Neural Network Parameter Reduction Using Pruning and Matrix Decomposition

Christopher Rohlicek

May 2020

Abstract

Much work in the field of machine learning research is devoted to reaching the peak performance of a network on a given classification task, often resulting in overparameterized networks which are a ubiquitous byproduct of the experiments and innovations that have been made over the years. To address computational and performance issues associated with overparameterization, approaches to reduction of model size have become an increasingly popular topic of research, and one which presents many applications of different areas of linear algebra to lend insight into this problem.

Different methods of model size reduction, including those based on rank reduction and pruning are evaluated in the context of convolutional and fully connected sections of the Lenet-5 [1] architecture used in the classification task of identifying images of handwritten letters in the EMNIST dataset [2]. One goal is to compare very different approaches to model reduction, for example, to determine if solely the number of parameters is significant or if the properties of the remaining parameters have notable impact.

Results show that fully connected layers are well approximated by significantly reduced matrices but removal of singular values has significant detrimental effects on the classification power of the original layer. Additionally, we see that size reductions applied to convolutional layers in this context better preserve overall accuracy of the network, but contribute less to overall network size reduction.

1 Introduction

Highly accurate neural networks are often associated with large numbers of parameters, which is commonly a structural result of the networks themselves. Potential impacts of the number of parameters are overfitting to a limited set of training data, and excess computation costs in the forms of storage size and numerical operations both in training and ultimate use of the network. As these issues may be avoided by using smaller networks, research is expanding into different methods of reducing the size of networks while maintaining the results found by large and more highly parameterized structures. This goal has received even more more attention with a goal of creating networks that are lightweight enough to use on small devices such as smartphones, watches, and embedded devices.

Various approaches to reducing the memory size and computational intensity involved in implementing a neural network for practical use have been explored in the literature. A notable success is found in [3], which shows different contexts in which considerably "pruned" networks approximate their original counterparts to a small margin of accuracy. The main method of pruning in [3] is done by removing connections with weights that are sufficiently small in magnitude, such that only a subset of the original network connections remain, after which the pruned structure is trained further.

In this paper we compare pruning with rank-reduction for reducing the number of connections (and therefore parameters) in a neural network which employ the singular value decomposition (SVD) to reduce the number of connections in fully connected layers, and the CP-decomposition [4] to reduce the number of connections in convolutional layers.

The baseline model used is the Lenet-5 [1] architecture which comprises a two-layer convolutional section with interspersed max pooling, followed by three fully connected layers with ReLU activation functions and a final softmax function to generate probabilities over the 26 letter classes grouping upper and lower case examples. The classification task used is the recognition of images of handwritten letters from the EMNIST dataset [2].

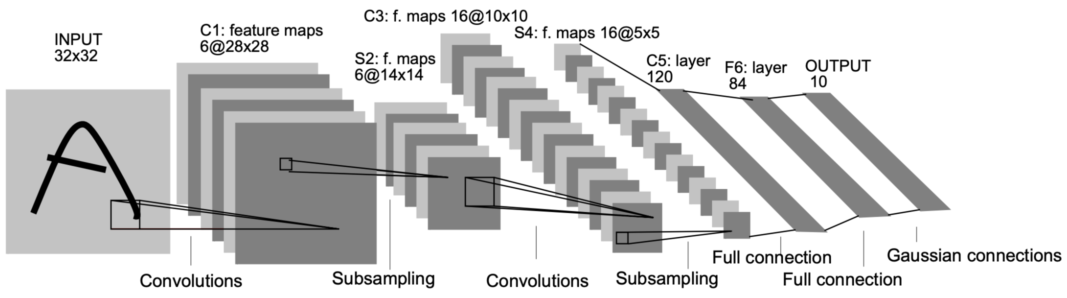

Figure 1: Baseline Architecure: Lenet-5 [1]

Baseline Model

As a baseline model we implement the Lenet-5 architecture as a classifier on the EMNIST dataset of handwritten letters. Based on the labeling in Figure 1, we will use C_1, C_2 to refer to the two convolutional kernels that are applied to the respective convolutional layers' inputs, and F_1, F_2, F_3 to refer to the matrices of weights applied to the respective fully connected layers' inputs, and S will represent the combination of ReLU nonlinear activation function and max pooling applied to the output of the two convolutional layers. With this notation we summarize the action of the baseline network on an input I with the following functional notation (for $\cdot$ representing matrix multiplication, and $*$ representing linear convolution):

$$\text{Output} = F_3 \cdot (ReLU(F_2 \cdot (ReLU(F1 \cdot (S(C_2 * (S(C_1 * I))))))))^1 \qquad (1)$$

Pruning Methods

Magnitude-Based Pruning. The first model reduction approach is a magnitude-based pruning method in which links are removed from the network if their weights fall below a certain threshold. Motivated by [3][2], removal of a link is simulated by assigning a "pruned" weight a value of zero, and adapting the training method to zero-out the corresponding partial derivatives of the gradient

[1]For clarity we diverge from LeCun's exact notation and index the fully connected and convolutional matrices with regard to their respective sections of the network, and we forgoe indexing the nonlinearities, as they remain unchanged in our experiments.

[2]We diverge from [3] in choosing to resume training with the remaining values in their current state after pruning rather than restoring them to their original weights.

in each parameter update step to ensure that the "removed" weights maintain their value of zero.

For the model's k^{th} iteration output $\mathbf{x_k}$ and step size γ, consider the optimization step given by,

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \gamma \nabla F(\mathbf{x_k}) \tag{2}$$

The training method that maintains our pruned weight matrix then includes a step in which the matrix $\nabla F(\mathbf{x_k})$ of partial derivatives of the loss function F is multiplied element-wise (an operate we will represent below with the Hadamard product: $\odot$) by the mask $\mathbf{M}$ of the pruned weight matrix (the binary indicator matrix identifying the locations of nonzero weights) to zero-out all elements of $\nabla F(\mathbf{x_k})$ that correspond to pruned weights:

$$\nabla F(\mathbf{x_k}) \leftarrow \nabla F(\mathbf{x_k}) \odot \mathbf{M} = \begin{bmatrix} \frac{\partial F(\mathbf{x_k})}{\partial x_{1,1}} & \cdots & \frac{\partial F(\mathbf{x_k})}{\partial x_{1,m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial F(\mathbf{x_k})}{\partial x_{n,1}} & \cdots & \frac{\partial F(\mathbf{x_k})}{\partial x_{n,m}} \end{bmatrix} \odot \begin{bmatrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,m} \end{bmatrix} \tag{3}$$

$$\text{For the elements of } \mathbf{M} \text{ given by: } b_{i,j} = \begin{cases} 0 & x_{i,j} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

In implementation, this in step is incorporated into the training method as follows:

Algorithm 1: Training method to maintain "removed" connections

```
Result: Training while preserving zero-valued weights
Masks $\mathbf{M_p}$ are passed in to training method;
for number of epochs do
    for training data samples do
        output calculated;
        loss $F$ calculated;
        for pruned matrices $\mathbf{P}$ do
            $\nabla F(\mathbf{P}) \leftarrow \nabla F(\mathbf{P}) \odot \mathbf{M_p}$
        end
        for weight matrices $\mathbf{W}$ do
            $\mathbf{W} \leftarrow \mathbf{W} - \gamma \nabla F(\mathbf{W})$
        end
    end
end
```

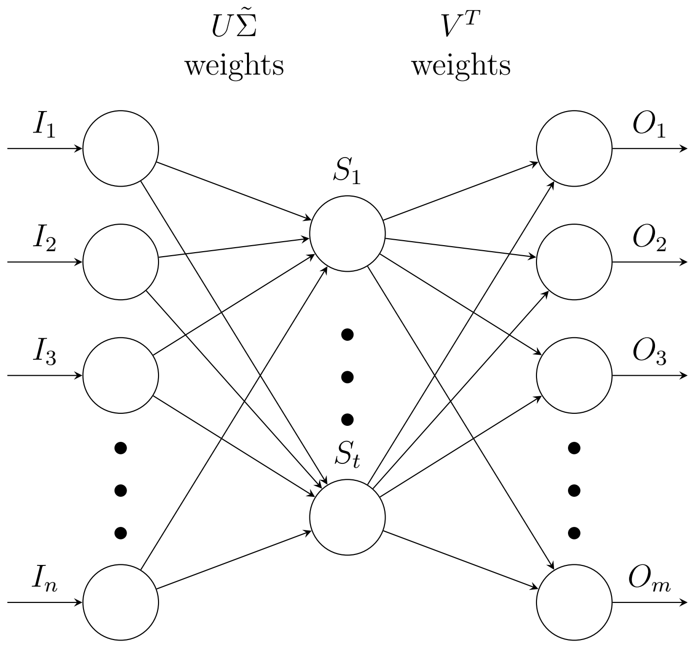

Figure 2: SVD Decomposition on $m \times n$ dense layer

The threshold used to remove links is determined by a desired proportion of weights to be kept in each experiment. The result of a matrix being pruned to $p\%$ of its initial number of weights is the corresponding layer of the network keeping only $p\%$ of the original number of connections it had with the subsequent layer. This method is applied equivalently to both convolutional and fully connected layers.

Fully connected SVD-pruning. For a given layer's matrix of weights W, we can apply singular value decomposition to decompose the matrix as follows:

$$W = U\Sigma V^T \tag{5}$$

Using these matrices, we divide the original layer into two fully connected layers characterized by the product $U\tilde{\Sigma}$ and the matrix V^T. The model size reduction is dictated in this decomposition through the matrix $\tilde{\Sigma}$, which is a diagonal matrix containing only the $p\%$ largest singular values of the original matrix. Implementing this size reduction with the modified architecture shown in Figure 2, the dimensionality reduction of $\tilde{\Sigma}$ results in $(n \times t) + (t \times m)$ parameters taking the place of the original $n \times m$ (for t the number of remaining singular values in $\tilde{\Sigma}$). This method is tested by applying it to both of the fully connected layers in our baseline model, and continuing training once the change is made. Referring back to equation 1, F_i is replaced with matrices F_{i,V^T} and $F_{i,U\tilde{\Sigma}}$, thus giving us the following expression for the fully connected section of the network:

$$\text{Output} = F_{3,U\tilde{\Sigma}} \cdot F_{3,V^T} \cdot (ReLU(F_{2,U\tilde{\Sigma}} \cdot F_{2,V^T} \cdot (ReLU(F_{1,U\tilde{\Sigma}} \cdot F_{1,V^T} \cdot (I))))) \quad (6)$$

CP-Decomposition on Convolutional Layers. Taking similar motivation from pruning via the removal of singular values of dense weight matrices, we reduce the number of parameters in convolutional layers through application of CP-decomposition [4]. CP-decomposition leverages the fact that a tensor of dimension n can be decomposed into the sum of outer products of n 1-dimensional vectors. The number of terms in the sum will be equal to the rank of the tensor. Below, we show the expression for a 4-dimensional tensor (analogous to the tensors that compose the convolution section of our network):

$$W = a_1 \circ b_1 \circ c_1 \circ d_1 + ... + a_R \circ b_R \circ c_R \circ d_R \quad (7)$$

In implementation, CP-decomposition replaces a given convolutional layer (a 4-dimensional tensor) with four reduced tensors, to an approximation level dictated by a specified rank parameter. Referring back to equation 1, C_i is replaced with tensors $C_{i,x}$, $C_{i,y}$, $C_{i,s}$, and $C_{i,t}$ [5], thus giving us the following expression for the convolutional section of the network:

$$\text{Output} = S((C_{2,x} \otimes C_{2,y} \otimes C_{2,s} \otimes C_{2,t}) * (S((C_{1,x} \otimes C_{1,y} \otimes C_{1,s} \otimes C_{1,t}) * I)))) \quad (8)$$

For a convolutional tensor C_i of size (S, T, d, d), $C_{i,x}$ will be of size $(R, S, 1, 1)$, $C_{i,y}$ will be of size $(R, 1, d, 1)$, $C_{i,s}$ will be of size $(R, 1, 1, d)$, and $C_{i,t}$ will be of size $(T, R, 1, 1)$. For a decomposition of rank 5, this results in a 50.2% decrease in size of the convolutional section (918 connections to 457).

In Figure 3 we see an example of how a convolution from S input channels to T output channels is decomposed at rank R. In the example of the first convolutional layer of Lenet-5, $S = 1$ and $T = 6$ would be the input and output dimensions and $d = 3$ would be the side length of the square convolution window.

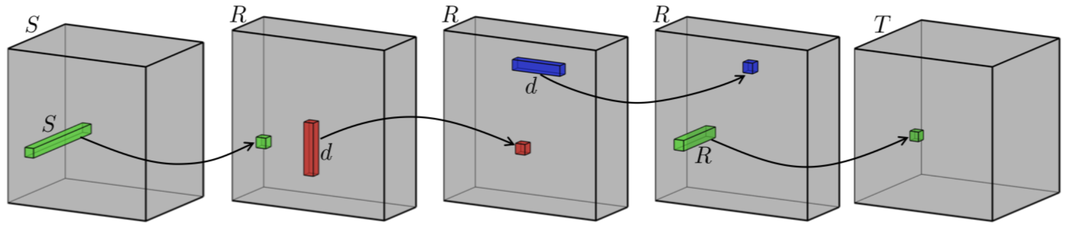

Figure 3: CP-decomposition on a $S \times T \times d \times d$ convolution [5]

2 Experimental Setup

To compare these different pruning methods, we ran four different experiments to test the performance of models created via magnitude- and SVD-based size reduction on the fully connected section of the network, and magnitude- and CP-decomposition-based size reduction methods on the convolutional section. For each of these four experiments we generated models at varying levels of sparsity. Each model was generated by applying one of the pruning methods to the same fully-trained instance of Lenet-5 that was loaded as a benchmark into each experiment.

The models generated by the magnitude-based approaches retain 5%, 10%, 20%, 50%, and 75% of the original number of connections in the respective sections of the network. For the SVD-based approach, the dimension of the reduced matrix $\tilde{\Sigma}$ was determined by retaining 5%, 10%, 20%, 50%, and 75% of the original matrix's singular values (corresponding to 6.28%, 12.57%, 25.27%, 63.74%, and 95.54% of the original number of connections in the fully connected section). For the CP-decomposition method, models were generated with the decomposition applied to the convolutional section at ranks 1, 5, 10, and 15 (corresponding to 4.47%, 22.33%, 44.66%, and 66.99% of the original number of connections in the convolutional section).

Once each new model was generated, they were trained starting from their current state after size reduction. All generated models were trained using early stopping on the criterion that validation loss not decrease for 10 epochs.

Magnitude-pruning on Fully Connected layers			
Percent weights remaining	Parameters in Network	Number of Epochs Trained	Test Accuracy
5%	5240	54	88.173%
10%	9309	14	88.635%
20%	17447	11	88.697%
50%	41862	15	88.216%
75%	62208	11	88.514%

Table 1: Fully Connected Pruning Results

3 Results

Tables 1 through 4 summarize metrics gathered from the experiments described above. In comparing the results it is worth noting the relative sizes of the fully connected and convolutional sections of the network in terms of number of free parameters. While these pruning methods were applied uniformly across the two types of layers, the results must be contextualized within the overall impact on the network as a whole (hence the inclusion of the number of parameters in each network to give a better picture of how each change effected the total number of connections).

Applying an early-stopping protocol to the training method used for each of these models had the twofold benefit of making sure that each model was able to give a measure of fully-trained accuracy against the test data, and providing us with the number of epochs the model took to restore itself to optimal performance after the pruning was performed.

4 Discussion

In considering the results shown in Figure 4, it appears that both magnitude-based and matrix decomposition-based size reductions methods work similarly

Magnitude-pruning on Convolutional layers			
Percent weights remaining	Parameters in Network	Number of Epochs Trained	Test Accuracy
5%	81683	44	86.880%
10%	81729	14	90.697%
20%	81820	10	92.12%
50%	82095	10	92.014%
75%	82325	10	92.351%

Table 2: Convolutional Pruning Results

SVD-based pruning on Fully Connected layers			
Percent singular values remaining	Parameters in Network	Number of Epochs Trained	Test Accuracy
5%	6283	36	39.918%
10%	11396	38	84.740%
20%	21733	26	91.745%
50%	53043	14	92.649%
75%	78924	14	92.668%

Table 3: SVD Reduction Results

CP-decomposition on Convolutional layers			
Decomposition Rank	Parameters in Network	Number of Epochs Trained	Test Accuracy
1%	81677	15	88.053%
5%	81841	12	92.096%
10%	82046	10	91.779%
15%	82251	12	92.625%

Table 4: CP-Decomposition Results

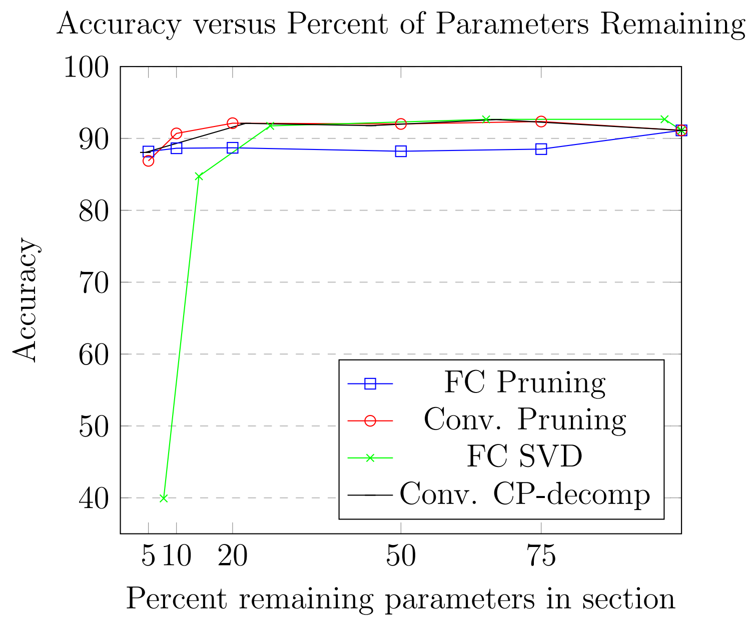

Figure 4: Results of size reductions with respect to their specific section of the network

on the two sections of the network. An important result is shown by the fact that reducing the number of singular values to 5% caused that model's accuracy to drop far below the model of comparable size in the fully connected section. This supports the hypothesis that maintaining the effects of a fully connected layer's singular values is more important to the classification accuracy of that layer than the sheer number of connections preserved. Another important conclusion to be drawn from the comparable performance of the pruning methods on the two sections of the network is that pruning on fully connected layers can sustain considerable reductions in size while preserving overall accuracy to a fairly small margin.

Additionally, it appears that relative to the separate sizes of the fully connected and convolutional layers, magnitude pruning was more effective per share of the total section's parameters when applied to the convolutional section than the fully connected section. Further, we see that both methods of size reduction applied to the convolutional section of the network perform very similarly. Future work would address this phenomenon and determine whether it is simply a result of the relative small size of the convolutional layers.

A System Description

Appendix 1 – The python files for the described experiments are in the GitHub repository: `https://github.com/crohlicek/model_size_reduction`. The README describes the contents of each python script.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “Emnist: Extending mnist to handwritten letters,” in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[3] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” *arXiv preprint arXiv:1803.03635*, 2018.

[4] G. Strang, *Linear algebra and learning from data.* Wellesley-Cambridge Press, 2019.

[5] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned cp-decomposition,” *arXiv preprint arXiv:1412.6553*, 2014.