Compressing Deep Learning Models

Nikhil Kumar Jha, Computer Engineering Group, University of Paderborn

Abstract- **Neural networks have recently achieved huge success in many deep learning tasks. However, existing deep models suffer from the problem of over-parametrization and are thus computationally expensive as well as memory intensive. This causes a big hindrance in their deployment on small embedded systems with low memory resources and strict latency requirements. There is a need to perform model compression and thus accelerate the model, at the same time making sure that it doesn't take a significant toll on the performance. By doing so, we will achieve a neural network structure which will speed up inference as well as training. In the past few years, many findings and results have been published in this domain. The aim of this paper is to provide an idea of existing pruning techniques to optimize the neural networks in order to achieve the aforementioned goals. For each technique, some insightful analysis, advantages, and limitations have been discussed.**

1 Introduction

Neural networks are famous for their considerably good learning capacity and in turn offering high classification accuracy. But they suffer some drawbacks like long training time and a huge number of parameters. The solution to these is to optimize the network architecture, the learning rate, the number of training epochs, and also the size of the data determines the capability of the network to generalize. Such an optimized architecture guarantees fewer computations and good generalization capability.

A large number of papers have been published and many pruning algorithms have been introduced in this research area. In these papers, different neural network structures like feedforward neural networks (FFNN), recurrent neural networks (RNN), convolution neural networks (CNN), etc. have been proposed and applied to different types of real-world datasets. Each network has its own set of advantages and disadvantages. The structural flexibility and good representation capabilities, make feedforward neural networks one of the most popular architectures. It basically consists of an input layer, one or more hidden layers, and an output layer. The number of the features and targets in the dataset determine the number of nodes in the input layer and the output layer respectively. The number of nodes in all the hidden layers combined will depend on the complexity of the problem.

An FFNN with large complexity will learn fast, but due to a large number of weights and layers in such a network, the computation cost and storage needed to store all these variables are too much. Also, such a structure will face the problem of overfitting the training data and thus generalize the test data poorly. A better approach is to use a less complex and small structure. It is easier to interpret a smaller network and extract out the knowledge by defining simple rules. Also, looking at it from the viewpoint of computation cost and resources needed, a smaller network will require very limited resources. However training such a network will not be easy and will take a lot of effort, since it will have lesser processing elements and hence might not have enough learning capacity. So it can be seen that both large and small networks have their fair share of advantages and disadvantages.

A good architecture is thus a network small enough to generalize the test data properly and large enough to learn the problem well. This paper has been organized as follows: In Section 2, the various approaches of network pruning have been discussed. Section 3 dives into surveying the existing pruning methods discussing its advantages. Section 4 provides a comparison of the various existing such techniques. This comparison is based on the percentage of pruning done, pruning speed, and classification accuracy by implementing these methods on different real datasets.

2 Related work

Trimming down the weights in the network architecture is defined as pruning. Which weights to be pruned off, is decided by calculating the sensitivity of the total error to the removal of each weight in the network. Weights which are not so relevant and exclusion of which will not make a difference in the performance can be eliminated at every iteration step during the training phase. Thus, pruning will result in a smaller network size which will have a better accuracy and more generalization power, owing to the removal of the redundant connections in the network.

There are various algorithms available in the literature for pruning off the redundant connections. One trivial approach is to use brute-force [1], to find the most optimal neural network architecture. The idea is to train the various smaller networks successively and find out which of the smaller networks performs the best. This approach is quite time-consuming. It requires the training of the smaller yet large number of models. Convergence with these models is still not guaranteed [2].

One other approach [3] is to define weight sets or vectors in a neural network and identify if the two weight sets are equal. If they are equal, then one of them can be easily removed and a surgery step is performed to compensate for the removal of the weight set. This approach is based on the *Hebbian* principle that "neurons that wire together, fire together", and is able to showcase a different form of redundancy in a neural network. Not only the single weights being zero or close to zero contribute to redundancy, but also the equal weight sets have a fair contribution. But a network might not always have two weight sets that exactly alike. In such cases, saliency values can be calculated to find out the pair which if removed will cause the least change in the output activation.

Other approaches basically involve growing the network, pruning off the unimportant weights, and a combination of the two methods called 'growing and pruning' [4]. The first method is one of the constructive methods, which starts with a very minimal network structure and during the training phase, it keeps on adding new hidden layer nodes. One of the major problems with this approach is that the small model might get stuck in local minima and be causing the training time to increase significantly. The other class is of destructive methods, which starts with a large network and during training, the redundant weights and nodes are removed. This requires a clear definition of the upper bound on the size of the initial large network. However, it has been proved that the comparison of the overall time to train a small network versus the time to train a large network and then prune it off, is quite similar. The other variation of the above is a hybrid method, which starts with training a small network, and incrementally adds hidden nodes when the network is not able to reduce the training error anymore.

Another technique is called dynamic network surgery [5], which performs on-the-fly connection pruning and is able to remarkably reduce the network complexity, in a non-greedy way. To compensate for the losses due to over pruning, splicing is done to enable connection recovery if at some point the pruned connection is found to be important. This method deals with the problem of incorrect pruning. But the challenge is to accurately identify the importance of a connection due to mutually connected neurons. A solution to this is to continue the learning process while maintaining the network structure.

The last technique to be discussed here is the runtime neural pruning [6], where the idea is to prune the neural network at runtime. Pruning operation is modeled as a Markov decision process and reinforcement learning is used for training. The importance of a convolution kernel is judged by the agent, which performs channel wise pruning. This method has one major advantage that the full capability of the network is preserved and the pruning is performed based on the information present in the input data and the feature maps adaptively. Also, the balance point is adjustable based on the resources at hand.

3 Classifying the different methods

The literature suggests a number of pruning techniques to obtain the optimized neural network architecture. These techniques, based on how they perform the pruning operation, can be classified into different categories discussed below.

3.1 Mutual Information based methods

This method employs singular value decomposition to determine the covariance matrix for the activation function of the hidden units of the network. The rank of this matrix will reveal the ideal number of hidden units that should be present. Xing and Hu [7] proposed a technique to prune the input as well as the hidden units of the multilayer perceptrons in a two-phase construction approach, based on the covariance matrix. The saliency of the input units is firstly calculated and ranked according to their performance contributions, removing the irrelevant ones from the network. This is followed by removal of non-critical hidden units in a similar way.
The work described by Qiao and Zhang [8] elaborates on a technique, where the covariance matrix for the connections in the neural network gives an idea of its complexity, which is used as a base for node pruning. The algorithm does not need to train the cost function to a local minimum, and hence the network pruning can be done online.

3.2 Weight decay method

The idea is to force the smaller weights to zero, by adding a penalty term to the objective function. Setiono's method [9] prevents the weights to take large values and suppresses irrelevant connections, by using a penalty function. The network is trained to minimize the penalty function and thus removing redundant weights. But there is a drawback that the network might settle for a local minimum during training, and might remove the weights which might actually be important for generalization.
Mabu et al. [10] published two approaches, one is to determine the importance of a node using the Gauss Schmidt algorithm in an epoch, update the weights connected to this most important node and leave others untouched. The other approach is to attach a penalty term with the error function. In the weight update rule, the derivatives of not just the current weight but all other weights are considered. This way, while minimizing the error and weight update, effect of all other weights is also considered.

3.3 Magnitude based methods

This is one of the most naive approaches in pruning, where the objective is to remove the smaller weights from the network, which are considered to be irrelevant to the network's performance. This is sometimes called the greedy approach. Castellano et al. [2] proposed a method to remove the unimportant weights whose magnitude were below a certain threshold and retrain the sparse network to fine tune the remaining weights. Retraining is the most important aspect of this technique otherwise the performance takes a dip.
Hagiwara's [10] technique uses strategies like Goodness factor to identify redundant weights and neurons, based on the magnitude. Such simple methods require fewer computations, however, the major drawback is pruning of the important connections as well [11].

3.4 Cross-validation methods

The pruning is still based on the magnitude as in the previous method, but here the effect of pruning is tested simultaneously by adding a validation step. The data is split into training and validation data sets, and after each phase of pruning, the validation score is checked. Only if it outperforms the original network, the pruned one is considered and the process continues until the performance starts to dip. Setiono and Huynh [12] were the first ones to publish this. They claim that this additional step is favorable for a better generalization performance.

3.5 Sensitivity analysis based methods

The idea is to determine the contribution of each weight in a network and prune off the ones which affect the objective function the least. Most popular in this category are two methods - Optimal Brain Damage (OBD) [13] and Optimal Brain Surgeon (OBS) [14]. Lecun et al. proposed the OBD technique which estimates the second order derivative of the error function with respect to a weight and is called saliency of that weight. The assumption taken is that the error function is of the second degree and that the Hessian matrix is diagonal. Weights with low saliency will be removed, and training is continued. Based on the same principle, Hassibi et al. proposed OBS method but without the assumption that the matrix is diagonal, since if the assumption had been wrong, it would lead to incorrect pruning. The performance is recorded on the validation set and early stopping is used to end the training when the error starts to increase. However, there is no guarantee that the optimal point is achieved by the learning curve through this procedure.

Engelbrecht's Variance Nullity Pruning (VNP) [15] approach was to use a new measure called variance nullity, which is based on the average sensitivity over all the network parameters, to remove redundant weights and nodes. Ponnapelli et al. [16] suggested to compare the sensitivities of weights with only those connected to the same node, and coined the term 'Local Relative Sensitivity Index'. Its the ratio of the sensitivity of a weight to the combined sensitivities of all the weights related to that node from the previous layer. However, this method only considers weight removal and not node pruning.

3.6 Significance based methods

This method uses a significance measure based on the input as well as the output of the network, and only the nodes with significance above a threshold are kept and others are pruned. Belue et al. [17] proposed to inject into the network a noisy parameter and then determine if statistically, the significance of the parameters is greater than that of the noisy one. If not, those parameters are pruned. Augasta et al. [18] proposed the N2PS algorithm which used a new significant measure based on the sigmoid activation value of the node and connected weights, to remove the non-critical input and hidden nodes, in order to get the most optimal architecture.

3.7 Summarizing advantages and disadvantages

Each of the methods discussed above has their own set of advantages and disadvantages. While some methods are able to prune both the hidden as well as the input neurons, other algorithms prune only the non-critical hidden neurons. One of the major drawbacks common to all the algorithms is their efficiency. For example, the OBD and OBS are highly computationally inefficient. Magnitude based pruning has a serious drawback of removing the small but important weights since it follows the simple greedy approach. Some methods require manually setting the threshold level for decision making if the pruning should be done or not. Going for more refined algorithms will surely give better performance, will result in escalating the computation time. Sensitivity based methods cannot be used when inputs are interdependent, instead Mutual Information and significance based methods should be used.

4 Analysis

Augusta et al. [19] performed the analysis of the different algorithms on four real-world datasets which are discussed further in the subsections, and also their performances were evaluated. In this section, algorithms like N2PS, VNP, Hu-Xing's method, magnitude based pruning, OBS, and OBD are compared with each other.

4.1 The datasets

The following datasets were used for the evaluation.

Iris Flowers dataset: The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant, namely Setosa, Versicolour, Virginica. The attributes are the lengths and widths of the sepal and petal.

Wisconsin Breast Cancer dataset: This contains the data to classify between the benign and malignant breast tumors. It has 699 observations with 9 attributes as input and 2 classes as output.

Hepatitis Domain dataset: This dataset classifies the hepatitis case as die or live. It has 155 observations with 19 attributes.

Pima Indians Diabetes dataset: The dataset classifies diabetes in a patient as negative or positive with values 1 and 2 respectively. It has 768 observations with 8 attributes for each.

Table 1: Properties of the 4 datasets

Properties	iris	cancer	hepatitis	diabetes
Number of classes	3	2	2	2
Number of observations	150	699	155	768
Number of training samples	75	350	81	384
Number of test samples	75	349	74	384
Number of attributes	4	9	19	8

Table 1 [19] describes the dataset information in more detail. The algorithms also need some hyper-parameters other than just the batch size and number of epochs. N2PS uses momentum μ 0.5 and the learning rate η 0.1 for each dataset. Training was done for 200 iterations. The other 5 algorithms use momentum μ 0.5 and learning rate η 0.1, and train the network until the mean square error is at least 0.01 for each dataset. For all the experiments, same architecture has been used which has an input and an output layer, and for simplicity it has only one hidden layer. The weight pruning algorithms (magnitude based, OBD, OBS) were compared with node pruning algorithms (VNP, N2PS, Hu-Xing's). Comparison of the different methods was done on metrics like classification accuracy, number of nodes pruned, the number of iterations required for training and the generalization ability.

4.2 Detailed comparison

Magnitude based pruning methods are quite inefficient since they only consider the weight's magnitude while pruning. The Hessian matrix of the complete network is needed to be calculated for the OBS and OBD methods, which is an overhead. The three methods only prune the hidden non-critical nodes. VNP method does the same for both the input as well hidden layer units to get a satisfactory result, but since it doesn't take into account the mutual dependency of input nodes and output of hidden nodes, it might not detect all the irrelevant nodes. This limitation is taken care of by Hu-Xing's method which considers the dependency between the nodes, but the pruning is performed in two separate phases. N2PS combines the best of the Hu-Xing's and VNP, by performing the input and hidden unit pruning using a single equation as VNP, and like Hu-Xing's it considers the dependencies as well.

Table 2 [19] shows the results of the classification accuracy for the 4 datasets. The main metric used for comparison was the classification accuracy, where all tests were run 10 times and their average was taken. Only if the dip in classification accuracy was less than 5%, the pruned model was considered. N2PS [19] performed better, with same or better accuracy and minimum number of nodes than other architectures. Reason being, it didn't need to perform any complex calculation of the significance measure for each node.

Table 2: Comparing the classification accuracy of the 6 pruning methods on all 4 datasets

Datasets	Non pruned	N2PS	VNP	Hu-Xing	OBD	OBS	MBP
iris	96%	98.67%	97.7%	98.67%	98%	98%	98%
cancer	95.4%	97.1%	97.8%	96.78%	92.5%	90%	94.2%
hepatitis	80.2%	86.4%	83.3%	84.62%	78.7%	73.8%	80.3%
diabetes	68.6%	70.3%	69.1%	74.22%	68.6%	65.4%	68.9%

The performance of the different algorithms in terms of classification accuracy is represented in Figure 1 [19]. For all the datasets, N2PS performs better than the algorithms which perform pruning on the hidden neurons only, and for 3 out of 4 datasets, N2PS performs better or equal to the Hu-Xing's and VNP.

Figure 2 [19] compares the number of hidden neurons removed from the network for the different methods and for all the datasets. This number is determined by the difference between the number of hidden neurons in the original network and the pruned network.

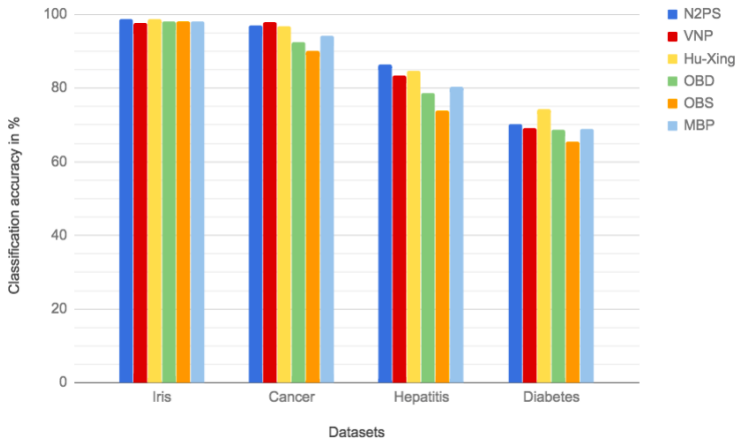

Figure 1: Comparing classification accuracies of pruning algorithms

From the plot, the observation is that VNP and N2PS prune almost equal number of neurons for all the datasets. Due to the fact that N2PS considers dependency between inputs of the network and the hidden node's output, it is seen to perform better than others.

The time complexity for each of the algorithms was determined by the selection method used to identify the node to be pruned. Let p be the number of training samples, n be the number of connections present in the network and e be the number of training steps. The total time complexity, which means taking into account selection of pruning nodes and performing the actual pruning, for OBD and OBS is $O(epn^3)$ and that for magnitude based is $O(epn^2)$.

To summarise, this study indicated that the significance based methods outperform the other methods in terms of classification accuracy and the number of neurons pruned.

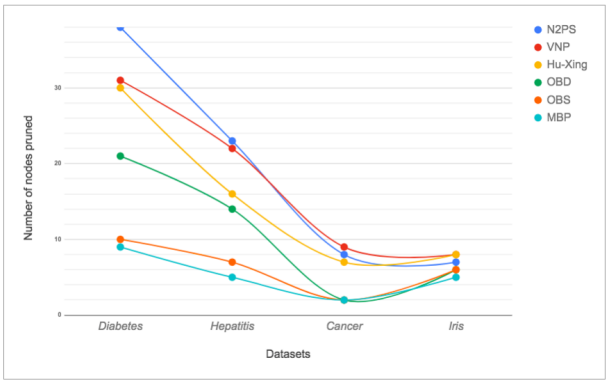

Figure 2: Comparison of the number of nodes pruned in the different pruning algorithms

5 Conclusion

This work summarizes the various pruning schemes available to compress neural network architectures. These algorithms are based on mutual information, sensitivity, magnitude, significance and sensitivity of the network. Their performances are compared using real-world datasets. The three methods OBD, OBS, and MBP, which depend mostly on the magnitude values, are only capable of pruning hidden units. However, sometimes they prune few important nodes as well and hence this result in poor generalization or a sub-optimal network.

The other non-trivial algorithms like N2PS, Xing-Hu's, and VNP can prune both the hidden as well as input nodes of the feedforward neural network. A detailed discussion on the performance is done in Section 4. Though these algorithms provide better results than the former ones, this gain is achieved with the increase in the computation time, since the number of iterations required is greater.

In a nutshell, algorithms based on sensitivity, significance and mutual induction, which use significance as a meta-heuristic for pruning as in N2PS and Xing-HU, are at present the most suitable methods available to optimize the feedforward neural network architectures.

References

[1] R. Reed, "Pruning algorithms-a survey," *IEEE transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[2] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[3] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[4] O. Aran, O. T. Yildiz, and E. Alpaydin, "An incremental framework based on cross-validation for estimating the architecture of a multilayer perceptron," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 02, pp. 159–190, 2009.

[5] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

[6] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, pp. 2181–2191, 2017.

[7] H.-J. Xing, B.-G. Hu, *et al.*, "Two-phase construction of multilayer perceptrons using information theory.," *IEEE Trans. Neural Networks*, vol. 20, no. 4, pp. 715–721, 2009.

[8] Z. Zhang and J. Qiao, "A node pruning algorithm for feedforward neural network based on neural complexity," in *Intelligent Control and Information Processing (ICICIP), 2010 International Conference on*, pp. 406–410, IEEE, 2010.

[9] R. Setiono, "A penalty-function approach for pruning feedforward neural networks," *Neural computation*, vol. 9, no. 1, pp. 185–204, 1997.

[10] W. Wan, S. Mabu, K. Shimada, K. Hirasawa, and J. Hu, "Enhancing the generalization ability of neural networks through controlling the hidden layers," *Applied Soft Computing*, vol. 9, no. 1, pp. 404–414, 2009.

[11] J. Sietsma and R. J. Dow, "Neural net pruning-why and how," in *IEEE international conference on neural networks*, vol. 1, pp. 325–333, IEEE San Diego, 1988.

[12] T. Q. Huynh and R. Setiono, "Effective neural network pruning using cross-validation," in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 2, pp. 972–977, IEEE, 2005.

[13] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[14] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Neural Networks, 1993., IEEE International Conference on*, pp. 293–299, IEEE, 1993.

[15] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE transactions on Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[16] P. Ponnapalli, K. Ho, and M. Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 964–968, 1999.

[17] L. M. Belue and K. W. Bauer Jr, "Determining input features for multilayer perceptrons," *Neurocomputing*, vol. 7, no. 2, pp. 111–121, 1995.

[18] M. G. Augasta and T. Kathirvalavakumar, "A novel pruning algorithm for optimizing feedforward neural network of classification problems," *Neural processing letters*, vol. 34, no. 3, p. 241, 2011.

[19] M. G. Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networks—a comparative study," *Central European Journal of Computer Science*, vol. 3, no. 3, pp. 105–115, 2013.