Interval Adjoint Significance Analysis for Neural Networks

Sher Afghan[0000−0003−2870−8038] and Uwe Naumann[0000−0002−7518−5922]

Software and Tools for Computational Engineering,
RWTH Aachen University, 52074, Aachen, Germany
{afghan,naumann}@stce.rwth-aachen.de

Abstract. Optimal neural network architecture is a very important factor for computational complexity and memory footprints of neural networks. In this regard, a robust pruning method based on interval adjoints significance analysis is presented in this paper to prune irrelevant and redundant nodes from a neural network. The significance of a node is defined as a product of a node's interval width and an absolute maximum of first-order derivative of that node's interval. Based on the significance of nodes, one can decide how much to prune from each layer. We show that the proposed method works effectively on hidden and input layers by experimenting on famous and complex datasets of machine learning. In the proposed method, a node is removed based on its significance and bias is updated for remaining nodes.

Keywords: Significance analysis · Sensitivity analysis · Neural network pruning · Interval adjoints.

1 Introduction

Neural networks and deep belief networks are powerful tools of machine learning for classification tasks. There are many things to consider for the construction of effective neural network architecture i.e., learning rate, optimization method, regularization, etc. But one of the most important hyper-parameter is network size. It is hard to guess the optimal size of a network. Large networks are good at memorization and get trained quickly but there is a lack of generalization in the large networks. We can end up in over-fitting our networks. We can solve this problem of generalization by constructing smaller networks and save the computational cost of classification but this approach can end up in under-fitting. Success is to come up with neural network architecture which can solve both problems [1].

Researchers have proposed different techniques such as; brute-force [2], growing [3] and pruning methods [4]. Out of these techniques, pruning results in effective compressed neural network architecture while not significantly hurting network accuracy. This technique starts with a well-trained network. Assuming the network is oversized, it tries to remove irrelevant or insignificant parameters

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

from the network. These parameters can be network's weights, inputs, or hidden units.

Over time multiple pruning methods have been proposed (see detailed surveys [1,2,5]). Among many methods, the sensitivity-based analysis technique is the most famous one [6,7,8,9,10]. It measures the impact of neural network parameters on the output. Our proposed method also utilizes the concept of sensitivity to define the significance of network parameters. Sensitivities of an objective function concerning weights of networks are used to optimize network's weights while sensitivities of output unit concerning input and hidden units are used to find the significance of the network's input and hidden units.

This paper presents a method for finding out the sensitivities of the network's output concerning the network's input and hidden units in a more robust and efficient way by using interval adjoints. Input and hidden unit values and their impact on output are used to define the significance of the input and hidden units. The significance analysis method defined in this paper takes care of all the information of the network units and the information stored during significance analysis is used to update the remaining parameters biased in the network.

The rest of the paper is organized as follows. Section 2 briefly describes algorithmic differentiation (AD) for interval data with examples. Section 3 presents our significance analysis method for pruning. Experimental results are given in section 4. The conclusion is given in section 5.

2 Interval Adjoint Algorithmic Differentiation

The brief introduction to AD [11,12] is given in this section along with the modes of AD and differentiation with one of the mode of AD commonly known as adoint mode of AD. Later, intervals for interval adjoint algorithmic differentiation are used.

2.1 Basics of AD

Let F be a differentiable implementation of a mathematical function $F : \mathbb{R}^{n+l} \longrightarrow \mathbb{R}^m : \mathbf{y} = (\mathbf{x}, \mathbf{p})$, computing an output vector $\mathbf{y} \in \mathbb{R}^m$ from inputs $\mathbf{x} \in \mathbb{R}^n$ and constant input parameter $\mathbf{p} \in \mathbb{R}^l$. Differentiating F with respect to $\mathbf{x}$ yields the Jacobian matrix $\nabla_{\mathbf{x}} F \in \mathbb{R}^{m*n}$ of F.

This mathematical function F can be transformed to coded form in some higher level programming language to apply AD on that code. AD works on the principle of the chain rule. It can be implemented using source transformation [13] or operator overloading [14] to change the domain of variables involved in the computation. It calculates the derivatives and different partials along with the each output (primal values) in a time similar to one evaluation of the function. There are multiple tools[1] available which can implement AD e.g. `dco/c++` [15,16]. `dco/c++` implements AD with the help of operator overloading and it has been successfully used for many applications [17,18].

[1] http://www.autodiff.org/?module=Tools

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

2.2 Modes of AD

There are many modes of AD [11,12] but two of them are most widely used; one is forward mode AD (also; tangent linear mode of AD) and second is reverse mode AD (also; adjoint mode of AD). Because, we are interested in adjoints for our research so we are going to describe reverse mode AD briefly.

There are two phases in reverse mode of AD; forward and backward pass. In forward pass, function code is run forward yielding primal values for all intermediate and output variables and storing all the relevant information which are needed during backward pass. During backward pass, adjoints of outputs (for outputs, adjoint is evidently 1) will be propagated backwards through the computation of the original model to adjoints of inputs.

Example (AD Reverse Mode) Below is an example of adjoint mode AD evaluation on function $f(\mathbf{x}) = sin(x_o \cdot x_1)$.

Forward Pass: With intermediate variables $v_i \in \mathbb{R}, i = 1, 2$, a possible code list of f is

$$\begin{aligned} v_1 &= x_1 \cdot x_2 \\ v_2 &= sin(v_1) \\ y &= v_2 \end{aligned}$$

Backward Pass: With intermediate variables $v_i \in \mathbb{R}, i = 1, 2$ and associated adjoint variables $v_{i(1)}$, a possible adjoint code of f is

$$\begin{aligned} v_{2(1)} &= y_{(1)} \\ v_{1(1)} &= cos(v_1) \cdot v_{2(1)} \\ x_{2(1)} &= x_1 v_{1(1)} \\ x_{1(1)} &= x_2 v_{1(1)} \end{aligned}$$

2.3 Interval Adjoints

Consider, lower case letters (e.g., $a, b, c, ...$) represents real numbers, uppercase letters (e.g., $A, B, C, ...$) represents interval data and an interval is represented by $X = [x^l, x^u]$, where l and u represents the lower and upper limit of the interval, respectively.

Interval Arithmetic (IA), evaluates a function $f[X]$ for the given range over a domain in a way that it gives guaranteed enclosure $f[X] \supseteq \{f[x]|x \ni [X]\}$ that contains all possible values of $f(x)$ for $x \ni [X]$. Similarly, interval evaluation yield enclosures $[V_i]$ for all intermediate variables V_i.

The adoint mode of AD can also be applied to interval functions for differentiation purpose [19]. The impact of individual input and intermediate variables on the output of an interval-valued function can easily be evaluated by using adjoint mode of AD over interval functions. AD not only computes the primal value of intermediate and output variables, it also computes their derivative with the help of chain rule. The first order derivatives $(\frac{\delta Y}{\delta X_i}, \frac{\delta Y}{\delta V_i})$ of output V with respect to all inputs X_i and intermediate variables V_i can be computed with the adjoint mode of AD in a single evaluation of function f. In the same way, IA and AD can be used to find out the interval-valued partial derivatives $(\nabla_{[V_i]}[Y]$,

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

$\nabla_{[X_i]}[Y]$) that contains all the possible derivatives of output Y with respect to intermediate variables V_i and input variables X_i over the given input interval X.

Example (AD Reverse Mode with Interval Data) Below is an example of adjoint mode AD evaluation on interval function $f(\mathbf{X}) = sin(X_o \,.\, X_1)$ for calculation of interval adjoints. Let $X = \{X_1, X_2\} \in \mathbb{R}^2$, where $X_1 = [0.5, 1.5], X_2 = [-0.5, 0.5]$

Interval output of $f(X)$:

$$f(X) = [-0.6816, 0.6816]$$

Differentiation:

$$\nabla_X f(X) = \begin{pmatrix} \frac{\delta f(X)}{\delta X_1} \\ \frac{\delta f(X)}{\delta X_2} \end{pmatrix} = \begin{pmatrix} cos(X_1 * X_2) * X_2 \\ cos(X_1 * X_2) * X_1 \end{pmatrix}$$

Interval evaluation of $\nabla_X f(X)$:

$$\nabla_X f(X) = \begin{pmatrix} cos(X_1 * X_2) * X_2 \\ cos(X_1 * X_2) * X_1 \end{pmatrix} = \begin{pmatrix} [-0.5, 0.5] \\ [0.3658, 1.5] \end{pmatrix}$$

3 Significance Analysis

Sensitivity based method is most useful in defining the significance of the network parameters. In [7,10], researchers used the network's output to find the sensitivity of network parameters. Sensitivity is defined as the degree to which an output responds to the deviation in its inputs [7]. Deviation in output Δy due to deviation in inputs is the difference of deviated and non-deviated outputs $f((X + \Delta X) * w) - f(X * w)$. Meanwhile, inputs X_i is treated as an interval [0, 1] for finding sensitivity not just on fixed points rather finding it for overall inputs range.

With the above-defined sensitivity, the significance is measured as a product of sensitivity of a node by the summation of the absolute values of its outgoing weights. This approach of significance defined by [7] has few limitations such as it can only be applied to hidden layer nodes, not to the network input layer. Secondly, one has to prune one layer first before moving to the second layer as it works by layer-wise.

To find out the significance for both input and hidden nodes of a network, another method proposed in [10], computes sensitivities by computing partial derivatives of network outputs to input and hidden nodes . Although this method is good in computing sensitivities of network's parameters, there is a high computational cost for this as it computes partial derivative at a given parameter and then finds out the average sensitivity for all training patterns. We proposed a new sensitivity method in section 3.1 based on interval adjoints to tackle the shortcomings of earlier defined sensitivity based pruning methods.

3.1 Interval Adjoint Significance Analysis

Let us first define significance, before applying the proposed method of significance analysis on neural networks to obtain the ranking of nodes in the network.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

According to [20], significance can be defined by the product of width of an interval and absolute maximum of first order derivative of that interval.

$$S_Y(V_i) = w[V_i] * max|\nabla_{[v_i]}[y]| \tag{1}$$

The influence of all input variables X_i, $i = 1, ..., n$ is given in V_j. And this influence can be quantified by width $w[V_j] = v^u - v^l$. Larger the width of an interval larger the influence and vice versa. This means that variable V_j is highly sensitive to input variable X_i and vice versa. But this information alone is not sufficient to define the significance of a variable. Further operations, during the evaluation of Y and different intermediate variables V, may increase or decrease the influence of variable V_j. Therefore, it is necessary to find the influence of that variable V_j on output Y. The absolute maximum of first order partial derivative $max|\nabla_{[v_i]}[y]|$ of variable V_j gives us this influence of variable V_j over output Y.

3.2 Selection of Significant Nodes

Suppose an already trained neural network illustrated in Fig. 1, with four nodes on each hidden layer, four nodes on the input layer and one output node. Thus the problem in this subsection is to find out the significance of each node in the network to correctly find out the output. This concept of significance defined in (1) can be applied to already trained neural nets and deep belief nets to find out the influence of individual input and intermediate nodes in determining the output of the network. There is no change in the standard back-propagation neural network architecture [21,22] except all the variables in the network are of interval type instead of scalar type. Interval arithmetic [23] is used to calculate the input-output relations. We can calculate this significance layer-wise and rank them based on their magnitude. A node with a high significance value is very sensitive to the overall output of the network.

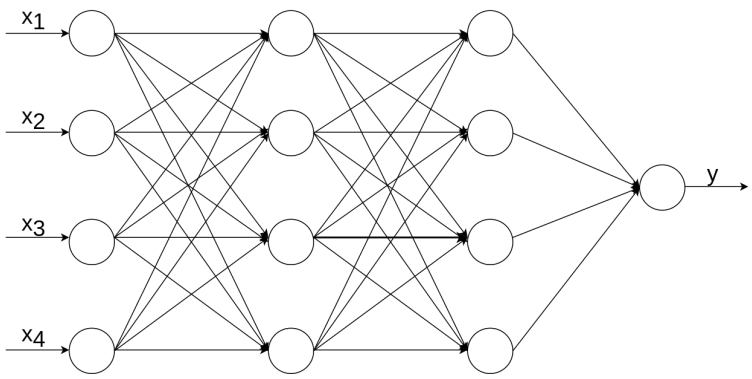

Fig. 1: Simple neural network.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

A single interval input vector is used for finding the significance. Let us assume that m training patters $x_k = \{x_{k1}, ..., x_{kn}\}$, $k = 1, 2, 3, ..., m$ are used to train the neural network. These m training patterns are used to generate the interval input vector for significance analysis by finding out the maximum and minimum value for each input ($e.g. x_1$) from all training patterns $(x_{11}, ..., x_{k1})$. These maximum and minimum values are used to construct the interval input vector $X = \{[min(x_{k1}), max(x_{k1})], ..., [min(x_{kn}), max(x_{kn})]\}$. As scalar is degenerated form of an interval whose upper and lower bounds are the same, one can use the trained weight and bias vector of the network and change them to interval vectors whose upper and lower bounds are the same.

A single forward and backward run of the network is required with new input, weight and bias vector for yielding the significance of intermediate and input nodes. Nodes on each layer can be ranked in decreasing order of significance and it's up to the user to select the number of nodes which will stay in the new architecture. Let's go back to our example of already trained network in Fig. 1. After the significance analysis, we find out middle two nodes on the first hidden layer, first and the last node on second hidden layer is not significant as shown in Fig. 2.

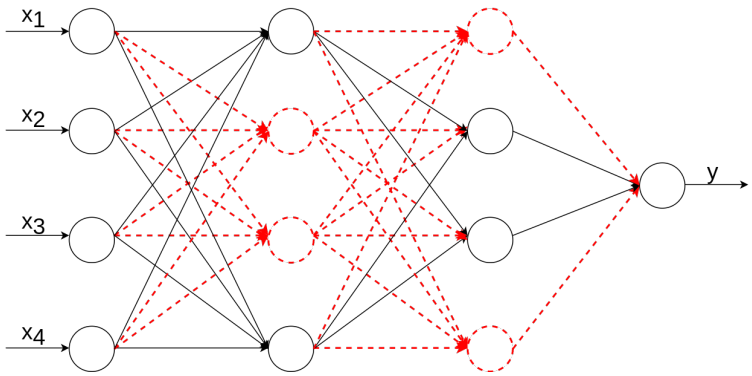

Fig. 2: Insignificant nodes and their connection identified by significance analysis.

3.3 Removal of Insignificant Nodes

After the selection of significant nodes in the network, it is necessary to preserve the information of insignificant nodes before throwing them out from the network to prune it, otherwise, we will be changing the inputs for next layer activations. Insignificant nodes have less impact on the network and the weight associated with its incoming and ongoing connections are mostly redundant and have very low values. Significance analysis together with interval evaluation not only gives us the influence of nodes but it also gives us a guaranteed enclosure that contains

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

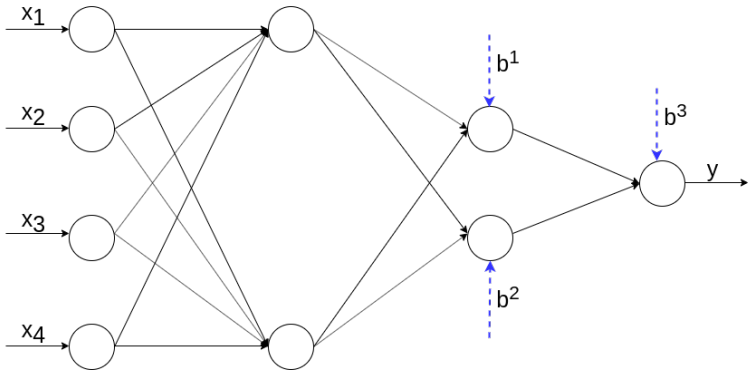

Fig. 3: Removal of insignificant nodes, and their incoming and outgoing connections.

all the possible values for insignificant nodes and their outgoing connections. We can store all the information of the previous layer insignificant nodes into significant nodes of the next layer by calculating the midpoints ($v_j = \frac{(v_j^l + v_j^u)}{2}$) of all incoming connections from previous layer insignificant nodes to significant nodes of next layer. We can sum up all these midpoints and add them as the bias of significant nodes. This process is illustrated in Fig. 3.

4 Experimental Results

There are so many parameters to optimize in all the layers of fully connected neural networks and on the fully connected layers of most of the large-scale convolutional neural networks [24,25]. Currently, we analyzed the performance of our method on fully connected networks to reduce the number of parameters and obtain a compressed network yet achieving the same accuracy. For this purpose, we choose four datasets; MNIST [26], MNIST_ROT [27], Fashion-MNIST [29] and CIFAR-10 [28].

In all the networks, there were two hidden layers with a size of five hundred nodes on each layer, refer to Table 1 for more details. For the activation functions, the sigmoid activation function was used on the hidden layer and softmax on the output layer. Adam with weight decay commonly known as AdamW [30] and Adamax [31] optimization methods were used for parameter optimization. In the initial epochs, the network was trained with AdamW but in the end, Adamax was used for better accuracy.

4.1 Experiments on MNIST

Significance analysis not only can be applied to hidden layers but our method is also effective for the input layer. It works as a feature selection method for

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

Table 1: Datasets and networks used for experiments

Dataset	Architecture
MNIST	784-500-500-10
MNIST_ROT	784-500-500-10
Fashion-MNIST	784-500-500-10
CIFAR-10	3072-500-500-10

inputs. Fig. 4 shows test and train error plots for removed input features after significance analysis. As we can see from flatter curves, the error did not increase after removing the significant number of input features from the original network. But after a certain point error is almost increasing exponentially.

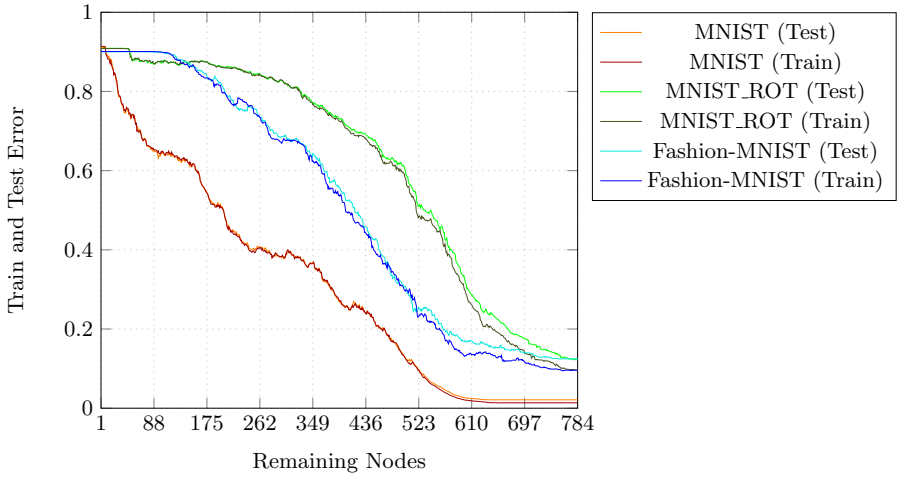

Fig. 4: Removal of insignificant nodes from input layer of MNIST, Fashion-MNIST, and MNIST_ROT data sets

Fig. 5 and 6 are test and train error plots for the first and second hidden layer respectively. Significance analysis was performed on each layer separately. It is clearly shown in Fig. 5 that we can remove more than eighty percent of the nodes from the first layer without compromising accuracy. Fig. 7 shows the result of the significance analysis applied on all hidden layers. If we observe the plots of significance applied to individual hidden layers in Fig. 5 and 6, we can clearly see the pattern when significance is applied on all hidden layers. The error increasing pattern is the same in Fig. 6 as it is in Fig. 7.

Significance analysis works pretty well on CIFAR-10 dataset too and we can remove a few hundreds of input features from the network as shown in Fig. 8. Fig. 5, 6 and 7 show plots of nodes removal from each hidden layer and nodes

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

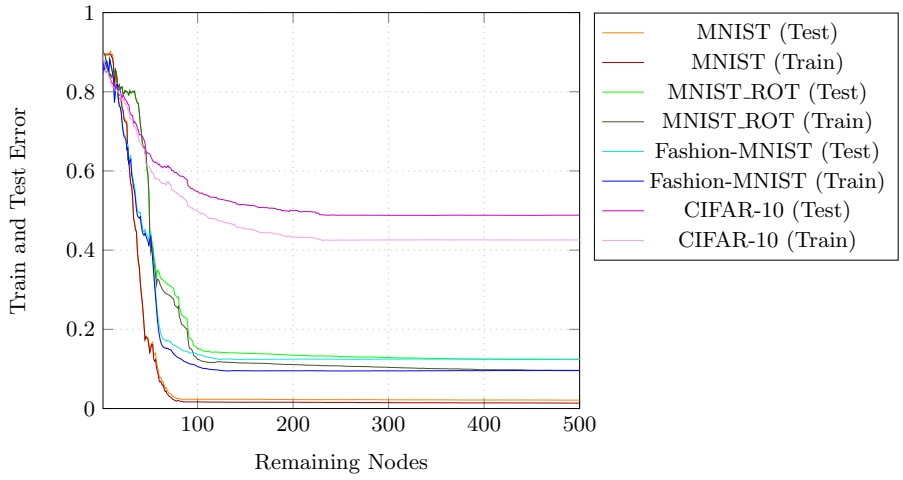

Fig. 5: Removal of insignificant nodes from first hidden layer of MNIST, Fashion-MNIST, MNIST_ROT, and CIFAR-10 data sets

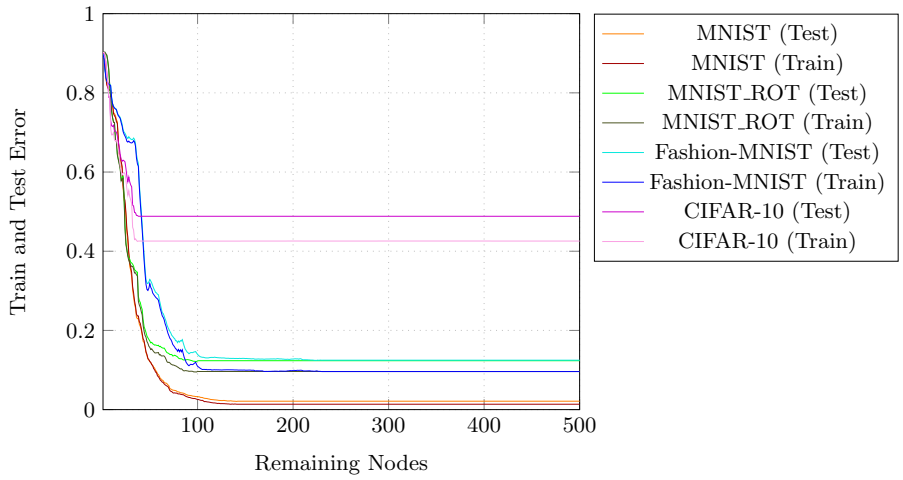

Fig. 6: Removal of insignificant nodes from second hidden layer of MNIST, Fashion-MNIST, MNIST_ROT, and CIFAR-10 data sets

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

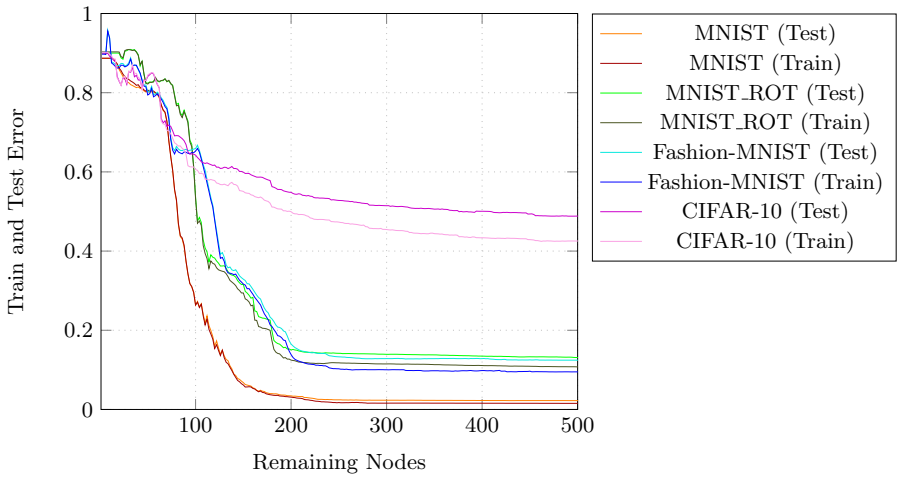

Fig. 7: Removal of insignificant nodes from first and second hidden layer of MNIST, Fashion-MNIST, MNIST_ROT, and CIFAR-10 data sets

removal from all hidden layers in the network after the significance analysis respectively.

4.2 Experiments on CIFAR-10

From the last 11 years, CIFAR-10 has been the focus of intense research which makes it an excellent test case for our method. It has been most widely used as a test case for many computer vision methods. After MNIST, it has been ranked the second most referenced dataset [32]. CIFAR-10 is still the subject of current research [33,34,35] because of its difficult problems. All of its 50k train samples were used to train the network and 10k test samples were used to test the network.

Like MNIST, we can also see the pattern of error increasing when we apply significance analysis on all hidden layers. Error plot of first hidden layer nodes removal in Fig. 5 and error plot of all hidden layer nodes removal in Fig. 7 looks quite the same. This is because the second hidden layer is not contributing much to the overall performance of the network.

4.3 Experiments on MNIST_ROT and Fashion-MNIST

We performed the same experiments on rotated version of MNIST. Error plots for this dataset are given in the appendix. Furthermore, we also quantify the performance of our algorithm on the newly generated Fashion-MNIST dataset.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

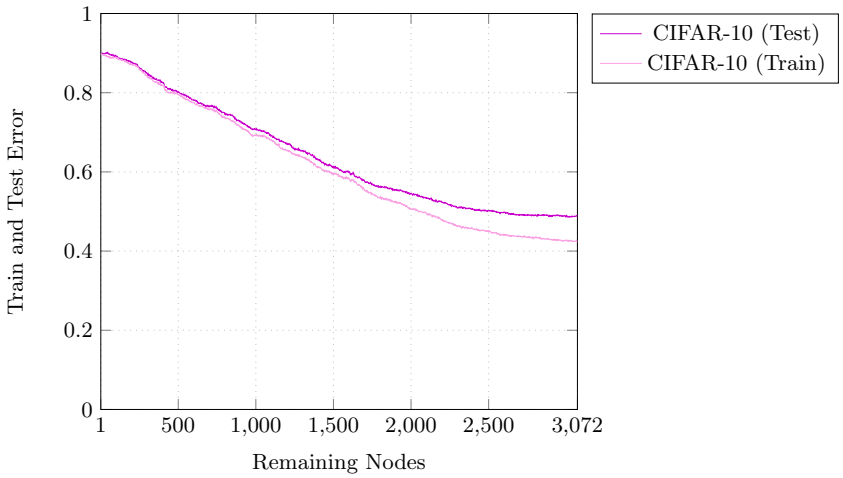

Fig. 8: Removal of insignificant nodes from input layer of CIFAR-10 data set

4.4 Network Performance before and after retraining

After the significance analysis and removing 50% of nodes from the network still, we can achieve the same accuracy on all the datasets used in experiments without additional retraining or fine-tuning. Table 2 lists the train and test error of different datasets on the different percentages of removed insignificant nodes in the network before and after retraining. In the case of MNIST and CIFAR-10, there is no increase in error if we remove 25% or 50% of all the hidden nodes from the network using significance analysis. If we further train them after removing the nodes there is no increase in the accuracy. But on the other two datasets (Fashion-MNIST and MNIST_ROT), we can increase the accuracy if we retrain them after removing the insignificant hidden nodes from the network.

Train and test error slightly increase if we remove 90% percent of the hidden nodes from the network and this is expected as we are taking away too much information from the original network. But we can improve the performance of network with retraining and using the remaining old weight vector and updated bias vector for significant nodes. It is better to use the remaining network connection for retraining than initializing the values again. After retraining, the error rate was decreased significantly for 90% nodes removal from the original network and in some cases decreasing the original error rate that was there before significance analysis.

5 Future Work and Conclsion

A new method of finding and removing redundant and irrelevant nodes from the neural network using interval adjoints is proposed in this paper. Our method

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

Table 2: Training and test error before and after retraining for different percentage of removed insignificant nodes. Initially, all the neural nets have 1000 hidden nodes.

Removal of neurons from hidden layers		90%	80%	75%	50%	25%	0%
MNIST	training error	0.26	0.03	0.01	0.01	0.01	0.01
	after retraining	0.002	0.001	0.0009	0.01	0.01	
	test error	0.26	0.03	0.02	0.02	0.02	0.02
	after retraining	0.02	0.02	0.02	0.01	0.01	
MNIST_ROT	training error	0.53	0.12	0.11	0.10	0.09	0.09
	after retraining	0.02	0.0001	0.0005	0.0004	0.003	
	test error	0.53	0.15	0.14	0.13	0.12	0.12
	after retraining	0.13	0.10	0.10	0.10	0.10	
Fashion-MNIST	training error	0.65	0.13	0.10	0.09	0.09	0.09
	after retraining	0.05	0.04	0.03	0.03	0.03	
	test error	0.65	0.16	0.13	0.12	0.12	0.12
	after retraining	0.11	0.12	0.11	0.12	0.11	
CIFAR-10	training error	0.60	0.49	0.47	0.42	0.42	0.42
	after retraining	0.46	0.42	0.43	0.42	0.42	
	test error	0.64	0.54	0.52	0.48	0.48	0.48
	after retraining	0.50	0.49	0.49	0.48	0.48	

finds out the significance of hidden as well as input nodes. The significance depends upon two factors, the impact of a node on output and width of a node interval. The use of interval data and finding sensitivities with interval adjoints make our method more robust than multiple existing methods. The results presented in this paper indicate indicate that the significance analysis correctly finds out irrelevant input and hidden nodes in a network and it also gives us much information to update the bias of relevant nodes so that performance of the network does not comprise by removing irrelevant nodes.

Our future work will be aimed at applying interval adjoint significance analysis on convolutional and fully connected layers of convolutional neural networks. Furthermore, investigation will be carried out on applying significance analysis during the training of a network and speed up the training process by eliminating the less significant nodes from the network.

References

1. Augasta, M. G., and Kathirvalavakumar, T. (2013). Pruning algorithms of neural networksa comparative study. Central European Journal of Computer Science, 3(3), (pp. 105-115).
2. Reed, R. (1993). Pruning algorithms-a survey. IEEE Transactions on Neural Networks, 4(5), (pp. 740-747).
3. Fahlman, S. E., and Lebiere, C. (1990). The cascade-correlation learning architecture. In Advances in neural Information Processing Systems (pp. 524-532).

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

4. Castellano, G., Fanelli, A. M., and Pelillo, M. (1997). An iterative pruning algorithm for feedforward neural networks. IEEE Transactions on Neural networks, 8(3), (pp. 519-531).
5. Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.
6. Xu, J., and Ho, D. W. (2006). A new training and pruning algorithm based on node dependence and Jacobian rank deficiency. Neurocomputing, 70(1-3), (pp. 544-558).
7. Zeng, X., and Yeung, D. S. (2006). Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure. Neurocomputing, 69(7-9), (pp. 825-837).
8. Lauret, P., Fock, E., and Mara, T. A. (2006). A node pruning algorithm based on a Fourier amplitude sensitivity test method. IEEE Transactions on Neural Networks, 17(2), (pp. 273-293).
9. Hassibi, B., Stork, D. G., and Wolff, G. J. (1993). Optimal brain surgeon and general network pruning. IEEE International Conference on Neural Networks, (pp. 293-299).
10. Engelbrecht, A. P. (2001). A new pruning heuristic based on variance analysis of sensitivity information. IEEE Transactions on Neural Networks, 12(6), (pp. 1386-1399).
11. Griewank, A., and Walther, A. (2008). Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM.
12. Naumann, U. (2012). The art of differentiating computer programs: an introduction to algorithmic differentiation. SIAM.
13. Hascoet, L., and Pascual, V. (2013). The tapenade automatic differentiation tool: Principles, model, and specification. ACM Transactions on Mathematical Software (TOMS), 39(3).
14. Corliss, G., Faure, C., Griewank, A., Hascoet, L. and Naumann, U. (2013). Automatic Differentiation of Algorithms. New York, NY: Springer.
15. J. Lotz, K. Leppkes, and U. Naumann, dco/c++ - derivative code by overloading in C++. Available: https://www.stce.rwth-aachen.de/research/software/dco/cpp
16. Lotz, J., Naumann, U., and Ungermann, J. (2012). Hierarchical algorithmic differentiation a case study. In Recent Advances in Algorithmic Differentiation (pp. 187-196).
17. Towara, M., and Naumann, U. (2013). A discrete adjoint model for OpenFOAM. Procedia Computer Science, 18, (pp. 429-438).
18. Lotz, J., Schwalbach, M., and Naumann, U. (2016). A case study in adjoint sensitivity analysis of parameter calibration. Procedia Computer Science, 80, (pp. 201-211).
19. Schichl, H., and Neumaier, A. (2005). Interval analysis on directed acyclic graphs for global optimization. Journal of Global Optimization, 33(4), (pp. 541-562).
20. Deussen, J., Riehme, J., and Naumann, U. (2016).Interval-adjoint significance analysis: a case study.
21. Kelley, H. J. (1960). Gradient theory of optimal flight paths. Ars Journal, 30(10), (pp. 947-954).
22. Rojas, R. (1996). The backpropagation algorithm. Springer, In Neural networks (pp. 149-182).
23. Moore, R. E. (1979). Methods and applications of interval analysis. Society for Industrial and Applied Mathematics.
24. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1097-1105).
25. Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27

26. LeCun and C. Cortes. (2010). MNIST handwritten digit database. Available: http://yann.lecun.com/exdb/mnist/
27. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. ACM Proceedings of the 24th International Conference on Machine learning (pp. 473-480).
28. Krizhevsky, A., and Hinton, G. (2009). Learning multiple layers of features from tiny images. 1(4), (pp. 7). Technical report, University of Toronto.
29. Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747.
30. Loshchilov, I., and Hutter, F. (2017). Fixing weight decay regularization in adam. arXiv:1711.05101.
31. Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.
32. Ben Hamner. Popular datasets over time. Available: https://www.kaggle.com/benhamner/populardatasets-over-time/code.
33. Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, 33, (pp. 4780-4789).
34. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., ... and Hodjat, B. (2019). Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing (pp. 293-312).
35. Su, J., Vargas, D. V., and Sakurai, K. (2019). One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_27