sensors

Article

No Fine-Tuning, No Cry: Robust SVD for Compressing Deep Networks

Murad Tukan [1,*,†], Alaa Maalouf [1,†], Matan Weksler [2] and Dan Feldman [1]

1 The Robotics and Big Data Lab, Department of Computer Science, University of Haifa, Haifa 3498838, Israel; amaalouf@campus.haifa.ac.il (A.M.); dfeldman@univ.haifa.ac.il (D.F.)
2 Samsung Research Israel, Herzliya 4659071, Israel; m.weksler@samsung.com
* Correspondence: mtukan@campus.haifa.ac.il; Tel.: +972-543298724
† These authors contributed equally to this work.

Abstract: A common technique for compressing a neural network is to compute the k-rank ℓ_2 approximation A_k of the matrix $A \in \mathbb{R}^{n\times d}$ via SVD that corresponds to a fully connected layer (or embedding layer). Here, d is the number of input neurons in the layer, n is the number in the next one, and A_k is stored in $O((n+d)k)$ memory instead of $O(nd)$. Then, a fine-tuning step is used to improve this initial compression. However, end users may not have the required computation resources, time, or budget to run this fine-tuning stage. Furthermore, the original training set may not be available. In this paper, we provide an algorithm for compressing neural networks using a similar initial compression time (to common techniques) but without the fine-tuning step. The main idea is replacing the k-rank ℓ_2 approximation with ℓ_p, for $p \in [1,2]$, which is known to be less sensitive to outliers but much harder to compute. Our main technical result is a practical and provable approximation algorithm to compute it for any $p \geq 1$, based on modern techniques in computational geometry. Extensive experimental results on the GLUE benchmark for compressing the networks BERT, DistilBERT, XLNet, and RoBERTa confirm this theoretical advantage.

Keywords: matrix factorization; neural networks compression; robust low rank approximation; Löwner ellipsoid

check for updates

Citation: Tukan, M.; Maalouf, A.; Weksler, M.; Feldman, D. No Fine-Tuning, No Cry: Robust SVD for Compressing Deep Networks. *Sensors* **2021**, *21*, 5599. https://doi.org/10.3390/s21165599

Academic Editor: Mario Luca Fravolini

Received: 26 July 2021
Accepted: 16 August 2021
Published: 19 August 2021

1. Introduction

Deep learning revolutionized machine learning by improving the accuracy by dozens of percents for fundamental tasks in natural language processing (NLP), speech/image recognition, etc. One of the disadvantages of deep learning is that in many cases, the classifier is extremely large compared to classical machine learning models. A large network usually requires expensive and stronger resources due to: (1) slower classification time, which may be a serious limitation, especially in real-time systems such as autonomous cars or real-time text/speech translations; (2) a large memory requirement, which makes it infeasible to store the network on RAM or on a device such as IoT/smartphones; and (3) high energy consumption which is related to the CPU/GPU time of each classification and requires larger batteries with shorter lifespan.

Pipeline of network compression. Given training data P, a common pipeline to obtain a compressed network consists of the following stages:

(i) Train a network N based on the training set P, starting from an initial random network.
(ii) Compress the network N to a small network $\tilde{N}$. The input P may be not involved in this stage.
(iii) Fine-tune the weights of $\tilde{N}$ by training it on P. This step aims to improve the accuracy of the network $\tilde{N}$ but does not change its size.

In this paper, our goal is to improve the compression step (ii) in order to avoid the fine-tuning step (iii) via suggesting a better and more robust compressing scheme. We suggest a novel low rank factorization technique for compressing an embedding layer of a given NLP model. This is motivated by the fact that in many networks, the embedding layer accounts for 20–40% of the network size. Indeed, the results are easily extended to fully connected layers.

1.1. Embedding Matrix

One of the most common approaches for compressing neural networks is to treat a layer in the network as a matrix operation and then to approximate this matrix by its compressed version. This is especially relevant in a fully connected layer. Specifically, in word embedding, this layer is called the embedding layer, which is defined by the following matrix.

The input of the embedding layer consists of d input neurons, and the output has n neurons. The nd edges between these layers define a matrix $A \in \mathbb{R}^{n\times d}$. Here, the entry $A_{i,j}$ in the ith row and jth column of A is equal to the weight of the edge between the jth input neuron to the ith output neuron. Suppose that a test sample (vector) $x \in \mathbb{R}^d$ is received as an input. The corresponding output n-dimensional vector is thus $y = Ax$. To simply, a column from A during training is read, and a standard vector x (a column of the identity matrix) is used and is called a one-hot vector.

ℓ_2 k-rank approximation. One of the natural and common matrix approximations, including in the context of network compression, is the ℓ_2 k-rank approximation (see, e.g., [1,2] and references therein). This is the matrix which minimizes the Frobenius norm, i.e., the sum of squared distances $\|A - A_k\|_F^2 := \sum_{i=1}^n \left\| A^{(i)} - A_k^{(i)} \right\|_2^2$ between the ith row $A^{(i)}$ in A and its corresponding row $A_k^{(i)}$ in A_k, over every rank k matrix A_k. It can be easily computed via the singular value decomposition (SVD) in $O(\min\{nd^2, dn^2\})$ time. Although A_k has the same size as A, due to its low rank, it can be factorized as $A_k = UW$, where $U \in \mathbb{R}^{n\times k}$ and $W \in \mathbb{R}^{k\times d}$. We can then replace the original embedding layer that corresponds to A by a pair of layers that correspond to U and W, which can be stored using $O(k(n+d))$ memory, compared to the $O(nd)$ entries in A. Moreover, the computation of the output $y_k := A_k x$ takes $O(k(n+d))$ time, compared to the $O(nd)$ time that it takes to compute Ax.

Handling other linear layers. The rank approximation technique can be also applied to a fully connected layer, where an activation function $f : \mathbb{R}^n \to \mathbb{R}$ is applied on the output Ax or each of its coordinates (as Relu) to obtain $f(Ax)$. By approximating A, in a sense, $f(Ax)$ is also approximated by $f(A_k x)$. Then, A_k is replaced by two smaller layers U, W, as explained above. Furthermore, it is known that convolutional layers (tensors) can be viewed as fully connected layers (matrix multiplication) applied to reshaped volumes of the input. Then, one can approximate the convolutional weights by approximating its corresponding weight matrix. Hence, the rank approximation technique can be also applied to a convolutional layer.

1.2. Motivation

In what follows, we explain the main motivation of this paper, which in sum, aims to eliminate the need for the fine-tuning step due to the reasons explained in Section 1.2.1. We also discuss the weaknesses of the known SVD factorization in Section 1.2.2, which in turn, give rise to the motivation behind our approach discussed in Section 1.2.3.

1.2.1. Fine-Tuning

The layers that correspond to the matrices U and W above are usually used only as initial seeds for a training process that is called fine-tuning, where the aim is to improve the initial results. Here, the training data are fed into the network, and as opposed to the ℓ_2 error, the error is measured with respect to the final classification, i.e., in the fine-tuning step, the compressed network $\tilde{N}$ is trained using the input P, similar to Step (i). The goal of this step is to improve the accuracy of the compressed network without increasing its size. Hence, the structure of the data remains the same, but the edges are updated in each iteration.

To be or not to be fine-tuned? Fine-tuning is a necessary step to recover the generalization ability damaged by the model compression. Despite its widespread use, fine-tuning is vaguely understood, e.g., what fraction of the pre-trained weights are actually changing and why? [3].

In many cases, the fine-tuning cannot be applied:

1. The original (large) training set is not necessarily available for us (e.g., for sake of data privacy) to apply the fine-tuning after compressing the model.
2. For large datasets and complex tasks, the fine-tuning process takes a very long time and requires strong resources [4,5], even on the pruned networks. Hence, due to the limited computational power (and/or insufficient training time) of the end user device (e.g., smartphone, IoT), fine-tuning is not a viable choice.
3. In the context of NLP, it is common to learn representations of natural language [6–9] via full-network pre-training followed by fine-tuning on a small dataset for the sake of learning a new task [10–12]. However such pre-trained models are very large. Thus, a natural coping mechanism would involve compression before the fine-tuning. After the compression, the model suffers from loss in its original learning capability, and unfortunately, the fine-tuning process is not sufficient to both retain the model's quality and make the network learn a new task, since we may not be able to obtain enough tagged information that we can rely on to perform meaningful training from scratch, e.g., when compressing the embedding layer, we may lose the richness of the vocabulary, as it is responsible for representing each word from a vocabulary by a vector that reflects its semantic and syntactic information which can be extracted from the language.

Hence, some have attempted to prune each layer independently, by which a fine-tuning process can be done with a small number of epochs to avoid the excessive computational power required by the fine-tuning process [5]. Finally, it is worth mentioning that the fine-tuned parameters are not constrained to share any components with the pre-trained weights and thus are equally expensive to store and to compute per iteration [13].

In this paper, we replace the "go-to" method for compression models using matrix factorization by a more robust low rank approximation scheme, where the emphasis here is that the learning capability of the model after the compression is less affected.

1.2.2. Should We Use SVD?

Training the network and compressing it are natural steps. However, it is not clear that the last fine-tuning step, which may be a serious time consumer, is necessary. The goal of this work is to remove this step by improving the previous (compression) step via more involved algorithms that provably approximate the more robust ℓ_p rank approximation. We begin with geometric intuition.

The geometry behind SVD. Geometrically, each row of A corresponds to a d-dimensional vector (point) in $\mathbb{R}^d$, and the corresponding row in A_k is its projection on a k-dimensional subspace of $\mathbb{R}^d$. This subspace (which is the column space of U) minimizes the sum of squared distances to the rows of A over every k-subspace in $\mathbb{R}^d$.

Statistically, if these n points were generated by adding a Gaussian noise to a set of n points on a k-dimensional subspace, then it is easy to prove that most likely (in the sense of maximum-likelihood) this subspace is U. The disadvantage of ℓ_2 k-rank approximation is that it is optimal under the above statistical assumption, which rarely seems to be the case for most applications. In particular, minimizing the sum of squared distances is heavily sensitive to outliers [14] (see Figure 1). As explained in [15], this is the result of squaring each term, which effectively weights large errors more heavily than small ones.

This undesirable property, in many applications, has led researchers to use alternatives such as the mean absolute error (MAD), which minimizes the ℓ_1 (sum of distances) of the error vector. For example, compressed sensing [16] uses ℓ_1 approximation as its main tool to clean corrupted data [17] as well as to obtain sparsified embeddings with provable guarantees as explained, e.g., in [18].

In machine learning, the ℓ_1-approximation replaces or is combined with the ℓ_2 approximation. Examples in scikit-learn include lasso regression, elastic-nets, or MAD error in decision trees [19].

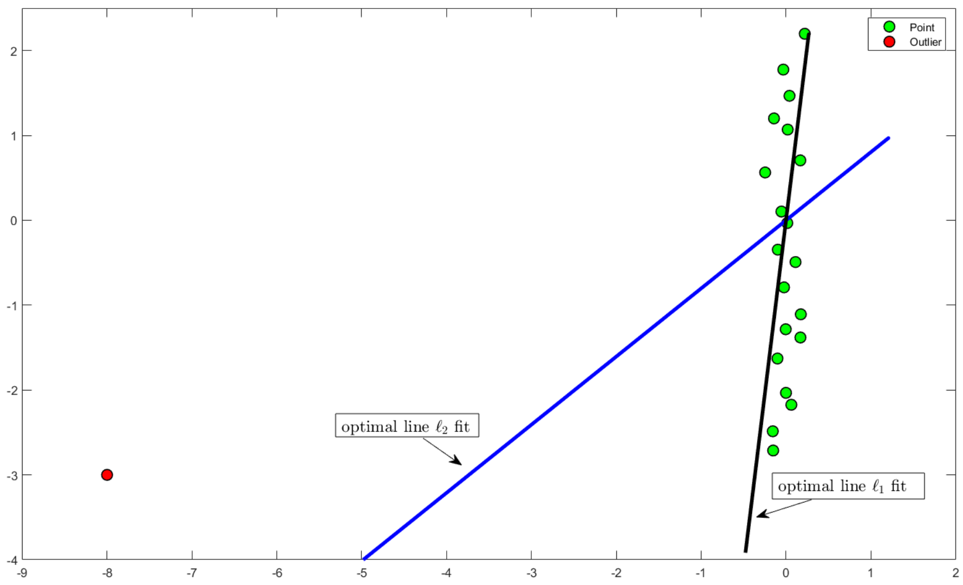


Figure 1. ℓ_1-low rank approximation versus ℓ_2-low rank approximation. Since the norm of a vector increases as the base of the norm decreases, the optimization problem becomes less susceptible towards outliers in the data as presented above.

1.2.3. Novel Approach and Its Challenges

Novel approach: deep learning meets subspace approximation. We suggest generalizing the above ℓ_2 approximation to ℓ_1 k-rank approximation, or even ℓ_p approximation for more general $p < 2$. Geometrically, we wish to compute the k-subspace that minimizes the sum of pth power of the distances to the given set of n points. This should result in more accurate compressed networks that are more robust to outliers and classification mistakes.

Unlike the case $p = 2$, which was solved more than a century ago [20] via SVD and its variants, the ℓ_p low rank approximation was recently proved to be NP-hard even to approximate up to a factor of $1 + 1/\text{poly}(d)$ (recall that d is the number of columns of A above) for $p \in [1, 2)$ [21] and even for general (including constant) values of p (see Section 2). In the most recent decade, there was great progress in this area; however, the algorithms were either based on ad hoc heuristics with no provable bounds or impractical, i.e., their running time is exponential in k [21,22], and their efficiency in practice is not clear. Indeed, we could not find implementations of such provable algorithms.

This motivates the questions that are answered affirmably in this paper: (i) Can we efficiently compute the corresponding ℓ_p k-rank approximation matrix A_k, similar to SVD? (ii) Can we remove the fine-tuning step by using the ℓ_p low rank approximation, while scarifying only a small decrease in the accuracy of the compressed network? (iii) Can we obtain smaller networks with higher accuracy (without fine-tuning) by minimizing the sum of non-squared errors, or any other power $p \neq 2$ of distances, instead of the ℓ_2 k-rank approximation via SVD?

1.3. Our Contribution

We answer these questions by suggesting the following contributions:

1. A new approach for compressing networks based on ℓ_p k-rank approximation instead of ℓ_2, for $p \in [1, \infty)$. The main motivation is the robustness to outliers and noise, which is supported by many theoretical justifications.

2. Provable algorithms for computing this ℓ_p low rank approximation of every $n \times d$ matrix A. The deterministic version takes time $O(nd^3 \log n)$, and the randomized version takes $O(nd \log n)$. The approximation factor depends polynomially on d, is independent of n for the deterministic version, and is only poly-logarithmic in n for the randomized version.
3. Experimental results confirming that our approach significantly improves existing results when the fine-tuning step is removed from the pipeline upon using SVD (see Section 5).
4. Full open source code is provided [23].

Our results are based on a novel combination of modern techniques in computational geometry and applied deep learning. We expect that future papers will extend this approach (see Section 7).

To obtain efficient implementations with provable guarantees, we suggest a leeway by allowing the approximation factor to be larger than k, instead of aiming for $(1+\varepsilon)$-approximation (PTAS). In practice, this worst-case bound seems to be too pessimistic, and the empirical approximation error in our experiments is much smaller. This phenomenon is common in approximation algorithms, especially in deep learning, when the dataset has a lot of structure and is very different from synthetic worse-case artificial examples. The main mathematical tool that we use is the Löwner ellipsoid, which generalizes the SVD case to general ℓ_p cases, inspired by many papers in the related work below.

To be part and not apart. Our technique can be combined with previous known works to obtain better compression. For example, DistilBERT [24] is based on knowledge distillation, and it reduces the size of the BERT [12] model by 40%, while maintaining 97% of its language understanding capabilities and being 60% faster. However, this result does not use low rank factorization to compress the embedding layer. We further compressed DistilBERT and achieved better accuracy than SVD.

2. Related Work

In the context of training giant models, some interesting approaches were suggested to reduce the memory requirement, e.g., [25,26]. However, those methods reduced the memory requirement at the cost of speed/performance. Later, [27] proposed a way to train large models based on parallelization. Here, the model size and evaluation speed are also still an obstacle. Hence, many papers were dedicated to the purpose of compressing neural networks in the field of NLP. These papers are based on different approaches such as pruning [28–32], quantization [33,34], knowledge distillation [24,35–41], weight sharing [42], and low rank factorization [42–44] (see the example table in [45] for compressing the BERT model). There is no convention for which approach from the above should be used. However, recent works, e.g., [42], showed that combining such approaches yields good results.

Subspace approximation. The ℓ_2 k-rank approximation can be solved easily in $\min\{nd^2, d^2n\}$ time, while a $(1+\varepsilon)$ approximation can be computed deterministically in $nd(k/\varepsilon)^{O(1)}$ time [46] for every $\varepsilon > 0$, and a randomized version takes $O(\text{nnz}(A)) + (n+d) \cdot (k/\varepsilon)^{O(1)}$ time, where $\text{nnz}(A)$ is the number of non-zero entries in A [47–49]. These and many of the following results are summarized in the seminal work of [21]. However, for $p \neq 2$, even computing a multiplicative $(1+\varepsilon)$-approximation is NP-hard when k is part of the input [21]. Nevertheless, it is an active research area, where techniques from computational geometry are frequently used. The case $p \geq 1$ was introduced in the theory community by [50], and earlier, the case $p = 1$ was introduced in the machine learning community by [51]. In [50], a randomized algorithm for any $p \geq 1$ that runs in time $nd2^{(k/\varepsilon)^{O(p)}}$ was suggested. The state of the art for $p \in [1,2)$ in [21] takes $O\left(\text{nnz}(A) + (n+d)(k/\varepsilon)^{O(1)} + 2^{((k/\varepsilon)^{O(1)})}\right)$ time.

Approximation algorithms for the ℓ_p low rank approximation were suggested in [52] for any $p \geq 1$, which we also handle. Although the obtained approximation, in some cases, is smaller than the approximation achieved in this paper, the running time in most cases (depending on k) is much larger than that of ours.

Regardless of the approximation, [52] suggests a polynomial time algorithm (one of many) as long as $k \in \Theta\left(\frac{\log n}{\log\log n}\right)$. Similar to the discussion in [52], our ℓ_1 low rank approximation allows us to recover an approximating matrix of any chosen rank, while the robust PCA [53] returns some matrix of unknown rank. Although variants of robust PCA have been proposed to force the output rank to be a given value [54,55], these variants make assumptions about the input matrix, whereas our results do not. The time complexity for $p = 1$ was improved in [56] to $nd(k/\varepsilon)^{O(1)} + (n+d)2^{(k/\varepsilon)^{O(1)}}$, and later, for general p to $nd(k/\varepsilon)^{O(1)} + 2^{(k/\varepsilon)^{O(p)}}$ [22]. The latter work, together with [57], also gives a *coreset* for subspace approximation, i.e., a way of reducing the number of rows of A so as to obtain a matrix A' such that the cost of fitting the rows of A' to any k-dimensional subspace F is within a $1+\varepsilon$ factor of the cost of fitting the rows of A to F; for $p = 2$, such coresets were known [47,58–60] and can be computed exactly $(\varepsilon = 0)$ [61,62].

Efficient approximations. The exponential dependency on k and hardness results may explain why we could not find (even inefficient) open or closed code implementations on the web. To our knowledge, it is an open problem to compute larger factor approximations $(\varepsilon \in O(1))$ in a time polynomial in k, even in theory. The goal of this paper is to provide such a provable approximation in time that is near-linear in n with practical implementation and to demonstrate our usefulness in compressed networks.

3. Method

Notations. For a pair of integers $n, d \geq 1$, we denote by $\mathbb{R}^{n\times d}$ the set of all $n \times d$ real matrices, by $I_d \in \{0,1\}^{d\times d}$ the identity matrix, and $[n] = \{1, \cdots, n\}$. For a vector $x \in \mathbb{R}^d$, a matrix $A \in \mathbb{R}^{n\times d}$, and a real number $p > 0$, the pth norm of x is defined as $\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p}$, and the ℓ_p entry-wise norm of A is defined as $\|A\|_{p,p} = \left(\sum_{i=1}^{d} \|Ae_i\|_p^p\right)^{1/p}$, where $e_i \in \{0,1\}^d$ is a vector whose ith entry is 1 and 0 elsewhere. We say that the columns of a matrix $A \in \mathbb{R}^{n\times d}$ (where $n \geq d$) are orthogonal if $A^T A = I_d$. In addition, a matrix $F \in \mathbb{R}^{d\times d}$ is called positive definite matrix if F is a symmetric matrix, and for every $x \in \mathbb{R}^d$ such that $\|x\|_2 > 0$, we have $x^T F x > 0$. Furthermore, we say that a set $L \subseteq \mathbb{R}^d$ is centrally symmetric if for every $x \in L$, it holds that $-x \in L$. Finally, a set $L \subseteq \mathbb{R}^d$ is called a convex set if for every $x, y \in L$ and $\theta \in [0,1]$, $\theta x + (1-\theta)y \in L$.

3.1. $\|\cdot\|_p$-SVD Factorization and the Löwner Ellipsoid

In what follows, we intuitively and formally describe the tools that will be used in our approach. Definition 1 is based on Definition 4 in [63]. While the latter defines a generic factorization for a wide family of functions, Definition 1 focuses on our case, i.e., the function we wish to factorize is $\|Ax\|_p$ for any $p \geq 1$, where $A \in \mathbb{R}^{n\times d}$ is the input matrix, and x is any vector in $\mathbb{R}^d$.

Definition 1 (Variant of Definition 4 [63])**.** *Let $A \in \mathbb{R}^{n\times d}$ be a matrix of rank d, and let $p \geq 1$ be a real number. Suppose that there is a diagonal matrix $D \in (0,\infty)^{d\times d}$ of rank d, and an orthogonal matrix $V \in \mathbb{R}^{d\times d}$, such that for every $x \in \mathbb{R}^d$,*

$$\left\|DV^T x\right\|_2^p \leq \|Ax\|_p^p \leq d^{\frac{p}{2}} \left\|DV^T x\right\|_2^p.$$

Define $U = A\left(DV^T\right)^{-1}$. Then, $UDV^T = A$ is called the $\|\cdot\|_p$-SVD of A.

Why $\|\cdot\|_p$-SVD? The idea behind using the $\|\cdot\|_p$-SVD factorization of an input matrix A is that we obtain a way to approximate the span of the column space of $A \in \mathbb{R}^{n\times d}$. This allows us to approximate the dot product Ax for any $x \in \mathbb{R}^d$, which implies an approximation for the optimal solution of the ℓ_p low rank approximation problem.

For example, in the case of $p = 2$, the $\|\cdot\|_2$-SVD of a matrix $A \in \mathbb{R}^{n\times d}$ is equivalent to the known SVD factorization $A = UDV^T$. This holds due to the fact that the columns of the matrix U are orthogonal, and for every $x \in \mathbb{R}^d$, we have $\|Ax\|_2^2 = \|UDV^Tx\|_2^2 = \|DV^Tx\|_2^2$. As for the general case of any $p \geq 1$, [63] showed that the $\|\cdot\|_p$-SVD factorization always exists, and can be obtained using the *Löwner* ellipsoid.

Theorem 2 (Variant of Theorem III [64])**.** *Let $D \in [0,\infty)^{d\times d}$ be a diagonal matrix of full rank and an orthogonal matrix $V \in \mathbb{R}^{d\times d}$, and let E be an ellipsoid defined as $E = \left\{x \in \mathbb{R}^d \middle| x^TVD^TDV^Tx \leq 1\right\}$.*

Let L be a centrally symmetric compact convex set. Then, there exists a unique ellipsoid E called the Löwner ellipsoid *of L such that $1/\sqrt{d}E \subseteq L \subseteq E$, where $1/\sqrt{d}E = \left\{1/\sqrt{d}x \middle| x \in E\right\}$.*

Computing $\|\cdot\|_p$-SVD via Löwner ellipsoid. Intuitively speaking, for an input matrix $A \in \mathbb{R}^{n\times d}$, the $\|\cdot\|_p$-SVD $A = UDV^T$ aims to bound from above and below the cost function $\|Ax\|_p^p$ for any $x \in \mathbb{R}^d$ by the term $\|DV^Tx\|_2^p$. Since $\|Ax\|_p^p$ is a convex continuous function (for every $x \in \mathbb{R}^d$), the level set $L = \left\{x \in \mathbb{R}^d \middle| \|Ax\|_p \leq 1\right\}$ is also convex. Having a convex set enables us to use the *Löwner ellipsoid*, which, in short, is the minimum volume enclosing ellipsoid of L. In addition, contracting the *Löwner ellipsoid* by $\sqrt{d}$ yields an inscribed ellipsoid in L. It turns out that D, V of the $\|\cdot\|_p$-SVD represents the Löwner ellipsoid of L as follows: D is a diagonal matrix such that its diagonal entries contain the reciprocal values of the ellipsoid axis lengths, and V is an orthogonal matrix which is the basis of the same ellipsoid. Using the enclosing and inscribed ellipsoids (the Löwner ellipsoid and its contracted form) enables us to bound $\|\cdot\|_p$ using the mahalonobis distance. Although in traditional k ℓ_2-low rank factorization with respect to an input matrix $A \in \mathbb{R}^{n\times d}$, the optimal result is equal to the sum of the smallest $d-k$ singular values, we generalize this concept to ℓ_p-low rank factorization. Specifically, the singular values of D (the reciprocal values of the ellipsoid axis lengths) serve as a bound on the "ℓ_p singular values of A".

3.2. Additive Approximation for the ℓ_p-Low Rank Factorization

In what follows, we show how to compute an approximated solution for the ℓ_p-low rank factorization for any $p \geq 1$ (see Algorithm 1). This is based on the $\|\cdot\|_p$-SVD factorization (see Definition 1).

From $\|\cdot\|_p$-SVD to ℓ_p-low rank factorization. For any $k \in [d-1]$ and any matrix $A \in \mathbb{R}^{n\times d}$ of rank d, the ℓ_p-low rank factorization problem aims to minimize $\|A - AXX^T\|_{p,p}^p$ over every matrix $X \in \mathbb{R}^{d\times k}$ whose columns are orthogonal. As a byproduct of the orthogonality of X, the problem above is equivalent to minimizing $\|AYY^T\|_{p,p}^p$ over every matrix $Y \in \mathbb{R}^{d\times(d-k)}$ whose columns are orthogonal such that $YY^T = I_d - XX^T$. By exploiting the definition of the entry-wise ℓ_p norm of AYY^T, we can use $\|\cdot\|_p$-SVD to bound this term from above and below using the mahalonobis distance. Furthermore, we will show that by using the $\|\cdot\|_p$-SVD, we can compute a matrix A_k of rank k such that $\|A - A_k\|_{p,p}^p$ depends on the ellipsoid axis lengths (see Algorithm 1 and Theorem 5).

Overview of Algorithm 1. Algorithm 1 receives as input a matrix $A \in \mathbb{R}^{n\times d}$ of rank d, a positive integer $k \in [d-1]$, and a positive number $p \geq 1$ and outputs a matrix A_k of rank k, which satisfies Theorem 5. At Line 1, we compute a pair of matrices $D, V \in \mathbb{R}^{d\times d}$ such that the ellipsoid $E := \left\{x \in \mathbb{R}^d \middle| x^TVD^TDV^Tx \leq 1\right\}$ is the Löwner ellipsoid of $L := \left\{x \in \mathbb{R}^d \middle| \|Ax\|_p \leq 1\right\}$, where D is a diagonal matrix of rank d, and V is an orthogonal matrix; we refer the reader to the Appendix A for computing the Löwner ellipsoid. At Line 2, we compute the matrix U from the $\|\cdot\|_p$-SVD of A (see Definition 1). At Lines 3–4, we set D_k to be the diagonal matrix of $d \times d$ entries where the first k diagonal entries are identical to the first k diagonal entries of D, while the rest of the matrix is set to 0 (see Figure 2 for an illustrative description of our algorithm).

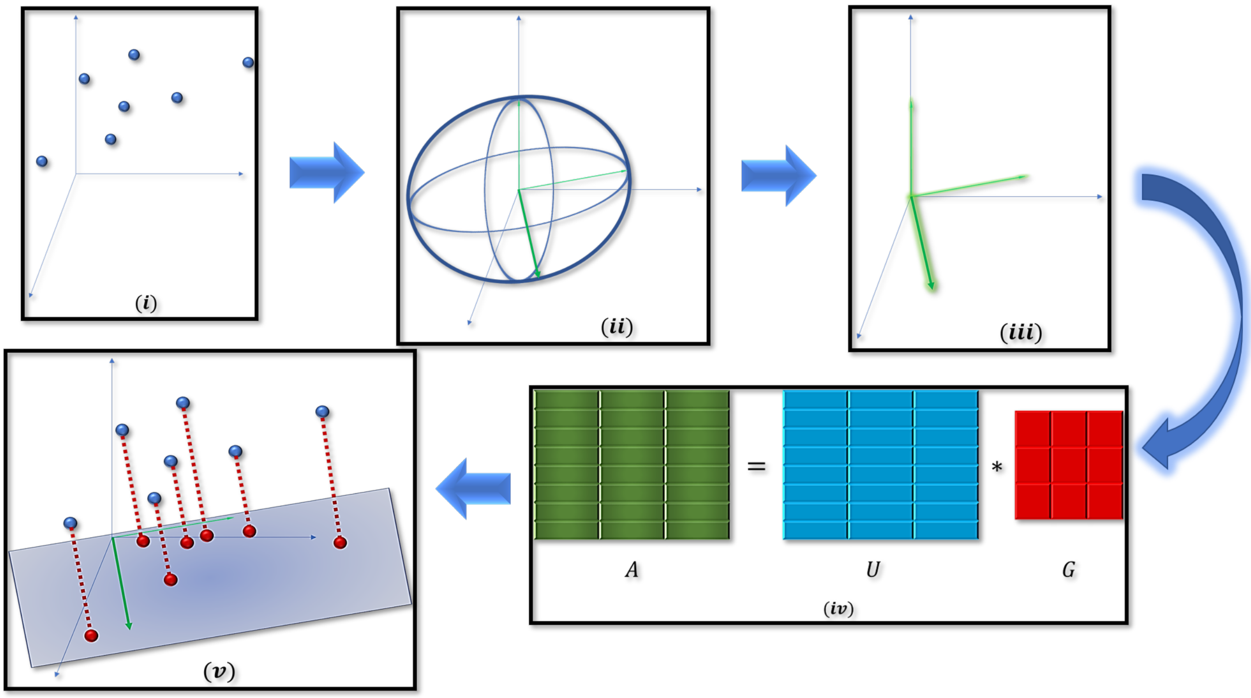


Figure 2. Illustration of our method. Given a matrix $A \in \mathbb{R}^{n \times d}$ whose rows are points in $\mathbb{R}^d$ (step (***i***)), we first compute the Löwner ellipsoid of $f(x) = \|Ax\|_p$ for every $x \in \mathbb{R}^d$ (step (***ii***)). This enables us to encapsulate the geometrical properties of f. After computing the minimum volume enclosing ellipsoid, we focus on the ellipsoids' axes which will form our matrix G (step (***iii***)). Due to the invertability of G, we can factorize A into a multiplication of two matrices, $U = AG^{-1}$ and G (step (***iv***)). Finally, we choose the longest k axes of the ellipsoid where these vectors will form a subspace on which the points will be projected on to form our low rank approximation as illustrated above (step (***v***), see red points).

```
Algorithm 1: $\ell_p$-LOW-RANK$(A, k, p)$.
    Input: A matrix $A \in \mathbb{R}^{n \times d}$ of rank $d$, $p \geq 1$, a positive integer $k \in [d-1]$, and a
           positive real number $p \geq 1$.
    Output: A matrix $U \in \mathbb{R}^{n \times d}$, a diagonal matrix $D_k \in [0, \infty)^{d \times d}$, an orthogonal
            matrix $V \in \mathbb{R}^{d \times d}$ where $U, V$ are from the $\|\cdot\|_p$-SVD of $A$, and a set of $d$
            positive real numbers $\{\sigma_1, \ldots, \sigma_d\}$.
1   $(D, V) :=$ LÖWNER$(A, p)$ // See Algorithm A1 at the Appendix A
2   $U := A\left(DV^T\right)^{-1}$ // computing $U$ from the $\|\cdot\|_p$-SVD of $A$ with respect
        to the $\ell_p$-regression problem
3   $\{\sigma_1, \ldots, \sigma_d\} :=$ the diagonal entries of $D$
4   $D_k := \mathrm{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0)$ // A diagonal matrix in $\mathbb{R}^{d \times d}$
5   return $U, D_k, V, \{\sigma_1, \ldots, \sigma_d\}$
```

4. Analysis

Some of the proofs in this section were moved into the Supplementary Material due to space limitations.

4.1. Deterministic Result

In what follows, we present our deterministic solution for the ℓ_p-low rank factorization problem.

Claim 3. *Let $D \in [0, \infty)^{d\times d}$ be a diagonal matrix of rank d, and let $\sigma > 0$ be the lowest singular value of D. Then, for every unit vector $x \in \mathbb{R}^d$, $\|Dx\|_2 \geq \sigma$.*

Proof. Let $x \in \mathbb{R}^d$ be a unit vector, and for every $i \in [d]$, let $D_{i,i}$ denote the ith diagonal entry of D, and x_i denotes the ith entry of x. Observe that

$$\|Dx\|_2 = \left(\sum_{i=1}^{d} |D_{i,i}x_i|^2\right)^{\frac{1}{2}} \geq \left(\sum_{i=1}^{d} |\sigma x_i|^2\right)^{\frac{1}{2}} = \sigma\|x\|_2 = \sigma,$$

where the first equality follows from the definition of norm, the inequality holds by definition of σ, and the last equality holds since x is a unit vector. □

Lemma 4 (Special case of Lemma 15 [63])**.** *Let $A \in \mathbb{R}^{n\times d}$ be a matrix of full rank, $p \geq 1$. Then, there exist a diagonal matrix $D \in [0, \infty)^{d\times d}$ of full rank and an orthogonal matrix $V \in \mathbb{R}^{d\times d}$ such that for every $x \in \mathbb{R}^d$,*

$$\left\|DV^T x\right\|_2^p \leq \|Ax\|_p^p \leq d^{\frac{p}{2}} \left\|DV^T x\right\|_2^p \tag{1}$$

Proof. First, let $L = \left\{ \tilde{x} \in \mathbb{R}^d \middle| \|A\tilde{x}\|_p^p \leq 1 \right\}$, and put $x \in \mathbb{R}^d$. Observe that (i) since $p \geq 1$, the term $\|A\tilde{x}\|_p$ is a convex function for every $\tilde{x} \in \mathbb{R}^d$ which follows from properties of norm function. This means that the level set L is a convex set. In addition, (ii) by definition of L, it holds that for every $\tilde{x} \in L$, also $-\tilde{x} \in L$, which makes L a centrally symmetric set by definition. Note that (iii) since A is of full rank, then L spans $\mathbb{R}^d$.

Since properties (i)–(iii) hold, we obtain by Theorem 2 that there exists a diagonal matrix $D \in [0, \infty)^{d\times d}$ of full rank and an orthogonal matrix $V \in \mathbb{R}^d$ such that the set $E = \left\{ \tilde{x} \in \mathbb{R}^d \middle| \tilde{x}^T V D^T D V^T \tilde{x} \leq 1 \right\}$ satisfies

$$\frac{1}{\sqrt{d}} E \subseteq L \subseteq E. \tag{2}$$

Proving the right hand side of Equation (1). Let $y = \frac{1}{\|DV^T x\|_2} x$, and observe that

$$\|Ax\|_p^p = \left(\sqrt{d}\left\|DV^T x\right\|_2\right)^p \left\|\frac{1}{\sqrt{d}} Ay\right\|_p^p \leq d^{\frac{p}{2}} \left\|DV^T x\right\|_2^p, \tag{3}$$

where the equality follows from the definition of y, and the inequality holds since $\frac{1}{\sqrt{d}} y \in L$ follows from Equation (2).

Proving the left hand side of Equation (1). Since L spans $\mathbb{R}^d$, there then exists $b > 0$ such that $\|A(bx)\|_p^p = 1$. By Equation (2), $bx \in E$, which results in $\|DV^T x\|_2 = \frac{1}{b}\|DV^T(bx)\|_2 \leq \frac{1}{b}$. Thus,

$$\|Ax\|_p^p = \frac{1}{b^p} \|A(bx)\|_p^p = \frac{1}{b^p} \geq \left\|DV^T x\right\|_2^p \tag{4}$$

Since Equations (3) and (4) hold for every $x \in \mathbb{R}^d$, Lemma 4 follows. □

Theorem 5. *Let $A \in \mathbb{R}^{n\times d}$ be real matrix, $p \geq 1$; $k \in [d-1]$ be an integer; and $(U, D_k, V, \{\sigma_1, \ldots \sigma_d\})$ be the output of a call to ℓ_p-LOW-RANK(A, k, p). Let $A_k = UD_k V^T$. Then,*

$$d\sigma_d^p \leq \|A - A_k\|_{p,p}^p \leq d^{1+\frac{p}{2}} \sigma_k^p.$$

Proof. First, we assume that $p \neq 2$; otherwise, the $\|\cdot\|_2$ factorization is the *SVD* factorization, and we obtain the optimal solution for the ℓ_2 low rank approximation problem. For every $i \in [d]$, let $e_i \in \mathbb{R}^d$ be a vector of zeros, except for its ith entry, where it is set to 1.

Observe that $\|A - A_k\|_{p,p}^p = \sum_{i=1}^{d} \|(A - A_k)e_i\|_p^p = \sum_{i=1}^{d} \left\| A\left(I - \left(DV^T\right)^{-1} D_k V^T\right) e_i \right\|_p^p$, where the first equality holds by definition of $\|\cdot\|_{p,p}^p$, and the second equality follows from the definition of A_k (see Lines 3–4 of Algorithm 1).

Plugging $A := A, D := D, V := V, x := \left(I - \left(DV^T\right)^{-1} D_k V^T\right) e_i$ into Lemma 4 yields that for every $i \in [d]$,

$$\begin{aligned} \left\|(D - D_k)V^T e_i\right\|_2^p &\le \|(A - A_k)e_i\|_{p,p}^p \\ &\le d^{\frac{p}{2}} \left\|(D - D_k)V^T e_i\right\|_2^p. \end{aligned} \tag{5}$$

Observe that for every $i \in [d]$,

$$\left\|(D - D_k)V^T e_i\right\|_2^p \le \|D - D_k\|_2^p \left\|V^T e_i\right\|_2^p \le \|D - D_k\|_2^p, \tag{6}$$

where the first inequality holds by properties of the ℓ_2 matrix induced norm, and the second inequality holds since V is an orthogonal matrix.

Since $V^T e_i$ is a unit vector,

$$\left\|(D - D_k)V^T e_i\right\|_2^p \ge \sigma_d^p, \tag{7}$$

where the inequality holds by plugging $x := V^T e_i$ and $D := (D - D_k)$ into Claim 3.

In addition, we have that

$$\sigma_d \le \|D - D_k\|_2 = \sigma_{k+1} \tag{8}$$

where both the inequality and equality hold since σ_d is the lowest eigenvalue of D, D being a diagonal matrix.

By combining Equations (5)–(8), we obtain that for every $i \in [d]$,

$$\sigma_d^p \le \|(A - A_k)e_i\|_p^p \le d^{\frac{p}{2}} \sigma_{k+1}^p. \tag{9}$$

Theorem 5 follows by summing Equation (9) over every $i \in [d]$. □

Note that the set $\{\sigma_i\}_{i=1}^d$ denotes the reciprocal values of the ellipsoid E axis's lengths, where E is the Löwner ellipsoid of $L = \left\{ x \in \mathbb{R}^d \middle| \|Ax\|_p \le 1 \right\}$. As discussed in the previous section, these values serve to bound the "ℓ_p singular values of A".

4.2. Randomized Result

In addition to our deterministic result, we also show how to support a randomized version that computes an approximation in a faster time, which relies on the following result of [65].

Theorem 6 (Variant of Theorem 10 [65]). *For any $A \in \mathbb{R}^{n \times d}$ of rank d and $p \ge 1$, one can compute an invertible matrix $R \in \mathbb{R}^{d \times d}$ and a matrix $U = AR^{-1}$ such that $\|Rx\|_2 \le \|Ax\|_p \le d\left(d^3 + d^2 \log n\right)^{|1/p - 1/2|} \|Rx\|_2$ holds with a probability of at least $1 - \frac{1}{n}$, where R can be computed in time $O(nd \log n)$.*

Theorem 7. *Let $A \in \mathbb{R}^{n \times d}$ be real matrix, $p \ge 1$, and $k \in [d-1]$ be an integer. There exists a randomized algorithm which, when given a matrix $A \in \mathbb{R}^{n \times d}$, $k \in [d-1]$, in time $O(nd \log n)$, returns $(U, D_k, V, \{\sigma_1, \ldots \sigma_d\})$, such that*

$$d\sigma_d^p \le \|A - A_k\|_{p,p}^p \le d^{1+p} \left(d^3 + d^2 \log n\right)^{|1 - p/2|} \sigma_{k+1}^p,$$

holds with a probability of at least $1 - \frac{1}{n}$*, where* $A_k = UD_kV^T$*.*

Proof. The algorithm is described throughout the following proof. Let $R \in \mathbb{R}^{d\times d}$ be as defined in Theorem 6 when plugging $A := A$ into Theorem 6. Let $R = \tilde{U}DV^T$ be the *SVD* of R; $D_k \in [0,\infty)^{d\times d}$ be a diagonal matrix where its first k diagonal entries are identical to those of D, while the rest of the entries in D_k are set to 0; and $\{\sigma_1, \cdots, \sigma_d\}$ be the set of singular values of D. Note that since for every $x \in \mathbb{R}^d$, by Theorem 6 it holds that

$$\|Rx\|_2^p \le \|Ax\|_p^p d^p \left(d^3 + d^2 \log n\right)^{|1-p/2|} \|Rx\|_2^p.$$

From here, similar to the proof of Theorem 5, we obtain that

$$d\sigma_d^p \le \|A - A_k\|_{p,p}^p \le d^{1+p}\left(d^3 + d^2 \log n\right)^{|1-p/2|} \sigma_{k+1}^p.$$

□

Remark 8. *Note that in our context of embedding layer compression, the corresponding embedding matrix* A *has more columns than rows. Regardless, our* ℓ_p *norm of any* $A - B$ *such that* $A, B \in d \times n$ *enables us to have* $\|A - B\|_{p,p}^p = \|A^T - B^T\|_{p,p}^p$*. Hence, substituting* $A := A^T$ *and* $A_k := A_k^T$ *yields*

$$d\sigma_d^p \le \|A - A_k\|_{p,p}^p \le d^{1+\frac{p}{2}} \sigma_{k+1}^p,$$

for our deterministic results, and similarly, we can obtain this for our randomized result.

5. Experimental Results

The compressed networks. We compress several frequently used NLP networks:

(i) BERT [12]: BERT is a bidirectional transformer pre-trained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia.

(ii) DistilBERT [24]: the DistilBERT model is smaller, faster, cheaper, and lighter than BERT. This model is a distilled version of BERT. It has 40% less parameters than bert-base-uncased and runs 60% faster, while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark [24].

(iii) XLNet [66]: XLnet is an extension of the Transformer-XL model [67] pre-trained using an autoregressive method to learn bidirectional contexts by maximizing the expected likelihood over all permutations of the input sequence factorization order.

(iv) RoBERTa [68]: RoBERTa modifies the key hyperparameters in BERT, including removing BERT's next-sentence pre-training objective, and training with much larger mini-batches and learning rates. This allows RoBERTa to improve on the masked language modeling objective compared with BERT and leads to better downstream task performance.

See full details on the sizes of each network and their embedding layer before compression in Table 1.

Implementation, Software, and Hardware. All the experiments were conducted on an AWS p2.xlargs machine with 1 GPU NVIDIA K80, 4 vCPUs, and 61 RAM [GiB]. We implemented our suggested compression algorithm (Algorithm 1) in Python 3.8 using the Numpy library [69]. To build and train networks (i)–(iv), we used the suggested implementation in the Transformers https://github.com/huggingface/transformers (accessed on 15 July 2021) library from HuggingFace [70] (Transformers version 2.3 and PyTorch version 1.5.1 [71]). Before the compression, all the networks were fine-tuned on all the tasks from the GLUE benchmark to obtain almost the same accuracy results as reported in the original papers. Since we did not succeed in obtaining close accuracy on the tasks QQP and WNLI (with most of the network), we did not include results from them.

Our compression. We compress each embedding layer (matrix) of the reported networks by factorizing it into two smaller layers (matrices) as follows. For an embedding layer that is defined by a matrix $A \in \mathbb{R}^{n \times d}$, we compute the matrices U, D_k, V by a call to ℓ_p-LOW-RANK$(A, k, 1)$ (see Algorithm 1), where k is the low rank projection we wish to have. Observe, that the matrix D_k is a diagonal matrix, and its last $d - k$ columns are zero columns. We then compute a non-square diagonal matrix $D'_k \in \mathbb{R}^{d \times k}$ that is the result of removing all the zero columns of D_k. Now, the ℓ_1 k-rank approximation of A can be factorized as $A_k = \left(U\sqrt{D'_k}\right)\left(\sqrt{D'_k}^T V^T\right)$. Hence, we save the two matrices (layers): (i) $U\sqrt{D'_k}$ of size $n \times k$, and (ii) $\sqrt{D'_k}^T V^T$ of size $k \times d$. This yields two layers of a total size of $nk + kd$ instead of a single embedding layer of a total size of nd.

Reported results. We report the test accuracy drop (relative error) on all the tasks from the GLUE benchmark [72] after compression for several compression rates:

1. In Figure 3, the x-axis is the compression rate of the embedding layer, and the y-axis is the accuracy drop (relative error) with respect to the original accuracy of the network. Each figure reports the results for a specific task from the GLUE benchmark on all the networks we compress. Here, all reported results are compared to the known ℓ_2-factorization using SVD. In addition, in all the experiments, we do not fine-tune the model after compressing; this is to show the robustness and efficiency of our technique.
2. Table 2 suggests the best compressed networks in terms of accuracy vs size. For every network from (i)–(iv), we suggest a compressed version of it with a very small drop in the accuracy and sometimes with an improved accuracy. Given a network "X", we call our compressed version of "X" "RE-X", e.g., RE-BERT and RE-XLNet. The "RE" here stands for "Robust Embedding".
3. Table 3 reports a comparison between our approach and different compressionmethods that do not require fine-tuning or any usage of the training data after compression:
   (i) SVD.
   (ii) L1PCA [73].
   (iii) Pruning [74].
   (iv) Random pruning.
   (v) Syn flow [75].

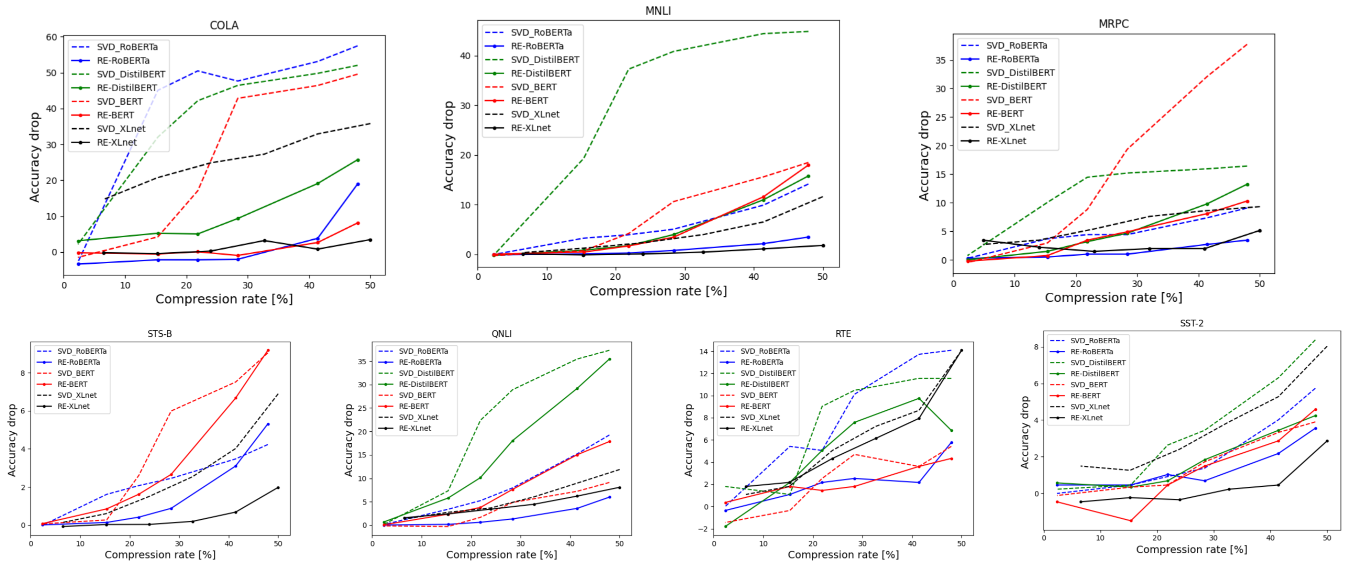


Figure 3. Here, we report the accuracy drop (additive error) as a function of the embedding layer's compression rate on the networks (i)–(iv). We compare our results with SVD over several tasks from the GLUE benchmark. For a network "X", our compressed version of it is called "RE-X", e.g., RE-BERT and RE-XLNet.

Table 1. The sizes of the networks BERT, RoBERTA, XLNet, and DistilBERT, and their embedding layers.

Model		Embedding Layer Size	Parameters
BERT	base	30,522 × 768	110 M
RoBERTa	base	50,265 × 768	125 M
XLNet	base	32,000 × 768	110 M
DistilBERT	base	30,522 × 768	66 M

Table 2. In the above, we present our compressed networks and their drop in accuracy based on the compression rate of the embedding layer. Specifically, each non-percentile value represents the accuracy drop achieved by our compressed model with respect to its original model (e.g., RE-RoBERTa is a compressed model of RoBERTa), while negative values present improvements in the accuracy upon the non-compressed version of the corresponding model. The last column is the average accuracy drop over all tested tasks. The "RE" here stands for "Robust Embedding".

Model		Embedding Layer Compression Rate	MRPC	COLA	MNLI	SST-2	STS-B	QNLI	RTE	Avg.
RE-RoBERTa	base	15%	0.49	−2.16	0.01	0.045	0.013	0.018	1.08	−0.072
	small	28%	0.98	−2.01	0.08	0.68	0.87	1.33	2.52	0.63
	tiny	41%	2.69	3.82	2.18	2.17	3.10	3.58	2.16	2.81
RE-XLNet	base	15%	2.20	−0.43	−0.07	0.22	0.03	2.39	2.16	0.92
	small	21%	1.47	0.26	0.11	−0.34	0.03	3.42	4.33	1.32
	tiny	28%	1.96	3.19	0.47	−0.22	0.19	4.46	6.13	2.31
RE-BERT	base	15%	0.73	−0.54	0.48	−1.49	0.85	2.36	1.80	0.59
	small	21%	3.43	0.08	1.72	0.45	1.62	3.78	1.44	1.78
	tiny	28%	4.90	−0.94	3.48	1.49	2.66	7.65	1.80	3
RE-DistilBERT	base	15%	1.47	5.24	0.86	0.34	0.13	5.80	2.16	2.28

Table 3. We evaluate our compression method against different compression techniques that do not use any fine-tuning steps on the RoBERTa model (or any usage of the training data after compression). The following table reports the drop in accuracy on the MRPC task after using these compression techniques.

Compression Method		Embedding Layer Compression Rate	MRPC
SVD	base	15%	5.70
	small	28%	6.25
	tiny	41%	9.51
L1PCA [73]	base	15%	3.04
	small	28%	4.94
	tiny	41%	18.48
Pruning [74]	base	15%	1.90
	small	28%	2.17
	tiny	41%	2.98
Random pruning	base	15%	1.36
	small	28%	3.06
	tiny	41%	4.38
SynFlow [75]	base	15%	0.81
	small	28%	1.60
	tiny	41%	2.75
Our compression	base	15%	0.49
	small	28%	0.98
	tiny	41%	2.69

6. Discussion

It can be seen by Figure 3 that our approach is more robust than the traditional SVD. In most of the experiments, our suggested compression achieves better accuracy for the same compression rate compared to the traditional SVD. Mainly, we observed that our compression schemes shine when either vocabulary is rich (the number of subword units is large) or the model itself is small (excluding the embedding layer). Specifically speaking, in RoBERTa, our method achieves better results due to the fact that RoBERTa's vocabulary is rich (i.e., 50 K subword units compared to the 30 K in BERT). This large vocabulary increases the probability of having outliers in it, which is the main justification for our approach. In DistilBERT, the network is highly efficient. This can lead to a "sensitive snowball effect", i.e., the classification is highly affected by even the smallest errors caused by the compression of the embedding layer. Since SVD is sensitive to outliers and due to the fact that the network is highly sensitive to small errors, the existence of outliers highly affects the results. This phenomenon is illustrated throughout Figure 3. Here, our compression scheme outperforms the SVD due to its robustness against outliers, which, in turn, achieves smaller errors. As for XLNet, the model encodes the relative positional embedding, which, in short, represents an embedding of the relative positional distance between words. In our context, this means that having outliers highly affects the relative positional embedding, which, in turn, affects the classification accuracy. Hence, this explains why we outperform SVD. Since none of the above phenomena hold for BERT, this may explain why SVD sometimes achieves better results. However, across most tasks, our compression scheme is favorable upon SVD.

Finally, for some tasks at low compression rates, the accuracy has been improved (e.g., see task SST-2 at Figure 3 when compressing BERT). This may be due to the fact that at low compression rates, we remove the least necessary (redundant) dimensions. Thus, if these dimensions are actually unnecessary, by removing them, we obtain a generalized model which is capable of classifying better.

7. Conclusions and Future Work

We provided an algorithm that computes an approximation for ℓ_p k-rank approximation, where $p \geq 1$. We then suggested a new approach for compressing networks based on k-rank ℓ_p-approximation, where $p \in [1, 2]$ instead of ℓ_2. The experimental results in Section 5 showed that our suggested algorithm overcomes the traditional ℓ_2 k-rank approximation and achieves higher accuracy for the same compression rate when there is no fine-tuning involved.

Future work includes: (1) Extending our approach to other factorization models, such as non-negative matrix approximation or dictionary learning; (2) experimental results on other benchmarks and other models; (3) suggesting algorithms for the ℓ_p k-rank approximation for any $p \in (0, 1)$, while checking the practical contribution in compressing deep networks for this case; and (4) combining this result with other compression techniques to obtain a smaller network with higher accuracy.

Author Contributions: Conceptualization, M.T., A.M., M.W., and D.F.; methodology, M.T. and A.M.; software, M.T. and M.W.; validation, M.T. and M.W.; formal analysis, M.T. and D.F.; investigation, M.T. and A.M.; resources, M.W. and D.F.; data curation, M.T. and M.W.; writing—original draft preparation, M.T. and A.M.; writing—review and editing, M.T., A.M., M.W., and D.F.; visualization, M.T.; supervision, D.F.; project administration, M.T.; funding acquisition, M.W. and D.F. All authors have read and agreed to the published version of the manuscript.

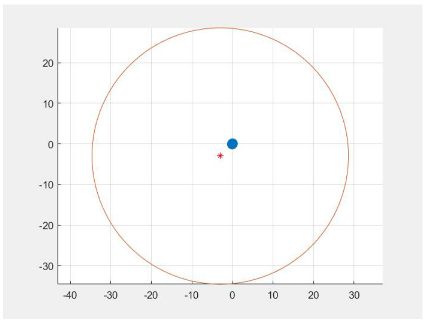
Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

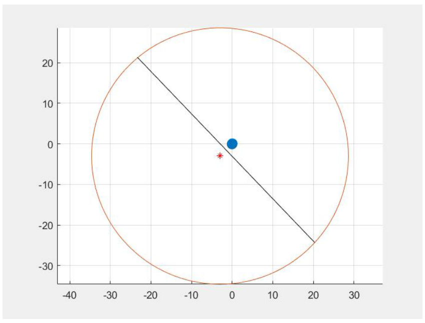
Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

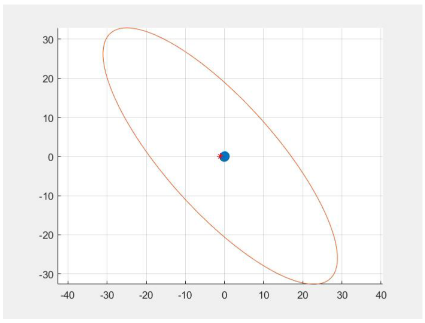
Appendix A. Computing the Löwner Ellipsoid



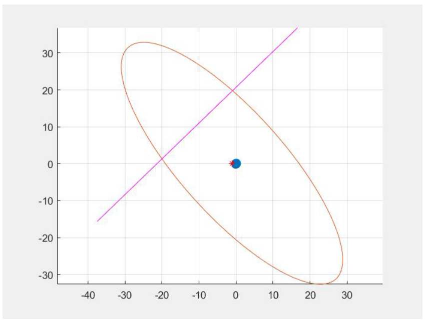
(**a**)



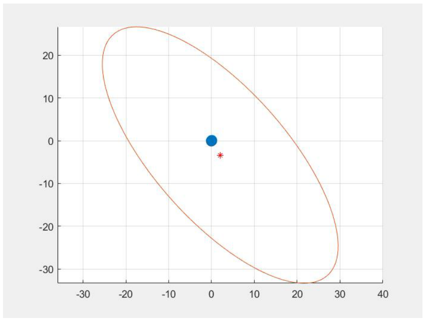
(**b**)



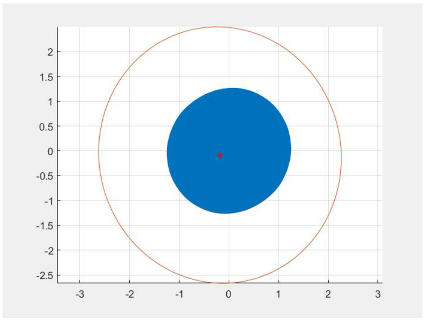
(**c**)



(**d**)



(**e**)



(**f**)

Figure A1. *Computing the Löwener ellipsoid. Step I*: We start with an ellipsoid that contains our level set (the blue body). From here, the basic ellipsoid method is invoked, i.e., while the center is not contained inside the level set (blue body), a separating hyperplane between the center of the ellipsoid and the level set is computed, and the ellipsoid is stretched in a way such that the center moves closer in distance to the level set. The basic ellipsoid method halts when the center is contained in the level set (see (**a**–**c**) for illustration of the ellipsoid method). *Step III:* We compute a contracted version of the current ellipsoid and check if all of its vertices are contained in the level set. If there exists one ellipsoid's vertex which is not contained in the level set, we find the farthest vertex of the contracted ellipsoid from the level set and compute a separating hyperplane between it and the level set. Then, the ellipsoid is stretched such that this vertex becomes closer to the level set presented in (**d**,**e**). We then loop StepsII–III until the contracted ellipsoid's vertices are contained in the level set (see (**f**)).

For an input matrix $A \in \mathbb{R}^{n \times d}$ of rank d and a number $p \geq 1$, we now show how to compute the *Löwner ellipsoid* for the set $L := \left\{ x \middle| x \in \mathbb{R}^d, \|Ax\|_p \leq 1 \right\}$. This is a crucial step towards computing the $\|\cdot\|_p$-SVD (see Definition 1) for the matrix A in the context of the ℓ_p-low rank approximation problem, which will allow us to suggest an approximated solution (See Theorem 5).

Overview of Algorithm A1 (computing the Löwner ellipsoid). Algorithm A1 receives as input a matrix $A \in \mathbb{R}^{n \times d}$ of rank d and a number $p \geq 1$. It outputs a Löwner ellipsoid for the set L (see Line 1 of Algorithm A1). First, at Line 1, we initialize L to be set of all the points x in $\mathbb{R}^d$ such that $\|Ax\|_p \leq 1$. At Lines 2–5, we find a ball E in $\mathbb{R}^d$ of radius r, which contains the set L, and its center is set to be the origin 0_d. Then, we build a diagonal matrix F, where we set its diagonal entries to r.

Lines 8–12 represent the pseudo-code of the basic ellipsoid method which is described in detail in [76], where we set H to the separating hyperplane between c (the center of the ellipsoid E) and L; b is set to be the multiplication between F and the normalized subgradient of $\|Ax\|_p$ at $x = c$, where b is used to set the next candidate ellipsoid.

In Lines 13–17, we compute the next candidate ellipsoid E, and based on it, we set $\tilde{V}$ to be the set containing the vertices of the inscribed ellipsoid $\frac{1}{d}(E - c) + c$ in L. Now, if $\tilde{V} \subseteq L$, then we halt the algorithm; otherwise, we find the farthest vertex point v in $\tilde{V}$ from L with respect to $\|Ax\|_p$, and finally, we set H to be the separating hyperplane between v and L.

Lines 19–25 present the pseudo code of applying a shallow cut update to the ellipsoid E; this is described in detail in [76]. Finally, at Line 27, we set G to be the Cholesky decomposition of F^{-1} (see [77] for more details). For formal details, see Theorem 2.

Algorithm A1: LÖWNER(A, p).

```
Input: A matrix $A \in \mathbb{R}^{n\times d}$ of rank $d$, and $p \geq 1$
Output: A diagonal matrix $D \in [0,\infty)^{d\times d}$ and an orthogonal matrix $V \in \mathbb{R}^{d\times d}$
        satisfying Theorem 2.
1  $L := \left\{x \middle| x \in \mathbb{R}^d, \|Ax\|_1 \leq 1\right\}$
2  $E :=$ A ball centered around the origin which contains $L$
3  $r :=$ the radius of $E$
4  $F := rI_d$
5  $c := \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d$
6  while true do
7      while $c \notin L$ do
               /* Here we apply the basic ellipsoid method                    */
8          $grad := \|Ac\|_p^{p-1} \cdot \|Ac\|_{p-1}^{p-1} \cdot \left[\sum_{i=1}^{n} \frac{A_{i,1}c_1}{|A_{i*}^T c|}, \cdots, \sum_{i=1}^{n} \frac{A_{i,d}c[d]}{|A_{i*}^T c|}\right]^T$ // The
               gradient of $\ell_p$-regression at $c$
9          $H := \frac{1}{\|grad\|_\infty} grad$ // Separating hyperplane between the center of
               the ellipsoid and $L$
10         $b := \frac{1}{\sqrt{H^T F H}} FH$
11         $c := c - \frac{1}{d+1} b$ // Updating the center of the ellipsoid
12         $F := \frac{d^2}{d^2-1}\left(F - \frac{2}{d+1} bb^T\right)$ // updating the basis of the ellipsoid
13     $E := \left\{x \in \mathbb{R}^d \middle| (x-c)^T F^{-1}(x-c) \leq 1\right\}$
14     $\tilde{V} :=$ vertices of $\frac{1}{d}(E - c) + c$ // compute the vertices of the enclosed
           version of $E$ in $L$
15     if $\tilde{V} \subseteq L$ then
16         Go to Line 27
17     $v := \arg\max_{x \in \tilde{V}} \|Ax\|_p$ // the farthest vertex of the contained
           ellipsoid from $L$ by the norm $\|Ax\|_p$
18     $grad := \|Av\|_p^{p-1} \cdot \|Av\|_{p-1}^{p-1} \cdot \left[\sum_{i=1}^{n} \frac{A_{i,1}v_1}{|A_{i*}^T v|}, \cdots, \sum_{i=1}^{n} \frac{A_{i,d}v_d}{|A_{i*}^T v|}\right]^T$ // The gradient
           of $\ell_p$-regression at $v$
19     $H := \frac{1}{\|grad\|_\infty} grad$ // Separating hyperplane between $v$ and $L$
       /* Here we apply the shallow cut ellipsoid update                          */
20     $z := \frac{1}{(d+1)^2}$
21     $\sigma := \frac{d^3(d+2)}{(d+1)^3(d-1)}$
22     $\zeta := 1 + \frac{1}{2d^2(d+1)^2}$
23     $\tau := \frac{2}{d(d+1)}$
24     $b := \frac{1}{\sqrt{H^T F H}} FH$
25     $F := \zeta\sigma\left(F - \tau\left(bb^T\right)\right)$
26     $c := c - zb$
27 $G := \mathrm{chol}\left(F^{-1}\right)$ // The Cholesky decomposition of $F^{-1}$
28 $(U, D, V) :=$ the $SVD$ of $G$
29 return $D, V$
```

References

1. Yu, X.; Liu, T.; Wang, X.; Tao, D. On compressing deep models by low rank and sparse decomposition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7370–7379
2. Acharya, A.; Goel, R.; Metallinou, A.; Dhillon, I. Online embedding compression for text classification using low rank matrix factorization. In Proceedings of the AAAI Conference on Artificial Intelligence, Hawaiian Village, HI, USA, 27 January–1 February 2019; Volume 33, pp. 6196–6203.
3. Wang, Y.X.; Ramanan, D.; Hebert, M. Growing a brain: Fine-tuning by increasing model capacity. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2471–2480.
4. He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.J.; Han, S. Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 784–800.
5. Luo, J.H.; Wu, J.; Lin, W. Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE International Conference On Computer Vision, Venice, Italy, 22–29 October 2017; pp. 5058–5066.
6. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. *Adv. Neural Inf. Process. Syst.***2013**, *2*, 3111–3119
7. Le, Q.; Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the PInternational Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 1188–1196.
8. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.
9. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
10. Dai, A.M.; Le, Q.V. Semi-supervised sequence learning. In Proceedings of the Neural Information Processing Systems, Montreal, QC, Canada, 7–10 December 2015; pp. 3079–3087.
11. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. Available online: https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf (accessed on 7 September 2020)
12. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MI, USA, 2–7 June 2019; pp. 4171–4186. [CrossRef]
13. Radiya-Dixit, E.; Wang, X. How fine can fine-tuning be? Learning efficient language models. In Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, Sicily, Italy, 3–5 June 2020; pp. 2435–2443.
14. Bermejo, S.; Cabestany, J. Oriented principal component analysis for large margin classifiers. *Neural Netw.* **2001**, *14*, 1447–1461. [CrossRef]
15. Wikipedia. Mean Squared Error—Wikipedia, The Free Encyclopedia. 2020. Available online: http://en.wikipedia.org/w/index.php?title=Mean%20squared%20error&oldid=977071088 (accessed on 7 September 2020).
16. Donoho, D.L. Compressed sensing. *IEEE Trans. Inf. Theory* **2006**, *52*, 1289–1306. [CrossRef]
17. Huang, X.; Liu, Y.; Shi, L.; Van Huffel, S.; Suykens, J.A.K. Two-level ℓ_1 minimization for compressed sensing. *Signal Process.* **2015**, *108*, 459–475. [CrossRef]
18. Donoho, D.L.; Elad, M. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proc. Natl. Acad. Sci. USA* **2003**, *100*, 2197–2202. [CrossRef] [PubMed]
19. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
20. Eckart, C.; Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* **1936**, *1*, 211–218. [CrossRef]
21. Clarkson, K.L.; Woodruff, D.P. Input sparsity and hardness for robust subspace approximation. In Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 17–20 October 2015; pp. 310–329.
22. Feldman, D.; Langberg, M. A unified framework for approximating and clustering data. In Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, San Jose, CA, USA, 6–8 June 2011; pp. 569–578
23. Code. Open Source Code for All the Algorithms Presented in this Paper. 2021. Available online: https://github.com/muradtuk/LzModelCompression (accessed on 7 September 2020).
24. Sanh, V.; Debut, L.; Chaumond, J.; Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv* **2019**, arXiv:1910.01108.
25. Chen, T.; Xu, B.; Zhang, C.; Guestrin, C. Training deep nets with sublinear memory cost. *arXiv* **2016**, arXiv:1604.06174.
26. Gomez, A.N.; Ren, M.; Urtasun, R.; Grosse, R.B. The reversible residual network: Backpropagation without storing activations. In Proceedings of the Neural Information Processing Systems, Long Beach, CA, USA, 4 December 2017; pp. 2214–2224.
27. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv* **2019**, arXiv:1910.10683.
28. McCarley, J.S. Pruning a bert-based question answering model. *arXiv* **2019**, arXiv:1910.06360.
29. Michel, P.; Levy, O.; Neubig, G. Are sixteen heads really better than one? In Proceedings of the Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 14014–14024.
30. Fan, A.; Grave, E.; Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv* **2019**, arXiv:1909.11556.

31. Guo, F.M.; Liu, S.; Mungall, F.S.; Lin, X.; Wang, Y. Reweighted proximal pruning for large-scale language representation. *arXiv* **2019**, arXiv:1909.12486.
32. Gordon, M.A.; Duh, K.; Andrews, N. Compressing BERT: Studying the effects of weight pruning on transfer learning. *arXiv* **2020**, arXiv:2002.08307.
33. Zafrir, O.; Boudoukh, G.; Izsak, P.; Wasserblat, M. Q8bert: Quantized 8bit bert. *arXiv* **2019**, arXiv:1910.06188.
34. Shen, S.; Dong, Z.; Ye, J.; Ma, L.; Yao, Z.; Gholami, A.; Mahoney, M.W.; Keutzer, K. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *AAAI* **2020**, *34*, 8815–8821 [CrossRef]
35. Zhao, S.; Gupta, R.; Song, Y.; Zhou, D. Extreme language model compression with optimal subwords and shared projections. *arXiv* **2019**, arXiv:1909.11687.
36. Tang, R.; Lu, Y.; Liu, L.; Mou, L.; Vechtomova, O.; Lin, J. Distilling task-specific knowledge from bert into simple neural networks. *arXiv* **2019**, arXiv:1903.12136.
37. Mukherjee, S.; Awadallah, A.H. Distilling transformers into simple neural networks with unlabeled transfer data. *arXiv* **2019**, arXiv:1910.01769.
38. Liu, L.; Wang, H.; Lin, J.; Socher, R.; Xiong, C. Attentive student meets multi-task teacher: Improved knowledge distillation for pretrained models. *arXiv* **2019**, arXiv:1911.03588.
39. Sun, S.; Cheng, Y.; Gan, Z.; Liu, J. Patient knowledge distillation for bert model compression. *arXiv* **2019**, arXiv:1908.09355.
40. Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; Liu, Q. Tinybert: Distilling bert for natural language understanding. *arXiv* **2019**, arXiv:1909.10351.
41. Sun, Z.; Yu, H.; Song, X.; Liu, R.; Yang, Y.; Zhou, D. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv* **2020**, arXiv:2004.02984.
42. Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv* **2019**, arXiv:1909.11942.
43. Wang, Z.; Wohlwend, J.; Lei, T. Structured pruning of large language models. *arXiv* **2019**, arXiv:1910.04732.
44. Maalouf, A.; Lang, H.; Rus, D.; Feldman, D. Deep Learning Meets Projective Clustering. *arXiv* **2020**, arXiv:2010.04290.
45. Gordon, M.A. All The Ways You Can Compress BERT. Available online: http://mitchgordon.me/machine/learning/2019/11/18/all-the-ways-to-compress-BERT.html (accessed on 15 July 2021).
46. Cohen, M.B.; Nelson, J.; Woodruff, D.P. Optimal approximate matrix product in terms of stable rank. *arXiv* **2015**, arXiv:1507.02268.
47. Clarkson, K.L.; Woodruff, D.P. Low-rank approximation and regression in input sparsity time. *J. ACM* **2017**, *63*, 1–45. [CrossRef]
48. Meng, X.; Mahoney, M.W. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 1–4 June 2013; pp. 91–100.
49. Nelson, J.; Nguyên, H.L. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 26–29 October 2013; pp. 117–126.
50. Shyamalkumar, N.D.; Varadarajan, K. Efficient subspace approximation algorithms. *SODA* **2007**, *7*, 532–540. [CrossRef]
51. Ding, C.; Zhou, D.; He, X.; Zha, H. R 1-PCA: rotational invariant L 1-norm principal component analysis for robust subspace factorization. In Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 25–29 June 2006; pp. 281–288.
52. Chierichetti, F.; Gollapudi, S.; Kumar, R.; Lattanzi, S.; Panigrahy, R.; Woodruff, D.P. Algorithms for ℓ_p Low-Rank Approximation. *Int. Conf. Mach. Learn.* **2017**, *34*, 806–814
53. Candès, E.J.; Li, X.; Ma, Y.; Wright, J. Robust principal component analysis? *J. ACM (JACM)* **2011**, *58*, 1–37. [CrossRef]
54. Netrapalli, P.; UN, N.; Sanghavi, S.; Anandkumar, A.; Jain, P. Non-convex robust PCA. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 1107–1115.
55. Yi, X.; Park, D.; Chen, Y.; Caramanis, C. Fast algorithms for robust PCA via gradient descent. *Adv. Neural Inf. Process. Syst.* **2016**, *30*, 4152–4160.
56. Feldman, D.; Monemizadeh, M.; Sohler, C.; Woodruff, D.P. Coresets and sketches for high dimensional subspace approximation problems. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, USA, 17–19 January 2010; pp. 630–649.
57. Varadarajan, K.; Xiao, X. On the Sensitivity of Shape Fitting Problems. In Proceedings of the 32nd International Conference on Foundations of Software Technology and Theoretical Computer Science, Hyderabad, India, 15–17 December 2012; p. 486.
58. Feldman, D.; Volkov, M.; Rus, D. Dimensionality reduction of massive sparse datasets using coresets. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 2766–2774
59. Maalouf, A.; Statman, A.; Feldman, D. Tight sensitivity bounds for smaller coresets. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Goa, India, 14–18 December 2020; pp. 2051–2061.
60. Maalouf, A.; Jubran, I.; Tukan, M.; Feldman, D. Faster PAC Learning and Smaller Coresets via Smoothed Analysis. *arXiv* **2020**, arXiv:2006.05441.
61. Maalouf, A.; Jubran, I.; Feldman, D. Fast and accurate least-mean-squares solvers. *Adv. Neural Inf. Process. Syst.* **2019**, *33*, 8307–8318.
62. Jubran, I.; Maalouf, A.; Feldman, D. Introduction to coresets: Accurate coresets. *arXiv* **2019**, arXiv:1910.08707.

63. Tukan, M.; Maalouf, A.; Feldman, D. Coresets for near-convex functions. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 4.
64. John, F. Extremum problems with inequalities as subsidiary conditions. In *Traces and Emergence of Nonlinear Programming*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 197–215.
65. Clarkson, K.L.; Drineas, P.; Magdon-Ismail, M.; Mahoney, M.W.; Meng, X.; Woodruff, D.P. The fast cauchy transform and faster robust linear regression. *SIAM J. Comput.* **2016**, *45*, 763–810. [CrossRef]
66. Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.R.; Le, Q.V. Xlnet: Generalized autoregressive pretraining for language understanding. *Adv. Neural Inf. Process. Syst.* **2019**, *1*, 5753–5763.
67. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.G.; Le, Q.; Salakhutdinov, R. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; pp. 2978–2988.
68. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
69. Van Der Walt, S.; Colbert, S.C.; Varoquaux, G. The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Eng.* **2011**, *13*, 22. [CrossRef]
70. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv* **2019**, arXiv:1910.03771v5.
71. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. NIPS-W. 2017. Available online: https://openreview.net/pdf?id=BJJsrmfCZ (accessed on 15 July 2021).
72. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Brussels, Belgium, 1 November 2018; pp. 353–355.
73. Markopoulos, P.P.; Karystinos, G.N.; Pados, D.A. Optimal algorithms for L_{1}-subspace signal processing. *IEEE Trans. Signal Process.* **2014**, *62*, 5046–5058. [CrossRef]
74. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
75. Tanaka, H.; Kunin, D.; Yamins, D.L.; Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 13.
76. Grötschel, M.; Lovász, L.; Schrijver, A. The Ellipsoid Method. In *Geometric Algorithms and Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 1993; pp. 64–101.
77. Golub, G.H.; Van Loan, C.F. *Matrix Computations*; JHU Press: Baltimore, MD, USA, 2012; Volume 3.