



Retraining Conditions: How Much to Retrain a Network After Pruning?

Soumya Sara John^(✉), Deepak Mishra, and J. Sheeba Rani

Indian Institute of Space Science and Technology,
Thiruvananthapuram 695547, India
soumyasara100@gmail.com,
{deepak.mishra,sheeba}@iist.ac.in

Abstract. Restoring the desired performance of a pruned model requires a fine-tuning step, which lets the network relearn using the training data, except that the parameters are initialised to the pruned parameters. This relearning procedure is a key component in deciding the time taken in building a hardware-friendly architecture. This paper analyses the fine-tuning or retraining step after pruning the network layer-wise and derives lower bounds for the number of epochs the network will take based on the amount of pruning done. Analyses on the propagation of errors through the layers while pruning layer-wise is also performed and a new parameter named ‘Net Deviation’ is proposed which can be used to estimate how good a pruning algorithm is. This parameter could be an alternative to ‘test accuracy’ that is normally used. Net Deviation can be calculated while pruning, using the same data that was used in the pruning procedure. Similar to the test accuracy degradation for different amounts of pruning, the net deviation curves help compare the pruning methods. As an example, a comparison between Random pruning, Weight magnitude based pruning and Clustered pruning is performed on LeNet-300-100 and LeNet-5 architectures using Net Deviation. Results indicate clustered pruning to be a better option than random approach, for higher compression.

Keywords: Pruning · Model compression · Deep neural networks · Retraining network

1 Introduction

Neural Networks have been widely used to solve different real world problems in different arenas because of its remarkable function approximation capability. Recently, Deep Neural Networks have attained much awaited attention across various discipline. Despite of all the advances in the field of Deep Learning, including the arrival of better and improved optimization algorithms and GPUs, several questions are still puzzling. One of them is the optimum architecture size. This includes, how one should decide the number of nodes and number of layers for a network to solve a particular problem using the data set. Both the deep and

shallow networks have their own share of advantages and disadvantages. This makes it difficult for the network designer to create the optimum architecture. Shallow networks exhibit better generalization performance and learn faster, but have a higher tendency to overfit [5]. Deeper networks like LeNet-300-100 or LeNet-5 [12], form complex decision boundaries but avoids overfitting.

Model Compression techniques are inspired by the fault tolerance property to network damage conditions, seen in larger networks. Pruning is one such model compression technique. Introducing damage to the network purposefully will compromise its performance in terms of accuracy. However, a procedure of retraining can be used to regain the original performance. In general, the percentage reduction in accuracy is proportional to the amount of damage made. When the damage to the network is rigorous, the network requires more retraining to regain the desired accuracy on the particular data set. [18] conducted experiments comparing the accuracy of large, but pruned models (large-sparse) with their smaller, but dense (small-dense) counterparts and gave results stating that the large-sparse models outperforms the small-dense models with $10\times$ reduction in the number of non zero parameters with minimal reduction in accuracy.

Major Contributions. The contributions of this paper are summarized as follows: (1) We derive theoretical bounds for the number of epochs a pruned network will require to reach the original performance, relative to the number of epochs the original unpruned network had taken to reach the same performance. (2) We derive a relation bounding the error that will be present at the output based on layer-wise error propagation due to the pruning done in different layers. (3) A new parameter ‘Net Deviation’ is proposed, that could serve as a measure to select the appropriate pruning method for a particular network and data, by comparing the net deviation curves for these methods for different percentage of pruning. This parameter could be an alternative to ‘test accuracy’ that is normally used. Net Deviation is calculated while pruning, using the same data that was used for pruning. The detailed proofs of the stated theorems are given in the Appendix.

2 Related Works

Research in the area of architecture selection has led to different pruning approaches. Recently, obtaining a desired network architecture received significant attention from various researchers [10, 14, 17] and [11]. Network pruning was found to be a viable and popular alternative to optimize an architecture. This research can be dichotomised into two categories, with and without retraining.

2.1 Without Retraining

The weights which contribute maximum to the output must not be disturbed, if no retraining is required. In order to achieve this, [7] have used the idea of Core-sets that could be found through SVD or Structured Sparse PCA or

an activation-based procedure. Even when this method could provide a high compression, the matrix decomposition complexity involved could be higher for larger networks.

2.2 With Retraining

Most of the research works focus on methods that have a retraining or fine-tuning step after pruning. Such methods can be classified again as given below:

As an Optimization Procedure. In this category, retraining is defined as a procedure over the trained model to find the best sparse approximation of the trained model, that doesn't reduce the overall accuracy of the network. [4] does the same in two steps- one to learn the weights that can approximate a sparser matrix (L step) and another to compress the sparser matrix again (C step). [1] does model compression by considering model compression as a convex optimization algorithm. Minor fine tuning is also done at the end of this retraining procedure.

Without Any Optimization Procedure. A pruning scheme without any optimization procedure, delves into two things: either to keep the prominent nodes or to remove redundant nodes using some relevant criteria. Prominent nodes are defined as the nodes that contribute the most to the output layer nodes. These nodes could be defined based on the weight connections or gradients, as seen in [3, 8, 9, 13] and [6]. Redundant nodes can be removed by clustering nodes that give similar output as done in [14]. Data-free methods also exist, like [15] that does not use the data to calculate the pruning measure.

3 Analysis of Retraining Step of a Sparse Neural Network

3.1 Preliminaries

Consider a Multi-Layer Perceptron with 'L' layers, with $n^{(l)}$ nodes in layer l . Corresponding weights and biases are denoted as $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(L-1)}$ and $\mathbf{B}^{(2)}, \mathbf{B}^{(3)}, \dots, \mathbf{B}^{(L)}$ respectively, where $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$ and $\mathbf{B}^{(l)} \in \mathbb{R}^{n^{(l)} \times 1}$. The loss function is denoted as $L(\mathbf{W})$. This could be mean-squared error or cross-entropy loss, defined on both the weights and biases, using the labels and the predicted outputs.

3.2 Error Propagation in Sparse Neural Network

Pruning process can be made parallel if the same is done layer wise. For the same cause, the layer wise error bound, with respect to the overall allowed error needs to be known. This section hence looks into the individual contributions

of the change in the parameter matrices in each layer to the final output error. The output of the neural network is given as

$$\mathbf{Y}^{(L)} = f(\mathbf{W}^{(L-1)T} \mathbf{Y}^{(L-1)} + \mathbf{B}^{(L)}) \quad (1)$$

The deviation introduced in the output error $\delta\mathbf{Y}^{(L)}$, due to pruning the parameter matrices $\mathbf{W}^{(L-1)}$ by $\delta\mathbf{W}^{(L-1)}$, can be bounded as

$$\|\delta\mathbf{Y}^{(L)}\| \leq \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}} \delta\mathbf{W}^{(L-1)} \right\| + \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}} \delta\mathbf{Y}^{(L-1)} \right\| \quad (2)$$

Theorem 1. *Assuming that the input layer is left untouched, the output error introduced by pruning the trained network $N(\{\mathbf{W}_l\}_{l=1}^L, \mathbf{X})$ will always be upper bounded by the following relation,*

$$\|\delta\mathbf{Y}^{(L)}\| \leq \sum_{l=2}^L \left[\prod_{\substack{i=l+1 \\ (i \neq L)}}^L \left\| \frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial\mathbf{Y}^{(l)}}{\partial\mathbf{W}^{(l-1)}} \right\| \|\delta\mathbf{W}^{(l-1)}\| \quad (3)$$

The above relation essentially explains the accumulation of the error in each layer to produce the error in the final layer i.e., if ϵ is the total allowed error in the final layer, then it can be bounded by the sum of individual layer errors, ϵ_l ($l = 2, 3, \dots, L$) as shown below:

$$\epsilon \leq \epsilon_2 + \epsilon_3 + \dots + \epsilon_L \quad (4)$$

The above equation sets apart error bounds on different layers and will be of much help in optimisation-based pruning techniques.

$$\epsilon_L = \left\| \frac{\partial\mathbf{Y}_L}{\partial\mathbf{W}_L} \delta\mathbf{W}_L \right\|$$

$$\text{and } \epsilon_l = \left[\prod_{i=l+1}^L \left\| \frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial\mathbf{Y}^{(l)}}{\partial\mathbf{W}^{(l-1)}} \right\| \|\delta\mathbf{W}^{(l-1)}\|, \text{ for } l \neq L$$

The assumption of $\epsilon_l = 0$ results in a simple relation given in Eq. (5), which can help in explaining two design practices used in classification networks:

$$\delta\mathbf{W}^{(l-1)T} \mathbf{Y}^{(l-1)} = \mathbf{0} \quad (5)$$

1. *An optimised structure of Multi-Layer Perceptrons used for classification will have $n^{(l)} \geq n^{(l+1)}$, where $n^{(l)}$ denotes the number of nodes in layer l and $l = 2, 3, \dots, L$.*

Since $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$ and $\mathbf{Y}^{(l)} \in \mathbb{R}^{n^{(l)} \times 1}$, for Eq. (5) to have a solution, $n^{(l)} \geq n^{(l+1)}$. Thus the minimum number of nodes the hidden layers can have equally to help the network train well from the data, is the number of nodes in the output layer.

2. Data dependent approaches results in better compression models.

Each neural network is unique because of its architecture and the data it was trained on. Any pruning approach must not change the behaviour of the network with respect to the application it was destined to perform. Consider $\mathbf{Y}^{(l)} \in \mathbb{R}^{n^{(l)} \times B}$, where B is the batch size and $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$. For Eq. (5) to be satisfied, the column space of $\mathbf{Y}^{(l)}$ must lie in the null space of $\delta\mathbf{W}^{(l)}$ and vice-versa. This implies that the entries for the appropriate $\delta\mathbf{W}^{(l)}$ can be obtained when the pruning measure is coined based on the features obtained from that layer.

3.3 Net Deviation(D)

Different pruning algorithms are currently present for model compression. A measure, similar to test accuracy, for comparing different pruning approaches based on the compression ratios, is the normalized difference between the obtained error difference and the bound, which is defined as Net Deviation, given in (6). An example explaining the use of D, is given in Sect. 4.1.

$$D = \|\delta\mathbf{Y}^{(L)}\| - \sum_{l=2}^L \left[\prod_{\substack{i=l+1 \\ (i \neq L)}}^L \left\| \frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial \mathbf{Y}^{(l)}}{\partial \mathbf{W}^{(l-1)}} \right\| \|\delta\mathbf{W}^{(l-1)}\| \quad (6)$$

3.4 Theoretical Bounds on the Number of Epochs for Retraining a Sparse Neural Network

Assume that the loss function is continuously differentiable and strictly convex. The losses decide the number of epochs the network takes to reach convergence and hence, the number of epochs to reach convergence can be viewed to be directly proportional to the loss. If the total number of parameters in the network is M , the parameters can be made p -sparse in $\frac{M!}{p!(M-p)!}$ number of ways. Hence, the bounds provided below must be understood in the average sense. Adding a regularisation term still keeps the loss function strongly convex, if the initial loss function is strongly convex. This makes Theorem 2 and all the accompanying relations valid even for loss functions with regularisation terms.

Theorem 2. *Given a trained network $N(\{\mathbf{W}_l\}_{l=1}^L, \mathbf{X})$, trained from initial weights $\mathbf{W}_{initial}$ using $t_{initial}$ epochs. For fine-tuning the sparse network $N_{sparse}(\{\mathbf{W}_l\}_{l=1}^L, \mathbf{X})$, there exists a positive integer γ that lower bounds the number of epochs (t_{sparse}) to attain the original performance as,*

$$t_{sparse} \geq \frac{\gamma\mu_1}{\mu_2} \left[\frac{\|\nabla L(\mathbf{W}_{initial})\|^2}{\|\nabla L(\mathbf{W}_{sparse})\|^2} \right] t_{initial} \quad (7)$$

When there are different hidden layers, the gradients would follow the chain rule and the following equation can be incorporated in (7) to obtain the bound.

$$\|\nabla L(\mathbf{W})\|^2 = \|\nabla L(\mathbf{W}^{(1)})\|^2 + \|\nabla L(\mathbf{W}^{(2)})\|^2 + \dots + \|\nabla L(\mathbf{W}^{(L-1)})\|^2 \quad (8)$$

The definitions for μ_1 and μ_2 vary for connection and node pruning and are given below. The equations are written for pruning the trained model with final parameter matrix \mathbf{W}^* . \mathbf{W}' could be either the initial or sparse matrix.

Connection Pruning: Taking advantage of the fact that connection pruning results in a sparse parameter matrix of the same size as that of the unpruned network, μ can be defined as:

$$\mu' \leq \frac{\|\nabla L(\mathbf{W}') - \nabla L(\mathbf{W}^*)\|}{\|\mathbf{W}' - \mathbf{W}^*\|} \quad (9)$$

Node and Filter Pruning: Node and filter pruning reduces the rank of the parameter matrix and hence Eq. (9) cannot be used. PL inequality is used instead.

$$\mu' \leq \frac{\|\nabla L(\mathbf{W}')\|^2}{\|L(\mathbf{W}') - L(\mathbf{W}^*)\|} \quad (10)$$

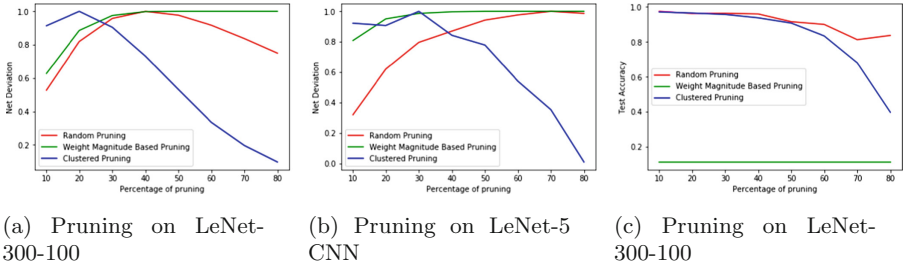


Fig. 1. Comparison of the three pruning methods for different percentage of pruning using Net Deviation ((a) and (b)) and Test Accuracy in (c). (a) and (c) are results on LeNet-300-100 and (b) are of LeNet-5

4 Experimental Results and Discussion

Simulations validating the theorems stated were performed on two popular networks: LeNet-300-100 and LeNet-5 [12], both trained on MNIST digit data set and had test accuracies of 97.77% and 97.65% respectively.

Table 1. The number of epochs taken for fine-tuning LeNet-300-100 pruned at different pruning and sparsity ratios.

Node pruning			Connection pruning		
Pruning ratio	Theoretical lower bound	Average epochs taken	Sparsity ratio	Theoretical lower bound	Average epochs taken
0.1	0.23	1.7	0.1	0.24	2.2
0.5	2.6	2.6	0.6	1.48	2.4
0.9	19.71	50	0.9	2.26	2.8

Table 2. The number of epochs taken for fine-tuning LeNet-5 pruned at different pruning and sparsity ratios.

Node pruning			Connection pruning		
Pruning ratio	Theoretical lower bound	Average epochs taken	Sparsity ratio	Theoretical lower bound	Average epochs taken
0.1	0.52	2	0.1	1.26	2.3
0.5	5.049	13	0.6	2.783	2.8
0.9	18.623	25.6	0.9	1.62	2.9

4.1 Analysis of Net Deviation

To explain the application of the parameter ‘Net Deviation’, LeNet-300-100 and LeNet-5 were pruned using Random, Weight magnitude based and Clustered Pruning approaches. In random pruning, the connections were made sparse randomly, while in clustered pruning, the features of each layer were clustered to the required pruning level. One out of each node or filter in the cluster is kept. The second method chose the nodes that had higher weight magnitude connections. The results for different percentage of pruning in an average sense, are given in Fig. 1, which explains that, for lower level of pruning, D is lower for random pruning approach. But for higher pruning or higher model compression, random pruning is not a good pruning method to look into. Net deviation is calculated using the same batch of data that was used for pruning. A similar comparison has been done on LeNet-300-100 using test accuracy as the parameter and the results are shown in Fig. 1(c). It could be seen that when choosing the appropriate method for pruning, for a particular data set and network, test accuracy does not give much information with respect to the amount of compression or the percentage of pruning. Because of similar inferences, random pruning could be applied by a user, who wants smaller compression, with lower computational complexity as random pruning is computationally less expensive than clustered pruning.

4.2 Theoretical Bounds on the Number of Epochs for Retraining a Sparse Neural Network

Both the networks were pruned randomly with the same seed in two ways: Connection Pruning and Node Pruning. Tables 1 and 2 show the results obtained at different pruning ratios for LeNet-300-100 and LeNet-5 respectively. For LeNet-300-100 pruned at pruning ratios 0.1, 0.5 and 0.9 and for sparsity ratios of 0.1, 0.6 and 0.9, the value of γ was obtained as 0.01, 1 and 100 and 1e-4, 0.2 and 0.5, respectively. Similarly for LeNet-5, γ was found to be 1e-5, 2e-4 and 5e-5 for pruning ratios 0.1, 0.5 and 0.9 and 3e-5, 1e-6 and 1e-5 for sparsity ratios 0.1, 0.6 and 0.9 respectively. The results validate the bound provided in Theorem 2.

5 Conclusions

This paper has theoretically derived and experimentally validated the amount of retraining that would be required after pruning, in terms of the relative number of epochs. Also, the propagation of errors through the layers, due to pruning different layers is analysed and a bound to the amount of error that the layers contribute was derived. The parameter ‘Net Deviation’ can be used to study different pruning approaches and hence can be used as a criteria for ranking different pruning approaches. If not completely avoided, reducing the number of epochs linked to retraining the network will reduce the computational complexity involved with training a Neural Network. An empirical formula to calculate the γ parameter that bounds the number of epochs required for retraining, is considered as a future work.

A Appendix

A.1 Proof of Theorem 1

From [16], for a function with more than one variable $q(x, y)$, when the uncertainties in x and y are independent and random, the uncertainty in q can be written as

$$\delta q^2 = \left[\frac{\partial q}{\partial x} \delta x \right]^2 + \left[\frac{\partial q}{\partial y} \delta y \right]^2 \quad (11)$$

Applying triangular inequality, the following equation is valid.

$$\delta q \leq \left| \frac{\partial q}{\partial x} \right| \delta x + \left| \frac{\partial q}{\partial y} \right| \delta y \quad (12)$$

In the concept of pruning of neural networks, the weight changes and output changes in earlier layers are independent to each other. The output of the neural network is

$$\mathbf{Y}^{(L)} = f(\mathbf{W}^{(L-1)T} \mathbf{Y}^{(L-1)} + \mathbf{B}^{(L)}) \quad (13)$$

The function $f(\cdot)$ can be sigmoid or relu or softmax function as per the layer considered. Usually, the softmax function is used in the output layer. Assuming no change in the bias of the layers,

$$\|\delta\mathbf{Y}^{(L)}\|^2 = \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)} \right\|^2 + \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}}\delta\mathbf{Y}^{(L-1)} \right\|^2 \quad (14)$$

and

$$\|\delta\mathbf{Y}^{(L)}\| \leq \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)} \right\| + \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}}\delta\mathbf{Y}^{(L-1)} \right\| \quad (15)$$

Accumulating the effect of weight changes in all the layers and assuming no change in the input layer, we get

$$\begin{aligned} \|\delta\mathbf{Y}^{(L)}\| \leq & \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)} \right\| + \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}} \frac{\partial\mathbf{Y}^{(L-1)}}{\partial\mathbf{W}^{(L-2)}}\delta\mathbf{W}^{(L-2)} \right\| + \dots \\ & + \left[\prod_{i=3}^L \left\| \frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial\mathbf{Y}^2}{\partial\mathbf{W}^1}\delta\mathbf{W}^1 \right\| \end{aligned}$$

This is given in Theorem 1.

$$\|\delta\mathbf{Y}^{(L)}\| \leq \sum_{l=2}^L \left[\prod_{\substack{i=l+1 \\ (l \neq L)}}^L \left\| \frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial\mathbf{Y}^{(l)}}{\partial\mathbf{W}^{(l-1)}} \right\| \|\delta\mathbf{W}^{(l-1)}\| \quad (16)$$

Hence the proof. Suppose the output error is bounded by ϵ , which corresponds to the LHS in (16) given above. Expanding the RHS,

$$\begin{aligned} \epsilon_1 &= 0 \\ \epsilon_2 &= \left[\prod_{i=3}^L \left\| \frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}} \right\| \right] \left\| \frac{\partial\mathbf{Y}^2}{\partial\mathbf{W}^1}\delta\mathbf{W}^1 \right\| \\ \epsilon_L &= \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)} \right\| \end{aligned}$$

Hence, the result given in (4) is obtained, which could also be seen in [2].

$$\epsilon \leq \epsilon_1 + \epsilon_2 + \dots + \epsilon_L \quad (17)$$

An alternative expression can be found in a similar pattern of derivation starting with (14)

$$\epsilon^2 = \epsilon_1^2 + \epsilon_2^2 + \dots + \epsilon_L^2 \quad (18)$$

The proof of (5) is quite obvious. Assuming that the error in each layer is approximately zero, $\epsilon_L = 0$ and that the function $f(\cdot)$ is sigmoid,

$$\begin{aligned} \left\| \frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)} \right\| &= 0 \\ \|\mathbf{Y}^{(L)} \odot (\mathbf{1} - \mathbf{Y}^{(L)})\| \|\delta\mathbf{W}^{(L-1)T} \mathbf{Y}^{(L-1)}\| &= 0 \\ \delta\mathbf{W}^{(L-1)T} \mathbf{Y}^{(L-1)} &= \mathbf{0} \end{aligned}$$

\odot refers to the Hadamard product of matrices. Similar results can be obtained for other layers as well.

A.2 Proof of Theorem 2

Assume that the loss function is continuously differentiable and strictly convex. By the strong convexity assumption, the Polyak-Lojasiewicz (PL) inequality can be implied, which is stated as follows:

For a continuously differentiable and strongly convex function f , over parameter x , with minimum point x^* ,

$$\frac{1}{2} \|\nabla f(x)\|^2 \geq \mu(f(x) - f(x^*)), \forall x \quad (19)$$

Using the above relation on the loss function $L(W)$,

1. Initial training of the network, till convergence

$$\frac{1}{2} \|\nabla L(W_{initial})\|^2 \geq \mu_1(L(W_{initial}) - L(W^*)) \quad (20)$$

2. Fine-tuning the sparse network, to reach back to the original performance

$$\frac{1}{2} \|\nabla L(W_{sparse})\|^2 \geq \mu_2(L(W_{sparse}) - L(W^*)) \quad (21)$$

The number of epochs to reach convergence is directly proportional to the difference in losses and can be written as:

$$t_{initial} \propto \|L(W_{initial}) - L(W^*)\| \quad (22)$$

$$t_{sparse} \propto \|L(W_{sparse}) - L(W^*)\| \quad (23)$$

Combining (20–23) given above, and with a γ to accommodate the proportionality in (22) and (23),

$$t_{sparse} \geq \frac{\gamma \mu_1}{\mu_2} \left[\frac{\|\nabla L(W_{initial})\|^2}{\|\nabla L(W_{sparse})\|^2} \right] t_{initial} \quad (24)$$

References

1. Aghasi, A., Abdi, A., Nguyen, N., Romberg, J.: Net-trim: convex pruning of deep neural networks with performance guarantee. In: Advances in Neural Information Processing Systems, pp. 3177–3186 (2017)
2. Aghasi, A., Abdi, A., Romberg, J.: Fast convex pruning of deep neural networks. arXiv preprint [arXiv:1806.06457](https://arxiv.org/abs/1806.06457) (2018)
3. Augasta, M.G., Kathirvalavakumar, T.: A novel pruning algorithm for optimizing feedforward neural network of classification problems. Neural Process. Lett. **34**(3), 241 (2011)

4. Carreira-Perpinán, M.A., Idelbayev, Y.: Learning-compression algorithms for neural net pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8532–8541 (2018)
5. Castellano, G., Fanelli, A.M., Pelillo, M.: An iterative pruning algorithm for feed-forward neural networks. *IEEE Trans. Neural Netw.* **8**(3), 519–531 (1997)
6. Dong, X., Chen, S., Pan, S.: Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 4857–4867 (2017)
7. Dubey, A., Chatterjee, M., Ahuja, N.: Coreset-based neural network compression. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 454–470 (2018)
8. Engelbrecht, A.P.: A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Trans. Neural Netw.* **12**(6), 1386–1399 (2001)
9. Hagiwara, M.: Removal of hidden units and weights for back propagation networks. In: Proceedings of 1993 International Conference on Neural Networks (IJCNN-1993), Nagoya, Japan, vol. 1, pp. 351–354. IEEE (1993)
10. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint [arXiv:1510.00149](https://arxiv.org/abs/1510.00149) (2015)
11. Huynh, L.N., Lee, Y., Balan, R.K.: D-pruner: filter-based pruning method for deep convolutional neural network. In: Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning, pp. 7–12. ACM (2018)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
13. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)
14. Li, L., Xu, Y., Zhu, J.: Filter level pruning based on similar feature extraction for convolutional neural networks. *IEICE Trans. Inf. Syst.* **101**(4), 1203–1206 (2018)
15. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. arXiv preprint [arXiv:1507.06149](https://arxiv.org/abs/1507.06149) (2015)
16. Taylor, J.R.: Error Analysis. University Science Books, Sausalito (1997)
17. Tung, F., Mori, G.: Clip-q: deep network compression learning by in-parallel pruning-quantization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7873–7882 (2018)
18. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint [arXiv:1710.01878](https://arxiv.org/abs/1710.01878) (2017)