DRONE: Data-aware Low-rank Compression for Large NLP Models

Patrick H. Chen
UCLA
Los Angels, CA
patrickchen@g.ucla.edu

Hsian-fu, Yu
Amazon
Palo Alto, CA
rofu.yu@gmail.com

Inderjit S. Dhillon
UT Austin & Amazon
Palo Alto, CA
inderjit@cs.utexas.edu

Cho-jui, Hsieh
UCLA & Amazon
Los Angels, CA
chohsieh@cs.ucla.edu

Abstract

The representations learned by large-scale NLP models such as BERT have been widely used in various tasks. However, the increasing model size of the pre-trained models also brings efficiency challenges, including inference speed and model size when deploying models on mobile devices. Specifically, most operations in BERT consist of matrix multiplications. These matrices are not low-rank and thus canonical matrix decompositions do not lead to efficient approximations. In this paper, we observe that the learned representation of each layer lies in a low-dimensional space. Based on this observation, we propose DRONE (**d**ata-awa**r**e l**o**w-ra**n**k compr**e**ssion), a provably optimal low-rank decomposition of weight matrices, which has a simple closed form solution that can be efficiently computed. DRONE can be applied to both fully-connected and self-attention layers appearing in the BERT model. In addition to compressing standard models, our method can also be used on distilled BERT models to further improve the compression rate. Experimental results show that DRONE is able to improve both model size and inference speed with limited loss in accuracy. Specifically, DRONE alone achieves 1.92x speedup on the MRPC task with only 1.5% loss in accuracy, and when DRONE is combined with distillation, it further achieves over 12.3x speedup on various natural language inference tasks.

1 Introduction

The representations learned by large-scale Natural Language Processing (NLP) models such as BERT and its variations have been widely used in various tasks [8, 2, 25, 3, 24]. The successes of these large NLP models rely on the usage of large corpus and big models. Indeed, researchers have reported better results with models that have more parameters [31] and number of layers [1]. The increasing model size of the pre-trained models inhibits public users from training a model from scratch, and it also brings forth efficiency challenges, including inference speed and model size when deploying models on mobile devices.

To deal with efficiency issues, most existing work resorts to adjusting the model structure or distillation. For instance, [17] used locality-sensitive hashing to accelerate dot-product attention, [18] used repeating model parameters to reduce the size and [44] applied a pre-defined attention pattern to save computation. A large body of prior work focused on variants of distillation has also been explored

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

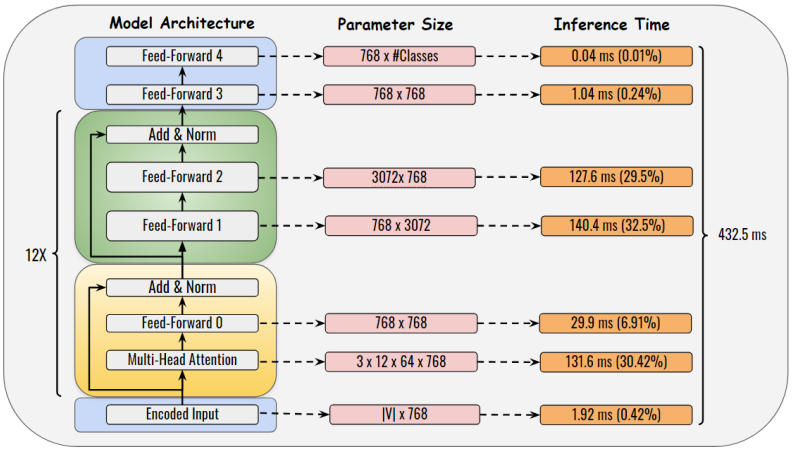


Figure 1: Illustration of the BERT-base computational model. $|V|$ (at bottom of the Parameter Size column) denotes the number of tokens in the model. #Classes (at top of the Parmeter Size column) denotes the number of classes in the down-stream classification task. Input encoding, Feed-forward 3 and Feed-forward 4 are computed only once and thus do not contribute much to overall time. The inference time (in milliseconds) listed here is based on the inference time measured on a CPU.

[27, 15, 35, 20, 34, 35, 41, 45, 4]. These methods require a specific design of model architecture, or a long training stage and thus it is less straightforward to combine these methods with each other.

In this paper, we explore a simpler acceleration method to speed up inference time which can be applied to most existing architectures. As shown in Figure 1, matrix multiplication (feed-forward layer) is a fundamental operation which appears many times in the Transformer [36], the backbone architecture of the BERT model. In fact, the underlying computation of both multi-head attention layers and feed-forward layers is matrix multiplication. Therefore, instead of resorting to the complex architecture redesign approaches, we aim to investigate whether low-rank matrix approximation, a classical and simple model compression approach, can be used to accelerate Transformers. Despite its successful application to CNNs [42, 33, 32], at first glance, low-rank compression does not appear to work for BERT since the matrices in both feed-forward layers and attention layers **are not low rank** (see Figure 2). Therefore, even the optimal low-rank approximation (e.g., by SVD) will lead to very large reconstruction error. This is probably why low-rank approximation has not been successfully used in BERT compression.

In this paper, we propose a novel low-rank approximation algorithm to compress the weight matrices even though they are not low-rank. The main idea is to **exploit the data distribution**. In NLP applications, the latent features (features fed into each matrix mulitplication layer) usually indicate some information extracted from natural sentences, and they often lie in a subspace with a low intrinsic dimension [5, 32, 22]. Therefore, in most of the matrix-vector products, even though the weight matrices are not low-rank, the input vectors lie in a low-dimensional subspace, allowing dimension reduction with minimal degraded performance. We mathematically formulate this generalized low-rank approximation problem which includes the data distribution term and provide a closed-form solution for the optimal rank-k decomposition of the weight matrices. By leveraging the data distribution idea, we propose DRONE (**d**ata-awa**r**e l**o**w-ra**n**k compr**e**ssion). Our decomposition significantly outperforms the SVD under the same rank constraint, and can successfully accelerate the BERT model without sacrificing too much test performance. In addition to compressing standard models, DRONE can also be used on distilled BERT models to further improve the compression rate. For example, DRONE alone achieves 1.92x speedup on the MRPC task with only 1.5% loss in accuracy, and when combined with distillation, DRONE achieves over 12.3x speedup on various natural language inference tasks.

2 Related Work

Fast inference is important for deploying NLP models in various applications. Generally speaking, inference efficiency can be enhanced by hardware [30] or lower-level instruction optimization [23]. On the other hand, the main focus of the current research is on using algorithmic methods to reduce computational complexity. These methods can be mainly categorized into two aspects: attention complexity reduction and model size reduction.

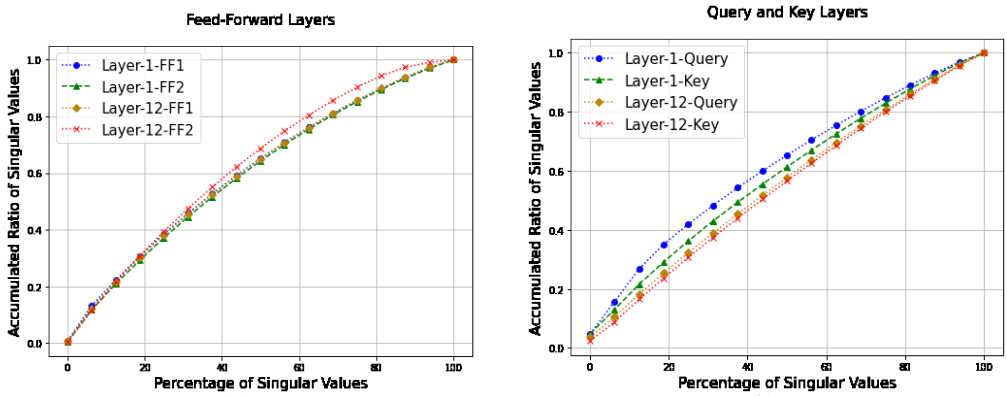


Figure 2: Illustration of the empirical observation that weight matrices in BERT model are not low-rank. The X-axis represents what percentage of singular values; the Y-axis represents sum of singular values connected to the selected ranks divided by sum of all singular values. Ideally, a low-rank structure will have a larger area under the curve, meaning that a small percentage of the singular values can explain their total sum. We observe that the sum of the top 50% of the ranks only accounts about 60% of all singular values for matrices in the BERT model. This shows that the matrices do not have a clear low-rank structure.

Attention Complexity Reduction

Attention mechanism is the building block of transformer models and has attracted the most attention of researchers recently in the NLP field [36]. Pre-training on large corpus of BERT, a transformer-based model, has contributed to state-of-the-art performance on various tasks after fine-tuning [8]. Attention on sequences of length L is $O(L^2)$ in both computational and memory complexity, which yields long inference time when the sequence is long. Thus, researchers have focused on reducing the complexity of the attention module. [17] used locality-sensitive hashing to reduce the complexity to $O(L \log L)$. [44, 7] pre-defined an attention map to have a constant computational time. [11] progressively eliminated the redundant context vectors within the attended sequence to improve efficiency of attention in the last few layers of the model. [38] proposed to train the low-rank attention by choosing a rank $r \ll L$. This is similar to our work in the sense of leveraging low-rank structures. But our method does not require training the model from scratch and can be applied to different modules other than attention. In fact, most of the above methods require special modules and thus need to train the proposed models from scratch. This prohibits the usage of a large body of publicly available open models for faster research progress. More importantly, these methods mainly focus on the long sequence scenario. As shown in Figure 1, we have found out that attention module is actually not the main inference bottleneck of inference time in common usage. In most, if not all, models of common usages, two layers of large feed-forward layer are appended after the attention module which incurs much more computational time. Attention complexity reduction only works when a long sequence is used but in current practice this is unusual. Thus, in many tasks accelerating the attention module itself does not contribute to a significant reduction of overall inference time.

Model Size Reduction

Inference speed is also related to model compression. In principle, smaller models lead to reduction in the number of operations and thus faster inference time. [28] explored pruning methods on BERT models to eliminate redundant links, and there is a line of research on pruning methods [12, 13, 6, 10]. Quantization methods [43, 14, 19, 9] convert the 32 bits float models into fewer-bits fixed-point representation and make model prediction faster with fixed point accelerator. [18] reduce the model size by sharing encoder parameters. A large body of prior work focused on variants of knowledge distillation [27, 15, 35, 20, 34, 35, 41, 45, 4]. These methods use different strategies to distill information from a teacher network and reduce the number of layers [27] or hidden dimension size [15]. Further, a hybrid compression method by combining matrix factorization, pruning and knowledge distillation is proposed by [21]. Notice that [21] performed SVD for some components and in this paper we propose an improvement over SVD by leveraging input distribution to each layer. Idea of using input distribution to compress model has also been explored in PCN method

[37], which is perhaps the closest to our work. However, DRONE differs from PCN in following three aspects. First, PCN only considers input distribution but not weight matrix and thus it's a special case of DRONE (i.e., W be an identity matrix in equation (3)). Second, PCN merely does dimension reduction whereas our formulation achieves dimension reduction and low-rank approximation simultaneously. Last, PCN does not guarantee the obtained transformation is the optimal; whereas, DRONE formulates an approximation optimization problem and we provide the optimal solution. Other forms of low-rank learning strategies including initialization and structure pruning were also explored in the literature [39, 16], and we will compare to these baseline methods. Among the above-mentioned methods, quantization requires hardware accelerator to maximally reduce the inference time. Pruning methods can only reduce the model size, but the inference time might not be reduced due to the limitation of sparse operations. Only algorithmic methods such as distillation serve as a more generic inference time accelerating method. We want to emphasize that our method is orthogonal to these distillation methods. In fact, the proposed method is an acceleration method that is applicable to all components in most NLP models. In Section 4, we show that DRONE can be combined with the distilled models to further improve the performance.

3 Proposed Method

We now introduce an algorithm for improving efficiency of matrix multiplication. The computation of feed-forward (FF) layer in the attention models can be described as:

$$h = Wx + b, \tag{1}$$

$$o = \sigma(h), \tag{2}$$

where $W \in \mathbb{R}^{d_2 \times d_1}$ and $b \in \mathbb{R}^{d_2}$ are model parameters, $x \in \mathbb{R}^{d_1}$ is the latent representation of a token, and $h \in \mathbb{R}^{d_2}$ is the intermediate representation before the activation function, $\sigma(\cdot)$ is the activation function, and $o \in \mathbb{R}^{d_2}$ is the output. Assuming the sequence length is L, all the token representations $x_1, \ldots, x_L \in \mathbb{R}^{d_1}$ will pass through this same operation, so in practice the whole FF layer can be computed by a matrix-matrix product $W[x_1, \ldots x_L] + b$, and the computation of the bias term b would be broadcast to all L input tokens. In practice we will normally have $L \ll \max(d_1, d_2)$ (e.g., $L = 128$, $d_2 = 3072$). Notice that applying $\sigma(\cdot)$ on h element-wisely costs $O(Ld_2)$, which is much smaller than the cost of computing Wx ($O(Ld_2d_1)$). Therefore, in this paper we focus on reducing the cost of computing Wx to accelerate the computation. A standard way to accelerate this computation is to perform low-rank approximation on W. A low-rank approximation can be obatined by using singular value decomposition (SVD), which achieves the best rank-k approximation in terms of Frobenius norm and we can write W as:

$$W = USV^T \approx U_{W,k} {V_{W,k}}^T,$$

with orthogonal matrices $U \in \mathbb{R}^{d_2 \times d_2}$, $V \in \mathbb{R}^{d_1 \times d_1}$ and a diagonal matrix $S \in \mathbb{R}^{d_2 \times d_1}$. $U_{W,k} \in \mathbb{R}^{d_2 \times k}$ and $V_{W,k} \in \mathbb{R}^{d_1 \times k}$ are the rank-k approximation matrices by taking $U_{W,k} = US_k^{\frac{1}{2}}$, $V_{W,k} = VS_k^{\frac{1}{2}}$, where $S_k^{\frac{1}{2}}$ is the square-root of the first k entries of the diagonal matrix S. Given such an approximation, we can simplify the computation in (1) by

$$h = Wx + b \approx U_{W,k} {V_{W,k}}^T x + b.$$

After conducting rank-k approximation, the computational complexity reduces from $O(d_2d_1)$ to $O((d_1+d_2)k)$. When k is small enough, low-rank approximation not only accelerates the computation [32] but also compresses the model size [26]. However, as shown in Figure 2, matrices in FF layer of BERT do not show obvious low-rank structures. We observe that choosing top 50% rank (e.g., $k = 0.5 \min(d_1, d_2)$) can only achieve around 60% of the accumulation ratio of singular values, which implies large matrix approximation error. In the meantime, the complexity is still about $O(d_2d_1)$ and there is no enhancement of speed.

Even though the matrices in the model are not low-rank, we now provide an illustrative example to show that a low-rank computation could still exist when data distribution lies in a lower intrinsic dimension. Suppose we have a matrix W defined as below and the input x lies in a subspace:

$$W = \begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}, \quad x \in \text{span}\left(\begin{bmatrix} 2 \\ 2 \\ 5 \\ 5 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 6 \end{bmatrix} \right).$$

In this case, W is a full-rank matrix so there is no lossless low-rank approximation of W. On the other hand, the input data x lies in a 2-dimensional subspace so that we could construct the following low-rank approximation:

$$\underbrace{\begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x}$$

$$= \underbrace{\begin{bmatrix} 43 & 23 \\ 90 & 39 \\ 66 & 41 \\ 45 & 37 \\ 29 & 21 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} -1 & -1 & 0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0.25 \end{bmatrix}}_{V^T} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x},$$

which gives a rank-2 matrix UV^T where $W \neq UV^T$ but $Wx = UV^Tx$ for any x in the low dimensional space. This shows that even if W cannot be approximated, it is still possible to construct a good low-rank decomposition, and the key is to exploit the space of input vectors.

3.1 DRONE: Data-aware Low-rank Compression

Assuming the input x of the FF layer follows some distribution, instead of minimizing the approximation error of the weight matrix (for which SVD is optimal), we want to minimize the approximation error of the outputs. Denoting X as the $\mathbb{R}^{d_1 \times n}$ matrix where columns of X capture the empirical distribution of the input (when n is large), our goal is to find projection matrix $V_{X,k} \in \mathbb{R}^{d_1 \times k}$ and recovery matrix $U_{X,k} \in \mathbb{R}^{d_2 \times k}$ such that the output is well approximated. We rewrite (1) as:

$$\begin{aligned} h = WX + b &\approx WU_{x,k}{V_{x,k}}^T X + b \\ &= (WU_{x,k}){V_{x,k}}^T X + b = W_{X,k}{V_{x,k}}^T X + b, \end{aligned}$$

where $W_{X,k} = WU_{x,k}$. Intuitively, when X lies in a lower-dimensional space, we could find such a pair by PCA decomposition on X to project X onto the subspace that explains the most variance. In this way, instead of considering the decomposition of W, we leverage the distribution of X to complete the low-rank approximation.

However, the best way is to consider the properties of both W and X simultaneously, and we can mathematically present this desideratum by the following optimization problem:

$$\min_M \|WX - WMX\|_F^2, \quad \text{s.t.} \quad \text{rank}(M) = k, \tag{3}$$

where M is the desired rank-k transformation which maximally preserves the results of the matrix multiplication. In the theorem below, we show that there exists a closed-form, optimal solution for the above optimization problem. Before stating the theorem, we first introduce some notation. Assuming $\text{rank}(W) = r$ and $\text{rank}(X) = t$, we can write $W = U_W S_W V_W^T$ and $X = U_X S_X V_X^T$ such that

$$U_W = \begin{bmatrix} U_{W,r} & \bar{U}_{W,r} \end{bmatrix}, S_W = \begin{bmatrix} S_{W,r} & 0 \\ 0 & 0 \end{bmatrix}, V_W = \begin{bmatrix} V_{W,r} & \bar{V}_{W,r} \end{bmatrix}$$

$$U_X = \begin{bmatrix} U_{X,t} & \bar{U}_{X,t} \end{bmatrix}, S_X = \begin{bmatrix} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}, V_X = \begin{bmatrix} V_{X,t} & \bar{V}_{X,t} \end{bmatrix}.$$

In other words, the decomposition $U_W S_W V_W^T$ and $U_X S_X V_X^T$ are the full-SVD decompositions of W and X, respectively. The matrices $U_{W,r}$,$V_{W,r}$, $U_{X,t}$,$V_{X,t}$ denote corresponding row spaces and column spaces, while $\bar{U}_{W,r}$, $\bar{V}_{W,r}$, $\bar{U}_{X,t}$ and $\bar{V}_{X,t}$ are null spaces. With this notation, we are ready to state the theorem.

Theorem 1. *Assume rank(W) = r and rank(X) = t. The closed form solution M^* of the optimization problem (3) is*

$$M^* = V_{W,r} S_{W,r}^{-1} Z_k S_{X,t}^{-1} U_{X,t}^T, \tag{4}$$

where Z_k is the rank-k truncated SVD of $Z = S_{W,r} V_{W,r}^T U_{X,t} S_{X,t}$.

The proof of Theorem 1 is provided in Appendix A. We note that since Z_k is the rank-k truncated SVD of Z, we could also write Z_k as $U_{Z,k}V_{Z,k}^T$ by distributing the top-k singular values of Z into left or right singular matrices. Thus the original computation can be rewritten as:

$$WX \approx (WV_{W,r}S_{W,r}^{-1}U_{Z,k})(V_{Z,k}^T S_{X,t}^{-1}U_{X,t}^T)X = U^* V^{*T} X, \tag{5}$$

where $U^* = WV_{W,r}S_{W,r}^{-1}U_{Z,k}$ and $V^{*T} = V_{Z,k}^T S_{X,t}^{-1}U_{X,t}^T$ are two rank-k matrices, and we will replace W by U^*V^{*T}.

3.2 Extension to Dot-product Attention

Although the optimization problem in (3) is proposed for feed-forward computation, in this section we show that it can also be applied to the dot-product part of the attention module. The key computation in the attention layer is to compute pairwise similarity between queries and keys of the sequence:

$$O = (Q\bar{Y})^T(KY), \tag{6}$$

where $\bar{Y} \in \mathbb{R}^{d_1 \times n}$ is the batch query data, $Q \in \mathbb{R}^{d_2 \times d_1}$ is the query transformation matrix, $Y \in \mathbb{R}^{d_1 \times m}$ is the batch key data, $K \in \mathbb{R}^{d_2 \times d_1}$ is the key transformation matrix and n, m are query and key batch sizes, respectively. We can again see that the desired low-rank approximation is the solution of the following optimization problem:

$$\min_M \|(Q\bar{Y})^T(KY) - (Q\bar{Y})^T M(KY)\|_F^2, \text{s.t. rank}(M) = k. \tag{7}$$

With $Q\bar{Y} = W$ and $K\bar{Y} = X$, we get the following corollary from Theorem 1 directly.

Corollary 1. *Assume rank($Q\bar{Y}$) $= r$ and rank(KY) $= t$. Let $Q\bar{Y} = U_W S_W V_W^T$ and $KY = U_X S_X V_X^T$ be the SVD decomposition of $Q\bar{Y}$ and KY respectively. The closed form solution M^* of the optimization problem (7) is given by*

$$M^* = V_{W,r}S_{W,r}^{-1}Z_k S_{X,r}^{-1}U_{X,r}^T, \tag{8}$$

where Z_k is the rank-k truncated SVD of $Z = S_{W,r}V_{W,r}^T U_{X,t}S_{X,t}$.

3.3 Overall Algorithm

We have shown that the proposed DRONE method is a generic acceleration module applicable to all parts of neural language models. We summarize the DRONE on feed-forward layer in Algorithm 1. Since in practice we don't have the exact distribution of X, we use training data to calculate the low-rank approximations as described in Algorithm 1. The attention map calculation can be done by the same procedure with $W = (Q\bar{Y})^T$ and $X = K\bar{Y}$ as given by the Corollary 1.

To accelerate the whole model, we need to select appropriate ranks for each component. However, since the approximation of one component affects the distribution of overall representations, the optimal rank for the model requires a complete search of all possible combinations of rank values, which is infeasible in practice. We thus resort to a simplified approach as shown in Algorithm 2. A more detailed description is provided in the Appendix B. In short, as the changes of lower layer parameters will cause the distribution of representation shifts in upper layers, we approximate each component one-by-one in their topological order of the model. In another words, we approximate the model from the lower layers toward the higher layers. Within each layer, we follow the topological order of underlying modules. We provide a total allowed increase of loss ratio r as an input to the Algorithm 2. The hyper-parameter r depends on the efficiency and efficacy trade-off which users are willing to pay. The larger the value r, the faster approximation we get at the cost of lower accuracy. We then distribute r into each module $R_{l,i}$ (allowed loss increase ratio of i-th module of l-th layer in Algorithm 2). The distribution from r to each $R_{l,i}$ is based on the observed inference time of each module $E_{l,i}$ (observed empirical inference time of i-th module of l-th layer in Algorithm 2). The longer a module takes to compute, the more budget is allocated. Overall, total allowed loss r and the distributed loss ratio for each module $R_{l,i}$ fulfil the equality $(1 + r) = \prod_l \prod_i (1 + R_{l,i})$. For each module, if the approximation with certain rank used won't increase the loss over the ratio (1 + $R_{l,i}$), we will use that rank to approximate the module and move on to the next module. The pseudo code is also provided in the Appendix to illustrate the process.

Algorithm 1 Data-Aware Low-rank Compression of feed-forward layer.

Input: rank k, training data D_{train}, Original weight matrix W, Prediction Model M.
Output: Low-rank Approximation U^*, V^*.
X = { }
for all batches x_b in D_{train} **do**
Feed the batch of training data x_b into M and extract the representation x. x is the representation which will be multiplied with W as in (1).
Append x to X.
end for
Given X,k and W, solve the optimal low-rank matrices U^*,V^* by (5).

Algorithm 2 Overall Low-rank Model Approximation Algorithm

Input: training data D_{train}, original weight matrix W. prediction Model M, total allowed loss increase ratio r, Observed inference time E, Search grids of ranks for each module G, original Training loss L.
Output: Low-rank Model $\hat{M}$.

Distribute allowed ratio r into each module by E
$E_{min} \leftarrow \arg\min_{l,i} E_{l,i}$
$E_{l,i} \leftarrow \frac{E_{l,i}}{E_{min}}$
$E_b \leftarrow exp(\frac{log(1+r)}{\sum_{l,i} E_{l,i}})$
$R_{l,i} \leftarrow E_b^{E_{l,i}} - 1$
for $l = 1, \cdots,$ total layers **do**
for each module $m_i \in M_l$ **do**
$W_{l,i} \leftarrow l$-th layer parameter of module m_i
(e.g., 2nd feed-forward matrix in first layer.)
for $i = 1, \cdots, |G_{l,i}|$ **do**
$k \leftarrow G_{l,i}$
$U, V \leftarrow$ Algorithm 1 $(k, D_{train}, W_{l,i}, M)$
$\hat{M} \leftarrow M$ with $W_{l,i}$ replaced by U, V.
Evaluate new loss $L_{new} = \hat{M}(D_{train})$
if $L_{new}/L < 1 + R_{l,i}$ **then**
$M \leftarrow \hat{M}$
break;
end if
end for
end for

4 Experimental Results

4.1 Experimental Setup

We evaluate DRONE on both LSTM and transformer-based BERT models. For LSTMs, we train a 2-layer LSTM-based language model from scratch with hidden sizes 1500 on Penn Treebank Bank (PTB) dataset. For BERT models, we evaluate the pre-trained BERT models on GLUE tasks. Various pre-trained models are offered in the open source platform [40]. For BERT models, we use BERT-base models and it contains 12 layers of the same model structure without sharing parameters. Each layer contains an attention module with hidden size 768 and 12 channels, a small 768×768 Feed-forward (FF) layer followed by 2 larger FF layers (768×3072 and 3072×768). As shown in Figure 1, these four components consume the most computational time in the BERT-base models.

For the baseline methods, most of the existing work pertaining to low-rank approximation [21, 32] leverages SVD in part of the compression procedure. Therefore, our baseline comparison will be the SVD approximation, and our work aims to provide an improvement over SVD. We also include the state-of-the-art distillation methods TinyBERT [15] in the comparison and show that the proposed method can be combined with it to further improve the performance. TinyBERT reduces the model

Table 1: The experimental results of running pret-rained BERT-base model on natural language inference tasks (Glue dataset). Each task has its own metric for performance measurement. Accuracy (SST-2, QNLI, RTE and WNLI), F1/Accuracy (MRPC and QQP), Matthew's correlation (CoLA), Matched accuracy/Mismatched accuracy (MNLI) and Person/Spearman correlation (STS-B) are used respectively. All the DRONE results are within 3% accuracy loss and show that DRONE can accelerate the whole BERT model across different tasks and devices.

Methods	MNLI	QQP	SST-2	QNLI	MRPC	RTE	CoLA	STS-B
Original	84.3	90.9	92.3	91.4	89.5	72.6	53.4	87.8
SVD	74.4	50.8	73.1	52.2	63.8	47.3	12.4	33.6
DRONE	82.0	89.4	90.0	88.5	86.7	70.0	52.5	85.8
DRONE-Retrain	82.6	90.1	90.8	89.3	88.0	71.5	53.2	87.8
CPU Speedup Ratio	1.60x	1.25x	1.64x	1.20x	1.92x	1.31x	1.33x	1.52x
GPU Speedup Ratio	1.28x	1.38x	1.45x	1.28x	1.56x	1.33x	1.29x	1.57x

into 4 layers of attention dimension 312 with 12 channels, and the FF layers are downsized to 312×1200. As we mentioned above, all the approximation methods need to consider efficiency and efficacy trade-off. In this paper, we follow previous literature [15, 5] and report the approximation results with about 3% loss in accuracy to compare the performance of all methods.

In real-world applications, NLP models are mostly evaluated on mobile devices or servers with multiple hardware accelerators. Thus, we measure the inference speed on both CPU (Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz) and GPU (GeForce GTX 1080 Ti) devices. All the experiments are repeated 10 times. The average single sequence prediction inference time in milliseconds is reported in the results. We want to emphasize that unlike many of the literature [38, 17, 7], which reported speedup only in the attention layers, our results reflect end-to-end speedup including both attention module and feed-forward layers. To perform the approximation, empirically we found randomly sub-sample 10% of the training data suffices to provide good results. Using more data can only provide limited performance boost but comes at a higher cost of longer preprocessing time. Thus, we will use 10% random sample of the training data to perform the experiments. After the proposed data-aware low-rank distribution, we slightly fine-tune the model to further improve the performance. We use a relatively smaller learning rate 10^{-7} and retrain 1 epoch on the sub-sampled training data to complete the fine-tuning procedure.

4.2 Results of BERT Models on GLUE Dataset

We summarize the results of DRONE on GLUE tasks in Table 1. Detailed inference time of each component of the compressed Transformer model is listed in the Appendix. Detailed inference time of an uncompressed BERT-BASE model can be found in Figure 1. We observe that each task exhibits different difficulty. The best acceleration we can achieve is nearly twice as fast (1.92x) with less than 2% accuracy loss after retraining (on the MRPC). In addition, DRONE achieves 1.52x acceleration without accuracy loss on the STS-B task. By applying the same selected rank for each module with SVD method, we can observe that the performance drops significantly. This shows that the matrices within the model is generally not low-rank; thus the direct low-rank approximation without considering data distribution does not work. On GPU, we see that the acceleration is more or less the same as on CPU except MNLI and MRPC tasks. This is due to the fact that GPU uses massive parallelism and low-rank approximation introduces a sequential computation which might hinder the speedup depending on the size of the matrices and ranks used. To resolve this problem, low-level cuda code optimization is needed and system researchers have studied the problem [29], which is out of the scope of the present work. Despite this, we can still observe that DRONE performs better on QQP, RTE and STS-B and it provides about 1.5x acceleration for various tasks on GPU. An example of ranks used in SST-2 is listed in Appendix F.

4.3 Combination with Model Size Reduction Methods

Our proposed DRONE is a general low-rank approximation technique, and it is complementary to many other model compression methods. To illustrative its power, we now demonstrate that DRONE can be combined together distillation. The discussion of combination with Quantization is left in the Appendix. Distillation methods compress the underlying model into a smaller one without losing much accuracy. Distilled models are much smaller in number of layers or hidden dimension, resulting in a smaller model size and faster inference time. As shown in the Table 2, TinyBERT, one of the most competitive distillation methods, indeed achieves good performance within 3% accuracy loss for some of the GLUE tasks. Due to the fact that the computation inside the distilled model is still

full matrix computation, DRONE can be applied to find data-aware low-rank approximation of these smaller matrices. Results are summarized in Table 2. As we can see combining DRONE with the distillation method further reduces the inference time without sacrificing accuracy. In particular, on the SST-2 task DRONE + TinyBERT speeds the inference time from 11.7x to 15.3x on CPU while achieving the same accuracy as the TinyBERT. Similarly, DRONE + TinyBERT speedups GPU results with 10.9x STS-B and 9.7x on SST-2 with competitive performance. These results again show that the proposed method has the potential to be applied under various scenarios and hardware devices to achieve a better model inference time speedup.

Table 2: The average inference time (in milliseconds) in comparison to distilled models on CPU and GPU. The unit is in millisecond. The results show that DRONE can be combined with distillation to further improve the performance. Compared to the state-of-the-art distillation method, the speedup ratio increases from 11.4x to 14.2x on STS-B and from 11.7x to 15.3x on SST-2.

Tasks	Models	CPU-speedup	GPU-speedup	Accuracy (%)
STS-B	BERT	1x	1x	87.8
	TinyBERT	11.4x	8.6x	86.9
	DRONE +TinyBERT	**14.2x**	**10.9x**	**87.0**
RTE	BERT	1x	1x	72.6
	TinyBERT	1.8x	1.9x	70.8
	DRONE +TinyBERT	**2.1x**	**2.2x**	**71.7**
MRPC	BERT	1x	1x	89.5
	TinyBERT	11.6x	7.8x	86.3
	DRONE +TinyBERT	**12.3x**	**8.6x**	**86.7**
SST-2	BERT	1x	1x	92.3
	TinyBERT	11.7x	8.4x	**90.7**
	DRONE +TinyBERT	**15.3x**	**9.7x**	**90.7**

Table 3: Illustration of SVD fine-tuning on MRPC, RTE, CoLA and STS-B. Using the same rank as the proposed DRONE method, SVD accuracy will drop significantly after the approximation. After fine-tuning done on the SVD approximation, the accuracy could be recovered for some tasks (e.g., MRPC), but SVD + Retrain still perform much worse than DRONE across all the tasks.

Models	MRPC	RTE	CoLA	STS-B
BERT	89.5	72.6	53.4	87.8
DRONE-Retrain	**88.0**	**71.5**	**53.2**	**87.8**
SVD	63.8	47.3	12.4	33.6
SVD-Retrain	85.8	63.5	24.4	66.3

4.4 Comparison to Structured Pruning Methods.

Instead of compressing model after training to accelerate inference time, another line of research called Structured Pruning tried to learn the low-rank structure during training of the model to save both training and inference time simultaneously. This raises the question if post-processing such as DRONE is necessary if no further compression is required once we can get a small model after training. Thus, it's worth comparing DRONE with the state-of-the-art Structure Pruning method [39]. In [39], attention modules are not approximated by low-rank matrices. To make the comparison fair, we apply DRONE on base models except the attention module and keep others the same. For MRPC, DRONE achieves 1.58x speedup with performance drop from 89.5 to 89.4. [39] achieves 1.43x with performance 88.61. For SST-2, DRONE achieves 1.41x speedup without sacrificing performance (92.3). [39] also achieves about 1.41x speedup with the performance 92.09. Thus, we can see that DRONE performs better than structured pruning.

One further question is that if we can use the idea of DRONE to do fast training? We conducted DRONE on the pre-trained RTE task before fine-tuning to get a low-rank structure, and then apply the regular end-to-end training over this compressed model. We found out this procedure with directly using the same rank as in our experiments (with 1.38x speedup) can only achieve 68.2 accuracy. But if we reduce the compression ratio into 1.2x training time speedup, this procedure can give us 72.9 accuracy. On the other hand, the same procedure with SVD as the initilialization of low-rank structure can only get 52.1 accuracy. This preliminary experiment shows that in addition to inference time acceleration, DRONE also has the potential to be applied in the training, but directly transport

DRONE into training can not lead to the optimal result. How to improve low-rank training is an interesting future direction.

4.5 Additional Experiments on Large-Scale Models and Language Generation Tasks

Concerns might be raised that if DRONE can also be generalized to other scenarios such as larger models or other NLP tasks. To validate DRONE on larger models. We conduct experiments on RTE dataset with BERT-LARGE model. BERT-LARGE doubled number of layers and the dimension is increased from 768 to 1024. Average inference time for a data increased to 1405ms and it achieves accuracy 74. The overall result is 72.9 with inference time 1018ms (1.38x speedup). This result is comparable to our BERT-BASE result (1.31x speedup). To validate DRONE on other NLP tasks, We conducted the method on the machine translation task via OpenNMT. It provides a 2-layer transformer model on en-de translation. On the transformer part, DRONE achieves 1.76x speedup with BLEU from 33.47 to 33.26. Through these two additional experiments, we can validate that the proposed DRONE is generic in the sense that so long as the underlying model is composed of matrix computation, DRONE can compress the model regardless of model sizes and target tasks.

4.6 Can we directly learn low-rank structures by end-to-end training?

From an optimization perspective, a natural question to ask is whether the same optimal low-rank structure could be learned by end-to-end fine-tuning once the rank is decided. We conduct experiments on 4 tasks to verify this, and the results are summarized in Table 3. We start by performing DRONE on the task to achieve the desired accuracy, and perform SVD with the same set of ranks. Accuracy of SVD drops significantly for all tasks. We then fine-tune hyper-parameters as in [40][1] to fine-tune the above SVD results. After fine-tuning, the accuracy improves across all tasks, but none of it can reach the same performance as DRONE. This shows that due to the difficulty of optimizing a non-convex objective function, fine-tuning the SVD result may not achieve the best low-rank result. On the other hand, the proposed DRONE method under the the optimization problem (3) can obtain the provably optimal low-rank approximation at a much lower computational cost than the fine-tuned SVD.

4.7 Pre-processing of DRONE is not too costly

DRONE accelerates inference speed at the cost of a pre-processing step. Thus, It's natural to ask if DRONE will take long pre-processing time. Given the rank, prep-rocessing of DRONE has a one-time distribution extraction plus low-rank solving of equation (3). Depending on training data size, first stage takes 2 mins (RTE) to 20 mins (MNLI). Second stage has 2 SVD computations and is about 5-10 mins. The matrix size involved is limited as we subsample training data. SVD costs about 3 mins and retrain costs from 5 mins to 2 hours depending on training data size. Thus, pre-processing of DRONE is about the same order as SVD but with much better performance. Distillation such as TinyBERT firstly has a general distillation of BERT-base on large data used to train original BERT, followed by task-specific distillation. Despite task-specific one is rather fast (15 mins to 2hrs), first stage takes a few days for a single GPU. Overall, DRONE is not costly compared to other methods.

5 Conclusions

In this paper, we propose DRONE, a data-aware low-rank approximation, to achieve a better low-rank approximation in BERT models. DRONE leverages the fact that data distribution in NLP tasks usually lies in a lower-dimensional subspace. By considering the data distribution, we propose a data-aware low-rank approximation problem and provide a closed-form solution. Empirical results validate that DRONE can significantly outperform the vanilla-SVD method, and can achieve at least 20% acceleration with less than 3% accuracy loss. When DRONE is combined with distillation methods, it further achieves up to 15.3 times acceleration with less than 2% accuracy loss.

[1] https://huggingface.co/transformers/v2.1.1/examples.html#glue

6 Acknowledgement

We would like to thank the anonymous reviewers for their helpful comments. This work is supported in part by NSF under IIS-1901527, IIS-2008173, IIS-2048280.

References

[1] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3159–3166, 2019.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[4] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. AdaBERT: Task-adaptive BERT compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020.

[5] Patrick H Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. Learning to screen for fast Softmax inference on large vocabulary neural networks. *arXiv preprint arXiv:1810.12406*, 2018.

[6] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. *arXiv preprint arXiv:2007.12223*, 2020.

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019.

[10] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

[11] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh M Raje, Venkatesan T Chakaravarthy, Yogish Sabharwal, and Ashish Verma. PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination.

[12] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR*, abs/1510.00149, 2015.

[13] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. *CoRR*, abs/1506.02626, 2015.

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[15] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[16] Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolo Fusi. Initialization and regularization of factorized neural layers. *arXiv preprint arXiv:2105.01029*, 2021.

[17] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[18] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. AlBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[19] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[20] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. FastBERT: a Self-distilling BERT with Adaptive Inference Time. *arXiv preprint arXiv:2004.02178*, 2020.

[21] Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Yaming Yang, Quanlu Zhang, Yunhai Tong, and Jing Bai. LadaBERT: Lightweight Adaptation of BERT through Hybrid Model Compression. *arXiv preprint arXiv:2004.04124*, 2020.

[22] Jeff Mitchell and Mirella Lapata. Language models based on semantic composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 430–439. Association for Computational Linguistics, 2009.

[23] Emma Ning. Microsoft open sources breakthrough optimizations for transformer inference on GPU and CPU . tinyurl.com/y26jdsn9, 2020. Accessed: 2010-09-30.

[24] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[25] 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.

[26] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[27] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[28] Victor Sanh, Thomas Wolf, and Alexander M Rush. Movement Pruning: Adaptive Sparsity by Fine-Tuning. *arXiv preprint arXiv:2005.07683*, 2020.

[29] Achal Shah and Angshul Majumdar. Accelerating low-rank matrix completion on gpus. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 182–187. IEEE, 2014.

[30] Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2018.

[31] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pages 10414–10423, 2018.

[32] Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. SVD-softmax: Fast softmax approximation on large vocabulary neural networks. In *Advances in Neural Information Processing Systems*, pages 5463–5473, 2017.

[33] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[34] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. *arXiv preprint arXiv:1908.09355*, 2019.

[35] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

[36] 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.

[37] Roger Waleffe and Theodoros Rekatsinas. Principal component networks: Parameter reduction early in training. *arXiv preprint arXiv:2006.13347*, 2020.

[38] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-Attention with Linear Complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[39] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019.

[40] Thomas Wolf, L Debut, V Sanh, J Chaumond, C Delangue, A Moi, P Cistac, T Rault, R Louf, M Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*, 2019.

[41] Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. BERT-of-theseus: Compressing BERT by progressive module replacing. *arXiv preprint arXiv:2002.02925*, 2020.

[42] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[43] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: Quantized 8bit BERT. *arXiv preprint arXiv:1910.06188*, 2019.

[44] Biao Zhang, Deyi Xiong, and Jinsong Su. Accelerating neural transformer via an average attention network. *arXiv preprint arXiv:1805.00631*, 2018.

[45] Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. Extreme language model compression with optimal subwords and shared projections. *arXiv preprint arXiv:1909.11687*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] . See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] . Results are mostly in Table format so it's not appropriate to show error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix for:
Data-Aware Low-Rank Compression for Large NLP Models

A Proof of Theorem 1

Theorem 1. *Assume rank(W) = r and rank(X) = t. The closed form solution M^* of the optimization problem in equation 3 is*

$$M^* = V_{W,r} S_{W,r}^{-1} Z_k S_{X,t}^{-1} U_{X,t}^T, \tag{9}$$

where Z_k is the rank-k truncated SVD of $Z = S_{W,r} V_{W,r}^T U_{X,t} S_{X,t}$.

Proof. We firstly consider the unconstrained problem:

$$\begin{aligned} M^* &= \arg\min_M \|WX - WMX\|_F^2 \\ &= \arg\min_M \|U_W^T WXV_X - U_W^T WMXV_X\|_F^2 \\ &= \arg\min_M \|S_W V_W^T U_X S_X - S_W V_W^T M U_X S_X\|_F^2, \end{aligned}$$

where the second equality holds due to the fact that U_W and V_X are orthonormal matrices. Note that we could expand the term $S_W V_W^T U_X S_X$ as:

$$\begin{aligned} S_W V_W^T U_X S_X &= \begin{bmatrix} S_{W,r} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{W,r}^T \\ \bar{V}_{W,r}^T \end{bmatrix} \begin{bmatrix} U_{X,t} & \bar{U}_{X,t} \end{bmatrix} \begin{bmatrix} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} S_{W,r} V_{W,r}^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}. \end{aligned}$$

Similarly, we will have

$$S_W V_W^T M U_X S_X = \begin{bmatrix} S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}.$$

Therefore, we continue above unconstrained problem as:

$$\begin{aligned} M^* &= \arg\min_M \|S_W V_W^T U_X S_X - S_W V_W^T M U_X S_X\|_F^2 \\ &= \arg\min_M \| \begin{bmatrix} S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix} \|_F^2 \\ &= \arg\min_M \|S_{W,r} V_{W,r}^T U_{X,t} S_{X,t} - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t}\|_F^2. \\ &= \arg\min_M \|Z - S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t}\|_F^2. \end{aligned}$$

The above minimization problem obtains the optimal value if $S_{W,r} V_{W,r}^T M U_{X,t} S_{X,t}$ equals the rank-k truncated SVD of Z by the fundamental property of SVD decomposition. Thus, we will have:

$$\begin{aligned} Z_k &= S_{W,r} V_{W,r}^T M^* U_{X,t} S_{X,t} \\ \implies M^* &= V_{W,r} S_{W,r}^{-1} Z_k S_{X,t}^{-1} U_{X,t}^T. \end{aligned}$$

□

B An algorithm to Search of Ranks under DRONE

The input to Algorithm 2 consists of training data, the model with all parameters of weight matrices and original training loss. In addition, a pre-defined search grid is also necessary. Taking $W \in R^{768\times768}$ as an example, we can perform a grid search for a proper low rank k over $[1, 768]$ such as $\{96, 192, 288, 384, \ldots, 768\}$. The finer the grid, the more compressed model we could get at the cost of longer running time of the DRONE method. With these input parameters, we firstly distribute the total allowed loss into each individual module. We then iteratively apply Algorithm 1 following the computational sequence illustrated in Figure 1. For each module, we search the rank k by going through the grid. If the approximated result will not increase the allowed loss increase ratio of the component, we will end the search and tie the found rank to the component and move on. The procedure will continue until all components are compressed. The whole process could guarantee us that the final loss L' of the compressed model $\hat{M}$ would not be greater than $(1 + r)L$, where L is the original loss before approximation.

Algorithm 2 Overall Low-rank Model Approximation Algorithm

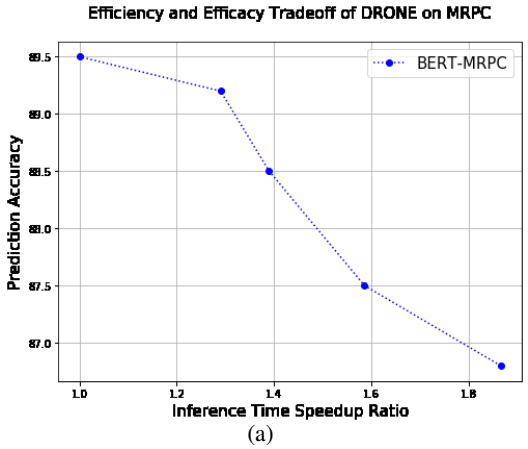
Input: training data D_{train}, original weight matrix W. prediction Model M, total allowed loss increase ratio r, Observed inference time E, Search grids of ranks for each module G, original Training loss L.

Output: Low-rank Model $\hat{M}$.

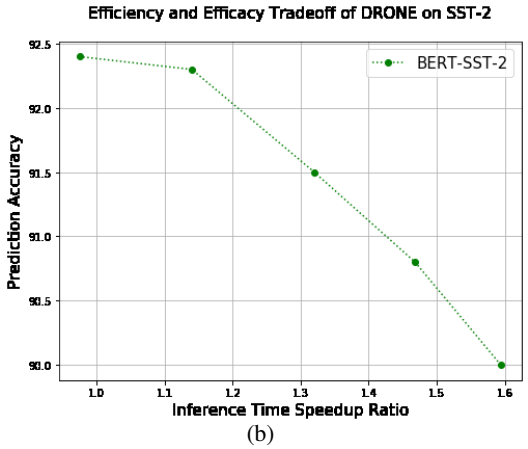
Distribute allowed ratio r into each module by E
$E_{min} \leftarrow \arg\min_{l,i} E_{l,i}$
$E_{l,i} \leftarrow \frac{E_{l,i}}{E_{min}}$
$E_b \leftarrow exp(\frac{log(1+r)}{\sum_{l,i} E_{l,i}})$
$R_{l,i} \leftarrow E_b^{E_{l,i}} - 1$
for $l = 1, \cdots$, total layers **do**
  for each module $m_i \in M_l$ **do**
    $W_{l,i} \leftarrow l$-th layer parameter of module m_i
    (e.g., 2nd feed-forward matrix in first layer.)
    for $i = 1, \cdots, |G_{l,i}|$ **do**
      $k \leftarrow G_{l,i}$
      $U, V \leftarrow$ Algorithm 1 $(k, D_{train}, W_{l,i}, M)$
      $\hat{M} \leftarrow M$ with $W_{l,i}$ replaced by U, V.
      Evaluate new loss $L_{new} = \hat{M}(D_{train})$
      if $L_{new}/L < 1 + R_{l,i}$ **then**
        $M \leftarrow \hat{M}$
        break;
      end if
    end for
  end for

C Efficiency and Efficacy Trade-off Graph

In this paper, we mentioned that we report the result of 3% accuracy drop as the performance of the baseline methods and DRONE. However, as we mentioned above that all the approximation methods need to consider efficiency and efficacy trade-off. 3% is chosen according to the literature. Here, we show two exemplar graph on MRPC and SST-2 task to demonstrate two facts. First, it's indeed a trade-off between the efficiency and efficacy as the speedup ratio goes higher at the cost of lower accuracy. Second, we want to point out that this trade-off relationship is not linear, and different task might have different characteristics. Thus, in the real application, users need to decide what's the best cutoff to use. We also want to point out that this 3% accuracy drop comparison is fair to all baseline methods. We could have chose another cutoff like 1% accuracy with lower speedup ratio to report, but this won't help too much when comparing different baseline methods.



(a)



(b)

Figure 3: Illustration of efficiency and efficacy trade-off. Each point in this graph represents a specific ratio of training loss increase after approximation.

D Detailed results

D.1 LSTM result

A 2-layer LSTM model is composed of two large matrices layers and one large softmax layer. Additional processing time includes applying activation functions, softmax function and computing the updated hidden representation. The detailed inference time in each layer is summarized in Table 4. We could observe that the overhead of the computation will be greatly incurred on GPU. Thus, despite the matrix is much smaller and well approximated by DRONE, the overall acceleration on GPU is less.

D.2 Transformer result

For BERT models, we use BERT-base models and it contains 12 layers of the same model structure without sharing parameters. Each layer contains an attention module with hidden size 768 and 12 channels, a small 768×768 Feed-forward (FF) layer followed by 2 larger FF layers (768×3072 and 3072×768). As shown in Figure 1, these four components consume the most computational time in the BERT-base models. The detailed average inference time of each module is summarized in Table 5 for CPU and Table 6 for GPU. There are two important points to note.

Firstly, we could see that attention module is not the bottleneck at all under the normal size of context (128). Therefore, many works on accelerating attention module alone would not improve the overall inference time of the module except a very long sequence appears. The necessity of the long sequence is out of the domain of this paper and what we want to show is that the proposed DRONE would work on both attention and feed-forward layer, which collectively could accelerate the real(overall) inference time.

Secondly, we could observe that the FF2 layer could be accelerated most. A plausible reason could be that the input dimension to the FF2 layer is in a larger dimension (3072) than all the other layers (64 or 768). When the input distribution actually lies in a lower-dimensional space, there is much more room for FF2 layer to be compressed and accelerated by the data-aware low-rank method.

Table 4: The average inference time of each component in the model of 2-layer LSTM model. Both proposed methods and SVD use same ranks so the inference time is approximately the same. The unit is in millisecond and the number in parenthesis shows the ratio respective to the overall inference time.

Device	Models	LSTM-1	LSTM-2	Softmax	Others	Total Time	Perplexity
CPU	PTB-Large	1.27ms	1.30ms	1.09ms	0.13ms	3.79ms	78.32
	PTB-Large-SVD	-	-	-	-	-	81.09
	PTB-Large-SVD-Retrain	-	-	-	-	-	80.89
	PTB-Large-DRONE	-	-	-	-	-	80.87
	PTB-Large-DRONE-Retrain	0.24ms	0.34ms	0.42ms	0.11ms	1.11ms(3.4x)	79.01
GPU	PTB-Large	0.019ms	0.018ms	0.015ms	0.32ms	0.11ms	78.32
	PTB-Large-SVD	-	-	-	-	-	81.09
	PTB-Large-SVD-Retrain	-	-	-	-	-	80.89
	PTB-Large-DRONE	-	-	-	-	-	80.87
	PTB-Large-DRONE-Retrain	0.01ms	0.01ms	0.015ms	0.055ms	0.09ms(1.2x)	79.01

Table 5: The detailed average inference time (in milliseconds) on CPU of each component in the model by retrained DRONE.

Tasks	Self-Attention	Feed-Forward 0	Feed-Forward 1	Feed-Forward 2	Others	Total Time
MNLI	122.7	19.5	78.5	46.1	4.2	271.0
QQP	131.5	29.9	99.2	66.5	5.8	333.0
SST-2	100.5	24.7	79.3	54.5	4.5	263.5
QNLI	128.3	28.4	111.0	79.0	5.9	352.6
MRPC	82.6	12.8	89.4	38.2	2.4	225.4
RTE	116.0	25.6	85.4	62.3	3.4	292.7
CoLA	108.2	22.7	93.1	70.8	3.4	298.2
STS-B	109.1	19.3	90.8	53.0	4.0	276.2

Table 6: The detailed average inference time (in milliseconds) on GPU of each component in the model by retrained DRONE.

Tasks	Self-Attention	Feed-Forward 0	Feed-Forward 1	Feed-Forward 2	Others	Total Time
MNLI	0.94	0.26	0.76	0.60	0.003	2.56
QQP	0.92	0.24	0.64	0.52	0.001	2.32
SST-2	0.85	0.25	0.59	0.52	0.005	2.22
QNLI	0.91	0.25	0.72	0.60	0.001	2.48
MRPC	0.89	0.22	0.57	0.43	0.001	2.11
RTE	1.02	0.29	0.60	0.59	0.008	2.51
CoLA	0.93	0.25	0.68	0.61	0.002	2.47
STS-B	0.83	0.2	0.57	0.42	0.002	2.02

E Combination with Quantization Methods

Distillation in practice achieves the STOA without extra hardware accelerator, so it serves as a good target to show how DRONE can be combined with other methods. The benefit of quantization/pruning can only be shown when a ASIC/FPGA accelerator is provided. Since we don't have one, we can't only use the software to simulate. We can apply any Quantization scheme and empirically show combined method can achieve a competitive accuracy with lower bit bandwidth. Algorithmically, we combine DRONE with vanilla quantization with fixed precision which gets 87.5 (vs 89.5 on MRPC) and 51.0 (vs 53.4 on CoLA) with 12 bits(vs 32 bits). Thus we can hypothesize that with the hardware accelerator, there could be at least further 3x speedup when DRONE is combined with Quantization methods.

F An example of ranks used in SST-2

Below are the ranks obtained by performing DRONE in SST-2 dataset. The order is from the bottom layer to the top layer. Full rank of all the matrices are 768.

Attention layer: [192, 384, 192, 768, 768, 192, 768, 768, 192, 192, 768, 192],

Feed-Forward 0 : [288, 768, 96, 192, 288, 192, 768, 768, 96, 96, 288, 96],

Feed-Forward 1: [96, 96, 768, 768, 288, 288, 768, 768, 96, 96, 288, 96],

Feed-Forward 2 : [192, 192, 768, 288, 768, 768, 768, 192, 96, 192, 96, 96].

G Python Pseudo Code for Solving equation (3)

Listing 1: The python function to solve the equation (3).

```
import numpy as np

def OPTsolver(x,y,k):
    '''
     compute the best rank k projection M such that \| x*y' - x*M*y'\|_{F}
     is minimized
     x \in shape n x d
     y \in shape m x d

    '''
    xSS = np.matmul(x.transpose(),x)
    kSS = np.matmul(y.transpose(),y)
    U1,S1,V1 = np.linalg.svd(xSS,False)
    S1 = S1 ** 0.5
    I1 = np.eye(S1.shape[0])
    U2,S2,V2 = np.linalg.svd(kSS,False)
    S2 = S2 **0.5
    I2 = np.eye(S2.shape[0])
    YK = np.dot(np.dot(I1*S1,V1),np.dot(V2.transpose(),I2*S2))
    U,S,V = np.linalg.svd(YK,False)
    L = np.dot(V1.transpose(),I1*(1/S1))
    R = np.dot(I2*(1/S2),V2 )
    M = np.dot(U[:,:k]*S[:k],V[:k,:])
    return L,R,U,S,V
```

H Python Pseudo Code of Rank Searching

Listing 2: A mixed of real code and pseudo code to illustrate the search algorithm.

```
import os
import numpy as np
import torch
import subprocess as sp

cuda_num = 7
n_heads = 12
total_layer = 12

prev_loss = .11159391902588509 # Initial Loss
the_model_name = 'bertSST2'

time_attn = 117.5 # Empirical Inference Time on Attention Module
time_0 = 34.27 # Empirical Inference Time on Attention FFL Module
time_1 = 133.11 # Empirical Inference Time on Feedforward 1 layer
time_2 = 128.84 # Empirical Inference Time on Feedforward 2 layer
minimal_time = min(time_attn,time_0,time_1,time_2)
multiplier = (time_attn+time_0+time_1+time_2)/(minimal_time)
tolerant = 2. # allowed loss increase ratio. $r$ in Algorithm 2.

# Code to Distribute the $r$ into individual Modules.
# The distribution depends on empirical inference time of each module and number of layers.
basic_tolerance = np.exp(np.log(tolerant)/multiplier)
tol_attn = np.exp(np.log(basic_tolerance**(time_attn/minimal_time))/n_layer)
tol_0 = np.exp(np.log(basic_tolerance**(time_0/minimal_time))/n_layer)
tol_1 = np.exp(np.log(basic_tolerance**(time_1/minimal_time))/n_layer)
tol_2 = np.exp(np.log(basic_tolerance**(time_2/minimal_time))/n_layer)
```

```
#### Omitted Code ###
# This part of the code is to change some parameters of the underlying hugginface framework
in order to extract the training distribution X of each module from the model.
### Omitted Code ###



for i in range(total_layer):
    for each module in the layer: # This line is pseudo code for clarity reason.
        # This part of the code extracts $R_{l,i}$(named the_tol here) in Algorithm 2.
        if save_symbol == "E":
            the_tol = tol_attn
        elif save_symbol == "F0":
            the_tol = tol_0
        elif save_symbol == "F1":
            the_tol = tol_1
        else:
            the_tol = tol_2
        # Update the allowed increase of loss
         prev_loss = prev_loss * the_tol

        # initial search rank for Attention(16) and FFL layers(96)

        rank = 16 if save_symbol == "E" else 96

        # Maximal rank
        tps = 64 if save_symbol == "E" else 768 specified in the original models.
        while rank <= tps:
            ### Omitted Code ###
            ## Write the tried rank into hugginface framework##
            ### Omitted Code ###

            # This line run the inference in the command line
            os.system(python run_glue.py --model_type bert --task_name SST-2)


            with open('/tmp/tmp0','r') as file:
                data = file.readlines()
            new_r = float(data[-1])
            if  new_r < prev_loss:
                break
            if save_symbol == "E": # Attention module, we increase search rank 16 at a time.
                rank += 16
            else:
                #rank += 96 # For FFL layer, we increase search rank 96 a time.
                if rank ==  384:
                    rank = 768
                    break
                else:
                    rank += 96

        ### Omitted Code ###
        # This part of code update the model #
```