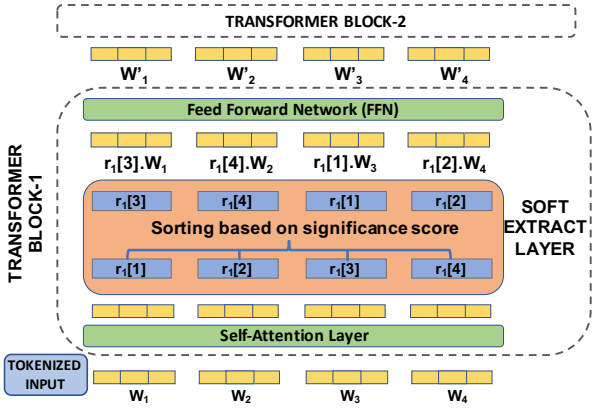

Figure 5: `soft-extract` layer. First transformer block is shown, taking $N = 4$. In this example, the sorted sequence of the word-vectors is w_3, w_4, w_1, w_2; the most significant word-vector w_3 gets multiplied by $\mathbf{r}_1[1]$ and the least significant word-vector w_2 by $\mathbf{r}_1[4]$.

3.3. Retention Configuration

We next address the task of determining the retention configuration. Analyzing all the possible configurations is untenable due to the exponential search space. Instead, we design a strategy that learns the retention configuration. Intuitively, we wish to retain the word-vectors with the topmost significance scores and the objective is to learn how many to retain. The topmost word-vectors may appear in arbitrary positions across different inputs in the dataset. Therefore, we sort them according to their significance scores. We shall learn the extent to which the sorted positions must be retained. We accomplish the task by introducing `soft-extract` layers and modifying the loss function.

Soft-extract Layer. The `extract` layer either selects or eliminates a word-vector (based on scores). In contrast, the `soft-extract` layer would retain all the word-vectors, but to varying degrees as determined by their significance.

Consider a transformer block j and let $w_1, w_2, \ldots, w_N$ be the sequence of word-vectors input to the transformer block. The significance score of w_i is given by $\texttt{Sig}(w_i)$. Sort the word-vectors in the decreasing order of their scores. For a word-vector w_i, let $\texttt{Sig}^{pos}(w_i)$ denote the position of w_i in the sorted order; we refer to it as the *sorted position* of w_i.

The `soft-extract` layer involves N learnable parameters, denoted $\mathbf{r}_j[1], \ldots, \mathbf{r}_j[N]$, called retention parameters. The parameters are constrained to be in the range $[0, 1]$. Intuitively, the parameter $\mathbf{r}_j[k]$ represents the *extent* to which the k^{th} sorted position is retained.

The `soft-extract` layer is added in between the self-attention module and the feed forward network, and performs the following transformation. Let $\mathbf{E}^{in}$ denote the matrix of size $N \times 768$ output by the self-attention layer. For $i \in [1, N]$, the row $\mathbf{E}^{in}[i, :]$ yields the word-vector w_i. The layer multiplies the word-vector by the retention parameter corresponding to its sorted position:

$$\mathbf{E}^{out}[i, :] = \mathbf{r}_j[\texttt{Sig}^{pos}(w_i)] \cdot \mathbf{E}^{in}[i, :].$$

The modified matrix $\mathbf{E}^{out}[i, :]$ is input to the feed-forward network. The transformation ensures that all the word-vectors in the k^{th} sorted position get multiplied by the same parameter $\mathbf{r}_j[k]$. Figure 5 presents an illustration.

Loss Function. We define the *mass* at transformer block j to be the extent to which the sorted positions are retained, i.e.,

$$\text{mass}(j; \mathbf{r}) = \sum_{k=1}^{N} \mathbf{r}_j[k]$$

Our aim is to minimize the aggregate mass over all the transformer blocks with minimal loss in accuracy. Intuitively, the aggregate mass may be viewed as a budget on the total number of positions retained; $\text{mass}(j; \mathbf{r})$ is the breakup across the transformer blocks.

We modify the loss function by incorporating an L_1 regularizer over the aggregate mass. As demonstrated earlier, the transformer blocks have varying influence on the classification output. We scale the mass of each transformer block by its index. Let $\mathbf{\Theta}$ denote the parameters of the baseline BERT model and $\mathcal{L}(\cdot)$ be the loss function (such as

cross entropy loss or mean-squared error) as defined in the original task. We define the new objective function as:

$$\min_{\boldsymbol{\Theta},\mathbf{r}} \left[\mathcal{L}(\boldsymbol{\Theta},\mathbf{r}) \; + \; \lambda \cdot \sum_{j=1}^{L} j \cdot \text{mass}(j;\mathbf{r}) \right]$$
$$\text{s.t.} \;\; \mathbf{r}_j[k] \in [0,1] \;\; \forall (j \in [1,L], k \in [1,N]),$$

where L is the number of transformer blocks. While $\mathcal{L}(\boldsymbol{\Theta},\mathbf{r})$ controls the accuracy, the regularizer term controls the aggregate mass. The hyper-parameter λ tunes the trade-off.

The retention parameters are initialized as $\mathbf{r}_j[k] = 1$, meaning all the sorted positions are fully retained to start with. We train the model to learn the retention parameters. The learned parameter $\mathbf{r}_j[k]$ provides the extent to which the word-vectors at the k^{th} sorted position must be retained. We obtain the retention configuration from the mass of the above parameters: for each transformer block j, set $\ell_j = \texttt{ceil}(\text{mass}(j))$. In the rare case where the configuration is non-monotonic, we assign $\ell_j = \min\{\ell_j, \ell_{j-1}\}$.

3.4. Training PoWER-BERT

Given a dataset, the scheme involves three training steps:

1. *Fine-tuning:* Start with the pre-trained `BERT` model and fine-tune it on the given dataset.
2. *Configuration-search:* Construct an auxiliary model by inserting the `soft-extract` layers in the fine tuned model, and modifying its loss function. The regularizer parameter λ is tuned to derive the desired trade-off between accuracy and inference time. The model consists of parameters of the original `BERT` model and the newly introduced `soft-extract` layer. We use a higher learning rate for the latter. We train the model and derive the retention configuration.
3. *Re-training:* Substitute the `soft-extract` layer by `extract` layers. The number of word-vectors to retain at each transformer block is determined by the retention configuration computed in the previous step. The word-vectors to be retained are selected based on their significance scores. We re-train the model.

In our experiments, all the three steps required only $2-3$ epochs. Inference is performed using the re-trained `PoWER-BERT` model. The `CLS` token is never eliminated and it is used to derive the final prediction.

4. Experimental Evaluation

4.1. Setup

Datasets. We evaluate our approach on a wide spectrum of classification/regression tasks pertaining to 9 datasets from the GLUE benchmark (Wang et al., 2019a), and the IMDB (Maas et al., 2011) and the RACE (Lai et al., 2017)) datasets. The datasets details are shown in Table 1.

Table 1: Dataset statistics: NLI and QA refers to Natural Language Inference and Question Answering tasks respectively. Note that STS-B is a regression task, therefore doesn't have classes.

DATASET	TASK	# CLASSES	INPUT SEQ. LENGTH (N)
COLA	ACCEPTABILITY	2	64
RTE	NLI	2	256
QQP	SIMILARITY	2	128
MRPC	PARAPHRASE	2	128
SST-2	SENTIMENT	2	64
MNLI-M	NLI	3	128
MNLI-MM	NLI	3	128
QNLI	QA/NLI	2	128
STS-B	SIMILARITY	-	64
IMDB	SENTIMENT	2	512
RACE	QA	2	512

Baseline methods. We compare `PoWER-BERT` with the state-of-the-art inference time reduction methods: `DistilBERT` (Sanh et al., 2019b), `BERT-PKD` (Sun et al., 2019b) and `Head-Prune` (Michel et al., 2019b). They operate by removing the parameters: the first two eliminate transformer blocks, and the last prunes attention heads. Publicly available implementations were used for these methods (Sanh et al., 2019a; Sun et al., 2019a; Michel et al., 2019a).

Hyper-parameters and Evaluation. Training `PoWER-BERT` primarily involves four hyper-parameters, which we select from the ranges listed below: a) learning rate for the newly introduced `soft-extract` layers - $[10^{-4}, 10^{-2}]$; b) learning rate for the parameters from the original `BERT` model - $[2 \times 10^{-5}, 6 \times 10^{-5}]$; c) regularization parameter λ that controls the trade-off between accuracy and inference time - $[10^{-4}, 10^{-3}]$; d) batch size - $\{4, 8, 16, 32, 64\}$. Hyper-parameters specific to the datasets are provided in the supplementary material.

The hyper-parameters for both `PoWER-BERT` and the baseline methods were tuned on the Dev dataset for GLUE and RACE tasks. For IMDB, we subdivided the training data into 80% for training and 20% for tuning. The test accuracy results for the GLUE datasets were obtained by submitting the predictions to the evaluation server[1], whereas for IMDB and RACE, the reported results are on the publicly available Test data.

[1] https://gluebenchmark.com

Table 2: Comparison between PoWER-BERT and $\text{BERT}_{\text{BASE}}$. We limit the accuracy loss for PoWER-BERT to be within 1% by tuning the regularizer parameter λ. Inference done on a K80 GPU with batch size of 128 (averaged over 100 runs). Matthew's Correlation reported for CoLA; F1-score for QQP and MRPC; Spearman Correlation for STS-B; Accuracy for the rest.

	Method	CoLA	RTE	QQP	MRPC	SST-2	MNLI-m	MNLI-mm	QNLI	STS-B	IMDB	RACE
Test Accuracy	$\text{BERT}_{\text{BASE}}$	52.5	68.1	71.2	88.7	93.0	84.6	84.0	91.0	85.8	93.5	66.9
	PoWER-BERT	52.3	67.4	70.2	88.1	92.1	83.8	83.1	90.1	85.1	92.5	66.0
Inference Time (ms)	$\text{BERT}_{\text{BASE}}$	898	3993	1833	1798	905	1867	1881	1848	881	9110	20040
	PoWER-BERT	201	1189	405	674	374	725	908	916	448	3419	10110
Speedup		**(4.5x)**	(3.4x)	**(4.5x)**	(2.7x)	(2.4x)	(2.6x)	(2.1x)	(2.0x)	(2.0x)	(2.7x)	(2.0x)

Table 3: Comparison between PoWER-BERT and ALBERT. Here PoWER-BERT represents application of our scheme on ALBERT. The experimental setup is same as in Table 2

	Method	CoLA	RTE	QQP	MRPC	SST-2	MNLI-m	MNLI-mm	QNLI	STS-B
Test Accuracy	ALBERT	42.8	65.6	68.3	89.0	93.7	82.6	82.5	89.2	80.9
	PoWER-BERT	43.8	64.6	67.4	88.1	92.7	81.8	81.6	89.1	80.0
Inference Time (ms)	ALBERT	940	4210	1950	1957	922	1960	1981	1964	956
	PoWER-BERT	165	1778	287	813	442	589	922	1049	604
Speedup		(5.7x)	(2.4x)	**(6.8x)**	(2.4x)	(2.1x)	(3.3x)	(2.1x)	(1.9x)	(1.6x)

Implementation. The code for PoWER-BERT was implemented in Keras and is available at https://github.com/IBM/PoWER-BERT. The inference time experiments for PoWER-BERT and the baselines were conducted using Keras framework on a K80 GPU machine. A batch size of 128 (averaged over 100 runs) was used for all the datasets except RACE, for which the batch size was set to 32 (since each input question has 4 choices of answers).

Maximum Input Sequence Length. The input sequences are of varying length and are padded to get a uniform length of N. Prior work use different values of N, for instance ALBERT uses $N = 512$ for all GLUE datasets. However, only a small fraction of the inputs are of length close to the maximum. Large values of N would offer easy pruning opportunities and larger gains for PoWER-BERT. To make the baselines competitive, we set stringent values of N: we determined the length N' such that at most 1% of the input sequences are longer than N' and fixed N to be the value from $\{64, 128, 256, 512\}$ closest to N'. Table 1 presents the lengths specific to each dataset.

4.2. Evaluations

Comparison to BERT. In the first experiment, we demonstrate the effectiveness of the word-vector elimination approach by evaluating the inference time gains achieved by PoWER-BERT over $\text{BERT}_{\text{BASE}}$. We limit the accuracy loss to be within 1% by tuning the regularizer parameter λ that controls the trade-off between inference time and accuracy. The results are shown in Table 2. We observe that PoWER-BERT offers at least 2.0x reduction in inference time on all the datasets and the improvement can be as high as 4.5x, as exhibited on the CoLA and the QQP datasets.

We present an illustrative analysis by considering the RTE dataset. The input sequence length for the dataset is $N = 256$. Hence, across the twelve transformer blocks, $\text{BERT}_{\text{BASE}}$ needs to process $12 \times 256 = 3072$ word-vectors for any input. In contrast, the retention configuration used by PoWER-BERT on this dataset happens to be $(153, 125, 111, 105, 85, 80, 72, 48, 35, 27, 22, 5)$ summing to 868. Thus, PoWER-BERT processes an aggregate of only 868 word-vectors. The self-attention and the feed forward network modules of the transformer blocks perform a fixed amount of computations for each word-vector. Consequently, the elimination of the word-vectors leads to reduction in computational load and improved inference time.

Comparison to Prior Methods. In the next experiment, we compare PoWER-BERT with the state-of-the-art inference time reduction methods, by studying the trade-off between accuracy and inference time. The Pareto curves for six of the GLUE datasets are shown in Figure 6; others are provided in the supplementary material. Top-left corners correspond to the best inference time and accuracy.

For PoWER-BERT, the points on the curves were obtained by tuning the regularizer parameter λ. For the two transformer block elimination methods, DistilBERT and BERT-PKD, we derived three points by retaining 3, 4, and 6 transformer

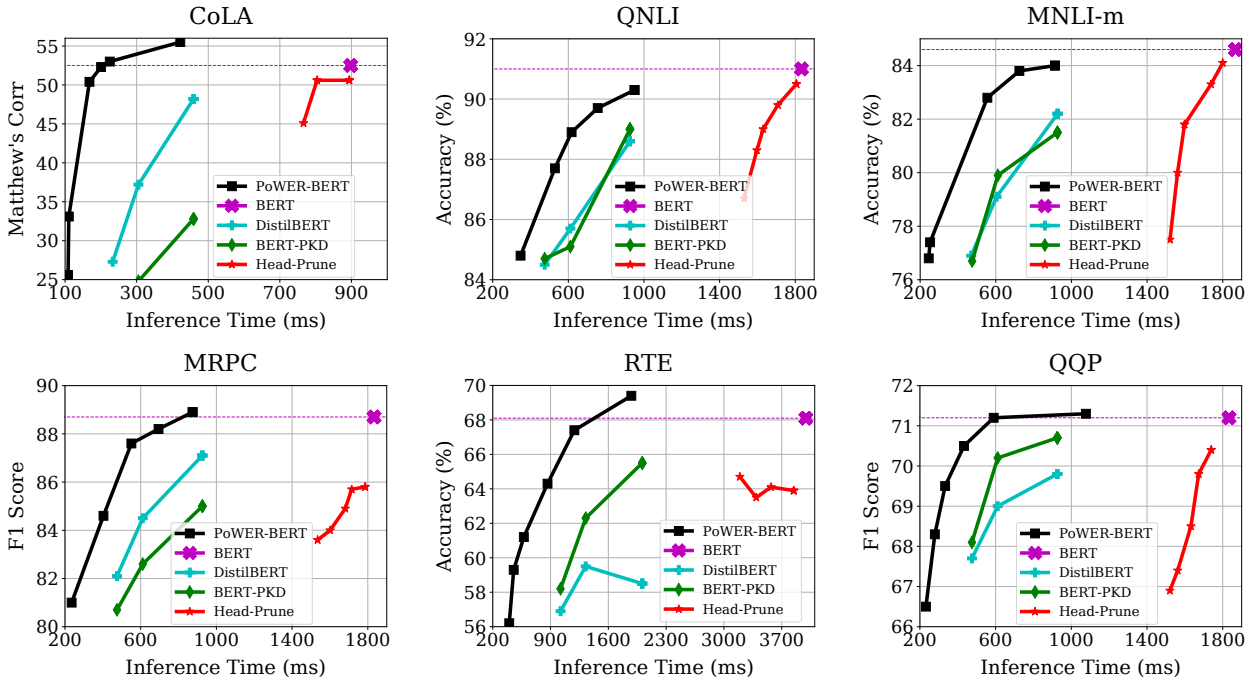

Figure 6: Comparison to prior methods. Pareto curves showing accuracy vs. inference time trade-off. Top-left corners correspond to the best inference time and accuracy. Points for PoWER-BERT obtained by tuning the regularizer parameter λ. For DistilBERT and BERT-PKD, the points correspond to retaining $\{3, 4, 6\}$ transformer blocks. For Head-Prune, points obtained by varying number of retained attention-heads. The cross represents $\text{BERT}_{\text{BASE}}$ performance; dotted line represents its accuracy (for the ease of comparison). Over the best baseline method, PoWER-BERT offers: accuracy gains as high as 16% on CoLA and 6% on RTE at inference time 305 ms and 1326 ms, respectively; inference time gains as high as 2.7x on CoLA and 2x on RTE at accuracy 48.2% and 65.5%, respectively.

blocks; these choices were made so as to achieve inference time gains comparable to PoWER-BERT. Similarly, for the Head-Prune strategy, the points were obtained by varying the number of attention-heads retained.

Figure 6 demonstrates that PoWER-BERT exhibits marked dominance over all the prior methods offering:

- Accuracy gains as high as 16% on CoLA and 6% on RTE for a given inference time.
- Inference time gains as high as 2.7x on CoLA and 2.0x on RTE for a given accuracy.

The results validate our hypothesis that fine-grained word-vector elimination yields better trade-off than coarse-grained transformer block elimination. We also observe that Head-Prune is not competitive. The reason is that the method exclusively targets the attention-heads constituting only 26% of the $\text{BERT}_{\text{BASE}}$ parameters and furthermore, pruning a large fraction of the heads would obliterate the critical self-attention mechanism of BERT.

Accelerating ALBERT. As discussed earlier, word-vector elimination scheme can be applied over compressed models as well. To demonstrate, we apply PoWER-BERT over ALBERT, one of the best known compression methods for BERT. The results are shown in Table 3 for the GLUE datasets. We observe that the PoWER-BERT strategy is able to accelerate ALBERT inference by 2x factors on most of the datasets (with $< 1\%$ loss in accuracy), with the gain being as high as 6.8x on the QQP dataset.

Table 4: Comparison of the accuracy of the word-vector selection methods on the SST-2 Dev set for a fixed retention configuration.

	Head-WS	Rand-WS	Attn-WS
Entire dataset	85.4%	85.7%	88.3%
Input sequence length > 16	83.7%	83.4%	87.4%

Ablation Study. In Section 3.2, we described three methods for word-vector selection: two static techniques, Head-WS and Rand-WS, and a dynamic strategy, denoted Attn-WS, based on the significance scores derived from the attention mechanism. We demonstrate the advantage of Attn-WS by taking the SST-2 dataset as an illustrative example. For all the three methods, we used the same sample retention configuration of $(64, 32, 16, 16, 16, 16, 16, 16, 16)$. The accuracy results are

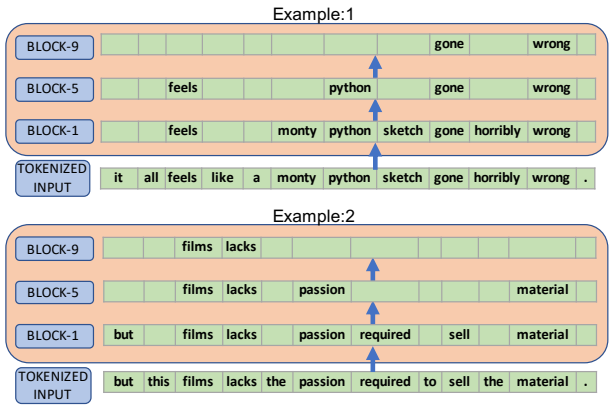

Figure 7: Anecdotal Examples. Real-life examples from SST-2 dataset demonstrating progressive word-vector elimination.

shown in Table 4. The first row of the table shows that `Attn-WS` offers improved accuracy. We perform a deeper analysis by filtering inputs based on length. In the given sample configuration, most transformer blocks retain only 16 word-vectors. Consequently, we selected a threshold of 16 and considered a restricted dataset with inputs longer than the threshold. The second row shows the accuracy results.

Recall that `Head-WS` relies on eliminating as many `PAD` tokens as possible on the average. We find that the strategy fails on longer inputs, since many important word-vectors may get eliminated. Similarly, `Rand-WS` also performs poorly, since it is oblivious to the importance of the word-vectors. In contrast, `Attn-WS` achieves higher accuracy by carefully selecting word-vectors based on their significance. The inference time is the same for all the methods, as the same number of word-vectors get eliminated.

Anecdotal Examples. We present real-life examples demonstrating word-vector redundancy and our word-vector selection strategy based on the self-attention mechanism (`Attn-WS`). For this purpose, we experimented with sentences from the SST-2 sentiment classification dataset and the results are shown in Figure 7.

Both the sentences have input sequence length $N = 12$ (tokens). We set the retention configuration as $(7, 7, 7, 7, 4, 4, 4, 4, 2, 2, 2, 2)$ so that it progressively removes five word-vectors at the first transformer block, and two more at the fifth and the ninth transformer blocks, each.

In both the examples, the first transformer block eliminates the word-vectors corresponding to stop words and punctuation. The later transformer blocks may seem to eliminate more relevant word-vectors. However, their information is captured by the word-vectors retained at the final transformer block, due to the diffusion of information. These retained word-vectors carry the sentiment of the sentence and are sufficient for correct prediction. The above study further reinforces our premise that word-vector redundancy can be exploited to improve inference time, while maintaining accuracy.

5. Conclusions

We presented `PoWER-BERT`, a novel method for improving the inference time of the `BERT` model by exploiting word-vectors redundancy. Experiments on the standard GLUE benchmark show that `PoWER-BERT` achieves up to 4.5x gain in inference time over $\texttt{BERT}_{\texttt{BASE}}$ with $< 1\%$ loss in accuracy. Compared to prior techniques, it offers significantly better trade-off between accuracy and inference time. We showed that our scheme can be applied over `ALBERT`, a highly compressed variant of `BERT`. For future work, we plan to extend `PoWER-BERT` to wider range of tasks such as language translation and text summarization.

References

Correia, G. M., Niculae, V., and Martins, A. F. T. Adaptively sparse transformers. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: 10.18653/v1/d19-1223. URL http://dx.doi.org/10.18653/v1/d19-1223.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and De Freitas, N. Predicting parameters in deep learning. In *NIPS*, 2013.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arxiv preprint: 1503.02531*, 2015.

Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. Compression of deep convolutional neural networks for fast and low power mobile applications. *arxiv preprint: 1511.06530*, 2015.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. BioBERT: Pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 2019.

Liu, X., He, P., Chen, W., and Gao, J. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arxiv preprint: 1904.09482*, 2019.

Liu, Y. and Lapata, M. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *ACL*, 2011.

Martins, A. F. T. and Astudillo, R. F. From softmax to sparsemax: A sparse model of attention and multi-label classification. 2016.

McCarley, J. S. Pruning a BERT-based question answering model. *arxiv preprint: 1910.06360*, 2019.

Michel, P., Levy, O., and Neubig, G., 2019a. URL "https://github.com/pmichel31415/are-16-heads-really-better-than-1".

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*, 2019b.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. *arxiv preprint: 1701.05369*, 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arxiv preprint: 1611.06440*, 2016.

Palakodety, S., KhudaBukhsh, A. R., and Carbonell, J. G. Mining insights from large-scale corpora using fine-tuned language models. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI-20)*, pp. To appear, 2020.

Peters, B., Niculae, V., and Martins, A. F. T. Sparse sequence-to-sequence models. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/p19-1146. URL http://dx.doi.org/10.18653/v1/p19-1146.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T., 2019a. URL "https://github.com/huggingface/transformers/tree/master/examples/distillation".

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019b.

Sau, B. B. and Balasubramanian, V. N. Deep model compression: Distilling knowledge from noisy teachers. *arxiv preprint: 1610.09650*, 2016.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian based ultra low precision quantization of BERT. *arxiv preprint:1909.05840*, 2019.

Sun, S., Cheng, Y., Gan, Z., and Liu, J., 2019a. URL "https://github.com/intersun/PKD-for-BERT-Model-Compression".

Sun, S., Cheng, Y., Gan, Z., and Liu, J. Patient knowledge distillation for BERT model compression. *arXiv preprint arXiv:1908.09355*, 2019b.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019a.

Wang, Z., Wohlwend, J., and Lei, T. Structured pruning of large language models. *arxiv preprint: 1910.04732*, 2019b.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. Q8BERT: Quantized 8bit BERT. *arxiv preprint:1910.06188*, 2019.

Zhao, G., Lin, J., Zhang, Z., Ren, X., Su, Q., and Sun, X. Explicit sparse transformer: Concentrated attention through explicit selection. *arXiv,1912.11637*, 2019.