International Journal of Engineering & Technology, 7 (4.39) (2018) 210-212

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Network Reduction Impact on Optimisation Algorithms by Predicting Robot Movement

Divyanshu Chauhan, Bhairvee Singh[1],Ishu Varshney[2]

Research Scholar,Research School of Computer Science, Australian National University
Department of Computer Science & Engineering, GLBITM, Gr. Noida, Uttar Pradesh, India
**Corresponding author E-mail: u6447876@anu.edu.au*

Abstract

Recently the size of the neural network has been increasing at a very fast pace. This increases the training time and computation cost required by the neural net. There are various ways to reduce the network to decrease the computation time and resource requirement.This paper measures the impact of network reduction on various optimisation algorithms by predicting Wall-Following robot movement. The network is reduced using the sensitivity of the neurons. Performance of various optimisation algorithms (Adadelta, Adagrad, Adam, Adamax, Rprop and SGD) are compared before and after network reduction. A single hidden layer neural network and a three hidden layered deep neural network are used for this experiment.

Keywords: *network reduction; optimisation algorithm; resources: pruning; sensitivity*

1. Introduction

Neural network is a computer system which is based on the human brain and the nervous system. The network consists of an input layer, hidden layers and the output layer. The input layer is the number of input parameters in the network and the output layer is the number of expected outputs from the network. The hidden layer, which is the main part of the neural network, can contain any number of layers and each layer can contain any number of neurons. In the past 20 years, the number of hidden neurons has increased significantly due to increase in computational power.
Although, the computational power has increased and the number of neurons that can be trained has increased with it, however, high computational cost is still an issue. To reduce the cost there are many network reduction techniques that can help reduce the size of the network. There are various simple and complex network reduction techniques, however, for the purpose of this paper we will be focusing on sensitivity.
Sensitivity (Karnin 1990) is a measure which can be calculated by the change in weight of each neuron after each epoch. The neurons whose weight change the most are more sensitive, whereas, the neurons whose weight does not change by high degree have less sensitivity. This implies that the neurons that have the least sensitivity do not contribute to any of the output and the optimisation does not change that neuron. Hence, it is not considered a useful neuron and just increases our computation cost.

1.1 Optimisation Algorithms

There are various optimisation techniques available and each of them can react in a different way to network reduction. Hence, we will evaluate the impact of network reduction on the following algorithms:

1.1.1. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD), is the incremental version of the basic gradient descent algorithm. Computing the gradient for the entire dataset can be very slow and computationally expensive. SGD solves this issue, as it can find the gradient using few values as well and it also converges very quickly to local minima. It is one of the basic algorithm used to reduce the error and many algorithms are derived from it.

1.1.2. Adagrad

Adagrad (Duchi 2011), is a variation of SGD with per parameter learning. It uses subgradient method that includes the geometric information from previous iterations. This information helps make the algorithm make more informed learning decisions. This gives it an edge over vanilla SGD.

1.1.3. Adam

Adam (Kingma 2017), is also a gradient based optimisation technique. It is based on dynamic lower-order moment. In this algorithm, second moments and gradients are used for the optimisation. The computational efficiency of the algorithm is good and is suited for a large number of attributes or parameters in the dataset. This algorithm is also suitable for datasets that have sparse or noisy gradients. This algorithm is derived from the above mentioned algorithm Adagrad.

1.1.4. Adamax

Adamax (Kingma 2017), is one of the variants of the above mentioned Adam algorithm and shares similar advantages with it. The only difference is the rule for updating the weights of the neuron is inversely proportional to the previous and present gradients.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1.1.5. Rprop

Rprop (Riedmiller 1993), stands for resilient propagation. It overcomes the disadvantages of pure gradient-descent and is based on the local adaptation of weight according to the error function. Basically, whenever the sign of the gradient change is reversed, the algorithm detects that it has missed a local minima. Then it decreases the weight by a very small amount and checks if the gradient is decreasing.

1.1.6. Adadelta

Adadelta (Zeiler 2012), provides a new learning rate for the gradient descent. The method required less overhead computation as compared to stochastic gradient descent and the learning rate adapts over time. The advantage of this algorithm is that we don't have to assign any learning rate and the algorithm automatically the learning rate for better performance.

2. Method

2.1 Neural Network Architecture

There are two types of neural networks used for this study. The first neural network consists of three layers. The first layer is the input layer and the number of neurons are equal to the number of inputs. The third layer is the output layer which is equal to the number of possible outputs. The middle layer is the hidden layer consisting of 40 hidden neurons.
The second network used consists of five layers. The input and output layers are similar to the above mentioned network. However, there are three hidden layers with 40 hidden neurons each.
Moreover, for this experiment the number of epochs used is 300 and learning rate is 0.05 for all the algorithms. The activation function which has been used for each layer is relu. Finally, on the output layer softmax is applied followed by logarithm. The loss function used for calculating the loss is negative log likelihood loss. All these parameters are kept constant for both the networks and all the algorithms. It is possible to find parameters with better performance for different algorithms, however our aim is not to maximise the performance but it is to compare the effects of pruning on different optimisation algorithms. Keeping these parameters constant makes sure that the external environment for all the algorithms are kept constant.

2.2 Dataset

The dataset used for the purpose of this experiment is obtained from UCI publication. It is wall following robot navigation with of 5456 instances and 24 attributes. The 24 attributes are values of sensors on the robot at different angles. According to these values, 4 possible moves are possible i.e. Move-Forward, Slight-Right-Turn, Sharp-Right-Turn, Slight-Left-Turn. While preprocessing, we convert the move values in the data set to integers. Furthermore, we randomly shuffle and then divide the dataset in 80:20 ratio for training and testing respectively. The dataset is appropriate for this study as it is big and complex enough and represents a real life problem that people are trying to solve.

2.3 Algorithm

For the purpose of this paper, we are reducing the network using the sensitivity of the neuron. The neuron with the least sensitivity is removed every 50 epochs. To calculate the sensitivity we calculate the difference in neuron weight with the original neuron weight. The weight of the neuron with least absolute change is removed. For simplicity of implementation, we maintain a list of removed neurons and convert its weight to zero in every iteration. Moreover, for the deep neural network we maintain a list of each layer's initial weights. We also maintain a zero list for each layer and in every 50 epochs we remove the least sensitive neuron from every layer.
Overall, we end up removing 6 neurons out of 40 from the neural network and 18 neurons out of 120 from the deep neural network.
To check the sensitivity of neurons the following algorithm is used every 50 epochs to remove the neuron. The sensitivity of all the neurons is calculated and the minimum is added to the array. Through that array, the neurons are converted to zero in every epoch.

Algorithm 1: Finding sensitivity of each neuron
If epoch mod 50==0 then
Sensitivity $\sum$ |hidden.weight-initialHidden.weight|
leastSensitivity $\min_{sensitivity}$
leastSensitiveArray.append(least.Sensitivity)
end if

3 Results and Discussion

Two different neural networks and the same reduction technique is used to check 6 different algorithms. The test accuracy for each algorithm is averaged over 100 runs so that it avoids the possibility of a rare good result by the algorithm. The results are shown in Table 1 for single layered neural network and table 2 for deep neural network.
For single layer neural network the accuracy with all the optimisation algorithm remained approximately the same before and after pruning. The highest decrease in accuracy is 1.24% which is observed with rprop and the highest increase in accuracy is 0.41% which is observed with adagrad. Adam algorithm proved to be very consistent after pruning as the accuracy is exactly the same even after average of 100 iterations. With other algorithms the change is almost negligible.

Table 1: Testing accuracy of various algorithms before and after network reduction for single layered neural network.

Optimisation Algorithm	Actual test accuracy	After reduction accuracy
SGD	61.00	60.95
Rprop	95.12	93.88
Adadelta	58.55	58.48
Adagrad	84.99	85.40
Adam	92.38	92.38
Adamax	90.25	90.50

Table 2: Testing accuracy of various algorithms before and after network reduction for deep neural network.

Optimisation Algorithm	Actual test accuracy	After reduction accuracy
SGD	51.97	54.75
Rprop	94.25	72.06
Adadelta	49.42	48.74
Adagrad	87.58	87.07
Adam	90.15	88.76
Adamax	94.40	92.80

In contrast to single layered network, the deep network experienced much significant decrease in accuracy with the only exception of SGD which experienced increase in performance. Even in deep network Rprop experienced the highest decrease of 22.19% and SGD experienced the highest performance gain of 2.78%.

The overall pattern that pruning affected deep neural network more can be attributed to the fact that more pruning has been performed in the deep network. Although the proportion of pruning is same, however more learning is lost in the deep neural network. This should be the primary reason for this performance difference.

The main reason for rprop being affected the most could be associated with the fact that each weight has an individual evolving update-value. And the weight-step is only determined by its update-value and the sign of the gradient. Hence, this implies that some of the weights might not change drastically initially but they might start changing after some iterations, when they hit a steep slope. As we are not really removing the pruned neuron, the Rprop algorithm might still try to learn the pruned neurons. This might be affecting the algorithm performance after pruning.

In the research paper (Freire et al. 2009), this data set has the maximum accuracy of 97.59% with a multilayer perceptron network. As compared to this our best performance is slightly low at 95.12%. Although our performance is low, it is averaged over 100 iterations. Moreover, our neural networks are not optimised for best performance because we had to keep the external environment constant for both the neural networks and had to settle for good performance for both.

Also note, that the way that we have implemented pruning might have some impact of the performance. Mainly because we don't actually remove the neuron from the network and we just zero the output of the neuron. Hence, the algorithm might still keep changing the weight even after we have removed it and result in more loss of learning.

4 Conclusions and Future Work

The study compared the impact of network reduction on six different neural network optimisation algorithm and on two different neural networks. It was a classification task and the dataset used for this experiment was Wall-Following Robot Navigation. The network reduction, decreased the performance of some algorithms, whereas, increased it for some. Overall, for the single layered neural network the impact was very less, however, for deep neural network the decrease in performance was substantial.

Overall, the performance decrease of single layer network was very less and we can employ pruning with single layer networks. However, as can be seen with the results pruning with deep neural networks impacts the performance and should only be employed if the performance decrease if not substantial in that particular case.

In future works, we aim to integrate genetic algorithm on the network parameters such as learning rate, hidden neurons, number of epochs, etc. This can ensure that each of our network reaches the maximum capability and then the actual impact of pruning can be better estimated.

References

[1] Duchi, J., Hazan, E. and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pp.2121-2159.

[2] Freire, AL, Barreto, GA, Veloso, M and Varela, AT 2009 'Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study', *In Robotics Symposium (LARS), 2009 6th Latin American* (pp. 1-6). IEEE.

[3] Gedeon, TD and Harris, D 1991, 'Network Reduction Techniques', *Proceedings International Conference on Neural Networks Methodologies and Applications, AMSE*, vol. 1, pp. 119-126.

[4] Hagiwara, M 1990, 'Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection', *IJCNN*, vol. I, pp. 625-630.

[5] Karnin, ED 1990, 'A simple procedure for pruning back-propagation trained neural networks', *IEEE Transactions on Neural Networks*, vol 1., pp. 239-242.

[6] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[7] Mozer, MC, Smolenski, P 1989, 'Using relevance to reduce network size automatically', *Connection Science*, vol. 1, pp. 3-16.

[8] Riedmiller, M. and Braun, H., 1993. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*(pp. 586-591). IEEE.

[9] Zeiler, M.D., 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.