Low Rank Pruning via Output Perturbation

Yuhan Liu, Siddharth Joshi, Baharan Mirzasoleiman

Abstract

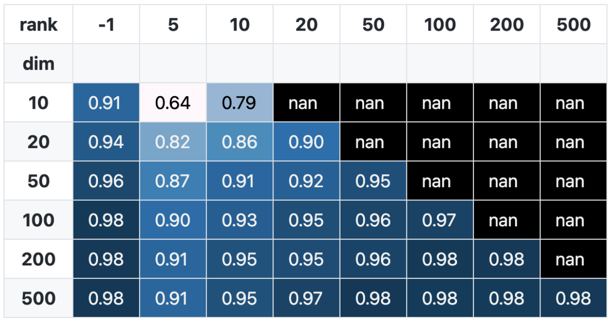
Neural networks have become very widespread due to the mainstream availability of computational devices such as GPUs, and as these devices become more powerful, these networks have become much larger. With the growing demand for fast, efficient networks, weight pruning has become a popular technique for reducing both the speed and computational time of these networks, but they introduce sparse matrices, which can be tedious to implement properly. In this paper, we investigate a different approach to model pruning involving low rank decompositions and output perturbation.

1 Introduction

Neural network pruning comes in several forms. Some methods like [4] and [7] aim to prune the connections between adjacent layers at initialization (or close to initialization); other methods like [8] prune entire filters of convolutions; lastly, there are methods such as [5], [1], and [6] that prune using low rank decomposition. We investigate the effectiveness of using low rank decompositions in the context of high sparsities as well as a simple blocking technique similar to that used in [5], [6], and [1] to achieve competitive results.

In many pruning methods, the layer pruning step can be motivated by the Eckart–Young–Mirsky theorem, which guarantees the minimal difference in both spectral norm and Frobenius norm of a single layer in the pruned network. However, as noted in [5] and [3], the trickier aspect of multi-layer network pruning lies in rectifying the pruning ratios across different layers–it's not a straightforward problem to decide how much each layer should be pruned.

Towards that endeavor, we explore two approaches that serve to score all singular vectors on the same metric: **output perturbation** and **output variance**. The main idea stems from recognizing that singular values are not comparable across layers; due to the scale invariant property of neural networks, one layer's singular values may by much larger than other layers', but it may not indicate higher importance.



rank	-1	5	10	20	50	100	200	500
dim								
10	0.91	0.64	0.79	nan	nan	nan	nan	nan
20	0.94	0.82	0.86	0.90	nan	nan	nan	nan
50	0.96	0.87	0.91	0.92	0.95	nan	nan	nan
100	0.98	0.90	0.93	0.95	0.96	0.97	nan	nan
200	0.98	0.91	0.95	0.95	0.96	0.98	0.98	nan
500	0.98	0.91	0.95	0.97	0.98	0.98	0.98	0.98

Figure 1: MNIST validation accuracy where the last layer's size and rank was varied

2 Motivation

We begin by analyzing some of the motivating factors regarding low rank neural networks. In [2], the authors show that neural networks naturally learn a low rank representation of the data. More notably, the linear function

$$y = W_n W_{n-1} \dots W_1 x \tag{1}$$

when trained using stochastic gradient descent will converge to a low rank representation of the effective weights $W_e = \prod_{i=1}^{n} W_i$

We also empirically observed this behavior by training a small neural network where the last layer's rank and size was varied across different training runs. From figure 1, we can see that in many settings, a low rank model was sufficient to achieve the same accuracy as a full rank model.

We've also observed that training at rank 14 for any period of time before switching to a different rank had no effect on the final validation accuracy. However, training at any rank below 14 for 50 epochs would harm the validation accuracy (see figure 2). This implies that the network's last layer exhibits an optimal rank of 14.

3 Problem Formulation

We define a neural network as the following function

$$y = \sigma(W_n \sigma(W_{n-1} \cdots \sigma(W_1 x))) \tag{2}$$

where there are n layers and $\sigma(\cdot)$ represents the activation function.

3.1 Convolutions as Matrix Multiplications

To incorporate convolutions into our framework, we treat a convolution with C_o filters, C_i input channels, and kernel size $W \times H$ as a matrix multiplication of an

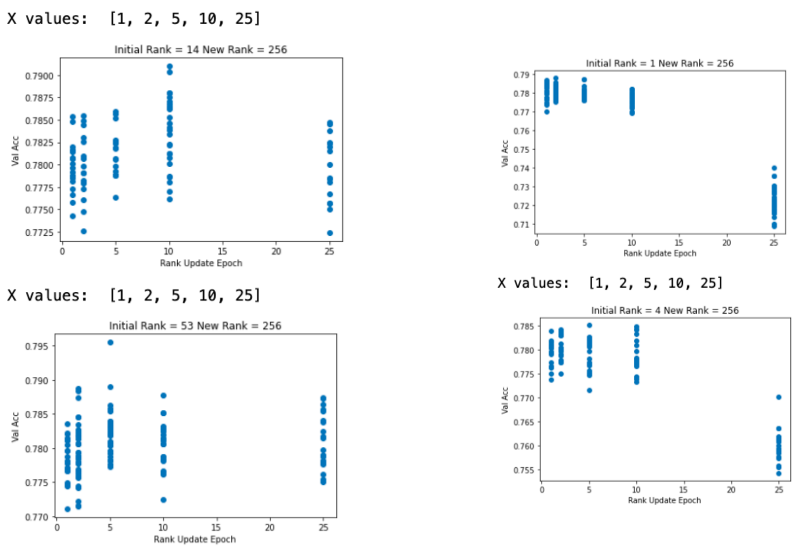

```
X values:  [1, 2, 5, 10, 25]
X values:  [1, 2, 5, 10, 25]
X values:  [1, 2, 5, 10, 25]
```

Figure 2

input image patch with a matrix of dimensions $C_o \times (C_i W H)$. This definition is consistent with other works that deal with low rank convolutions such as [5]. From this point forward, we'll simply refer to this matrix as W_i, akin to the dense layers in the network.

We'd like to point out that in PyTorch[1] and TensorFlow[2], convolutions are implemented first by unfolding the input into column vectors and then right multiplying each vector by the aforementioned matrix W_i. The unfolding operation, commonly known as "Im2Col", takes each patch of the input image and flattens it into a vector[3].

Because convolutions end up implemented as matrix multiplications, the rest of the paper will treat each W_i matrix as a 2D matrices (and not tensors).

3.2 Blocking

Before decomposing weight matrices as left and right singular vectors, we first perform a *blocking* operation on the weight matrix to allow for more flexibility in choosing weights. Each block of the weight matrix becomes treated as a its own independent matrix to be low rank approximated. The motivation is partially outlined in [6], but we also show empirical validation that this technique helps with low rank decomposition. Our low rank pruning techniques are thus done

[1] `https://github.com/pytorch/pytorch/blob/master/aten/src/ATen/native/Im2Col.cpp`

[2] `https://github.com/tensorflow/tensorflow/blob/8a20d54a3c1bfa38c03ea99a2ad3c1b0a45dfa95/tensorflow/python/ops/nn_ops.py#L2332`

[3] `https://towardsdatascience.com/how-are-convolutions-actually-performed-under-the-hood-226523ce7fbf`

on blocked weight matrices, whose details are described below.

3.2.1 ALDS Blocking

Inspired by the use of channel blocking in [5], we implement a simplified version of their blocking scheme by splitting each convolution's input channels into $k = 4$ subspaces. The choice is backed by the ablation study conducted in [5], where they noted that fixing a constant k value worsens performance but not significantly.

3.2.2 Square Blocking

Square blocking of a $n \times m$ weight matrix is motivated by solving the minimization problem

$$\underset{\substack{(p_i,q_i)\\ \mathcal{B}}}{\arg\min} \sum_{i \in \mathcal{B}} r \cdot (p_i + q_i) \tag{3}$$

where $\mathcal{B}$ is the set of blocks, (p_i, q_i) represents the size of block i and r is the rank of each block.

To solve this, we first assume that each block is of a fixed size $p_i q_i = C_i$. By the AM-GM inequality, the quantity $r \cdot (p_i + q_i)$ is minimized when $p_i = q_i = \sqrt{C}$. Next, by considering the sparsity of each block

$$1 - \frac{r \cdot (p_i + q_i)}{p_i q_i} = 1 - \frac{2r}{C} \tag{4}$$

we can see that it is maximized when $\sqrt{C}$ is maximized. Therefore, by creating the largest blocks per weight matrix whose dimensions are equal to each other, we arrive at the square blocking scheme.

4 Scoring Importance of Singular Vectors

Let y be the output of the original network and y_{b_i} be the output of the network where singular vector i in block b is pruned.

4.1 Output Perturbation

The limitation of using the magnitude of singular values as a metric for the importance of the singular vectors is that singular vectors corresponding to large singular values could potentially be *aligned* with singular vectors corresponding to small singular values in its input and/or subsequent layers. The converse is also possible. In such cases, the magnitude of the singular value is not a good metric for the importance of a singular vector both within a layer and across layers. Output perturbation addresses this by measuring the magnitude of the contribution of a given singular vector to the final output.

In particular, the importance score of singular vector i of block b is defined as $\|y - y_{b_i}\|^2$

In practice, this score is computed by taking the mean of the aforementioned expression over a subset of the data to be computationally efficient.

4.2 Output Variance

An alternative approach to scoring singular vectors is by quantifying how much *variance* they provide the model.

Since the network has a vector output, there are different approaches that can be used to measure the *variance* contributed by a singular vector.

A simple yet effective (validated empirically) approach is to measure the variance of $y - y_{b_i}$ component-wise across a subset of data.

5 Results

We evaluate our techniques on a VGG16 network with CIFAR10. There are three blocking schemes: alds, square, and no blocking; and four pruning algorithms: magnitude, relative error, alignment output, and alignment variance. We also compare our results with ALDS ([5]).

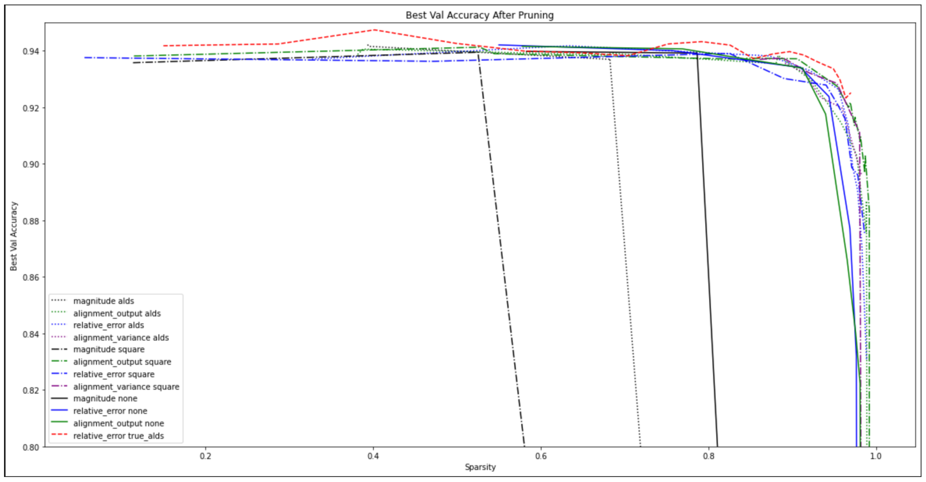
The first observation from figure 3 is that magnitude based pruning leads to layer collapse very early on. This supports the scale invariant hypothesis mentioned previously where some layers are scaled more than others as a result of training.

Next, we observe that blocking (of both simple ALDS and square blocking) improves accuracy at the 95% and above sparsity regime. More so, we note that square blocking creates more blocks than ALDS (around 400 blocks for square blocking and 50 for ALDS). Although the pure ALDS code performs better than our method, we would like to point out that ALDS takes 2 minutes to search over a large number of block sizes, whereas square blocking is computed in less than a second.

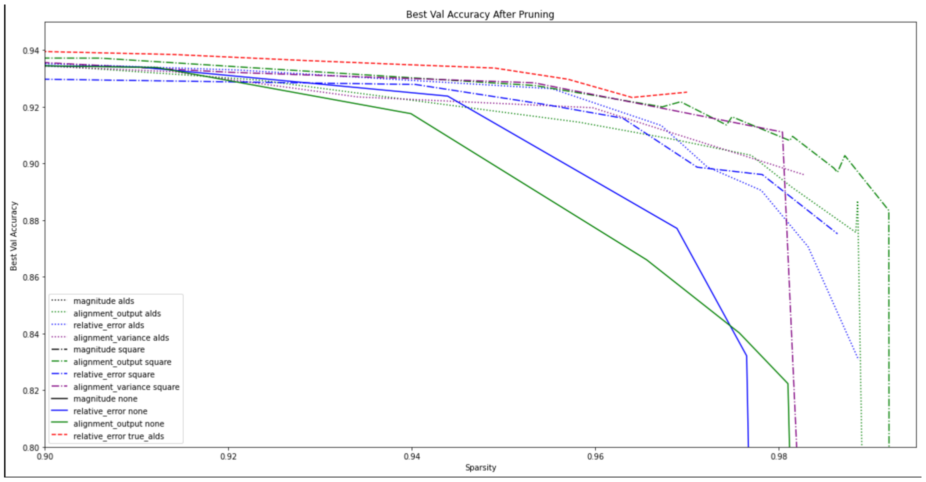
6 Future Work

There are several orthogonal directions that can be explored further from this project.

1. **Alignment pruning** One shortcoming of alignment pruning is that networks like VGG16 have thousands of singular vectors. This makes the process of testing each singular vector quite time consuming ($\sim$ 10 minutes). A possible direction to explore is efficient stochastic algorithms where singular vectors are sampled from a distribution. Alternatively, our analysis show that between alignment pruning and relative error pruning, there's a Spearman's rank correlation coefficient of 0.87. This high correlation suggests one may only need to perform alignment pruning/scoring on a subset of singular values, noticeably those with neither very high nor small singular values.



(a) Pruning results at all sparsities



(b) Pruning results at high sparsities

Figure 3: Pruning results for magnitude, relative error, alignment, and baseline ALDS.

2. **Blocking schemes** The square blocking scheme yields promising results, but can be expanded on through permutations of input and output neurons for more effective blocking. For example, [6] and [1] permute output neurons of an embedding layer by clustering or examining word frequencies. Another approach is to look at sets of outputs per layer that are closely correlated with each other and group these into one block. Then, one can expect the transformation whose image is those blocks to be low rank by definition.

3. **Blocked matrix multiplication implementation** A more technical direction would be to implement low rank layers in an efficient form within PyTorch and TensorFlow to measure actual speedups. Theoretically, the number of operations is improved by a factor of $\frac{nm}{r\cdot(n+m)}$, which for small ranks should see a significant speedup. Even though the left and right singular vectors must be multiplied sequentially, we can increase the batch size such that per forward/backward pass, the net throughput is still increased.

References

[1] Patrick Chen et al. "GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: `https://proceedings.neurips.cc/paper/2018/file/a2b8a85a29b2d64ad6f47275bf1360c6-Paper.pdf`.

[2] Minyoung Huh et al. *The Low-Rank Simplicity Bias in Deep Networks*. 2022. arXiv: `2103.10427 [cs.LG]`.

[3] Jaeho Lee et al. "Layer-adaptive Sparsity for the Magnitude-based Pruning". In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=H6ATjJ0TKdf`.

[4] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. "SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY". In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=B1VZqjAcYX`.

[5] Lucas Liebenwein et al. "Compressing Neural Networks: Towards Determining the Optimal Layer-wise Decomposition". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: `https://openreview.net/forum?id=BvJkwMhyInm`.

[6] Alaa Maalouf et al. "Deep Learning meets Projective Clustering". In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=EQfpYwF3-b`.

[7] Chaoqi Wang, Guodong Zhang, and Roger Grosse. “Picking Winning Tickets Before Training by Preserving Gradient Flow”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SkgsACVKPH.

[8] Shixing Yu et al. *Hessian-Aware Pruning and Optimal Neural Implant*. 2021. arXiv: 2101.08940 [cs.CV].