Reduction of Hidden Units and Parameters in Deep Neural Networks

Tian Huang

Research School of Computer Science,
the Australian National University
u6343091@anu.edu.au

Abstract: Deep neural network is a helpful deep learning method and can help people solve different kinds of problems. However, it is very difficult to decide the dimension of hidden layers. Too many parameters can slow down the run-time of training a network. In this paper, I am going to talk about two methods to prune unnecessary hidden units. Hidden units that have low *contributions* to outputs or fixed output will be pruned. And later, I will extend this method to prune unnecessary parameters using a method in deep learning called *partial connection*. Another method is to combine hidden units whose *distinctiveness* are similar by comparing vectors consist of outputs or parameters. I am going to build a deep neural network using two methods. The dataset I used to examine two methods is *Breast Cancer Wisconsin (Diagnostic) Data Set* collected by University of Wisconsin. The result shows that using these two methods can decrease run-time by 16.56% and 45.32% while the accuracy only increases by 0.0% and 0.58%.

Keywords: Deep Neural Network, Reduction, Prune, Partial connection, Hidden Units, Parameters, Contribution, Similarity, Run-Time, Training Time

1. Introduction

Deep neural networks (DNNs) are very useful tools to solve classification and regression problem. However, there are two major problems in DNNs: overfitting and computation time. In this paper, I will focus on reducing computation time while not having great change on accuracy.

The dimension of the input layer and output layer can be easily decided. However, it is a lot more difficult to decide the dimension of the hidden layer. If the dimension is too small than the ideal value, the accuracy of prediction can be low. On the other hand, if it is larger, it takes a long time to train the network and takes more memory to store the network. So, it is very important to find an ideal value for the dimension of the hidden layer.

A solution to this problem is that we can assume a relatively large number of hidden units. An empirical and brief dimension of hidden layers is equal to the dimension of the input layer. Then I can start to prune the unnecessary hidden units in the neural network. In this paper, two methods will be used. One is to prune a unit whose weights linking to output are all zero, very 'small' [2]. Using product of weights and outputs of the last layer to compare, it is easy to find them. Moreover, with bias included, hidden units with fixed outputs are identified. Later I will extend this method to prune parameters using *partial connection* [1], a method similar to *sparse connectivity* in convolutional neural network. Another is to remove those hidden units that are similar. We can form vectors using the output of each hidden units and compare their *distinctiveness* calculating the angle between vectors [2].

If the angle is close to specific degrees, such as 0 degrees or 180 degrees, I can remove it because two hidden units will stay the same in the training process afterwards.

The figures in the paper are presenting simple neural networks, but the technique stated are applicable to DNNs. The experiment is also conducted on a 4-layer deep neural network.

2. Method

2.1 Basic (deep) nerual networks and variables mentioned in the paper

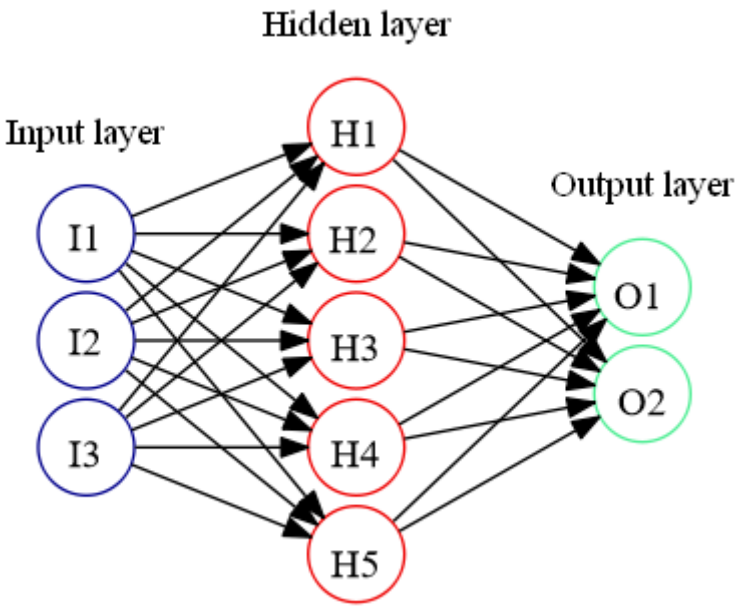

Figure 1 – A classical neural network. Although is a simple neural network, it works the same as DNN. (Figure is generated with Graphviz)

Assume I have a neural network as the one shown in figure 1. I1, I2, I3 are 3 input units. H1, H2, H3, H4, H5 are 5 hidden units. O1 and O2 are 2 output units. I call these three layers as layer 0, layer 1 and layer 2. Between each two units from adjacent layers, there is a weight. The weights between *i* th unit of layer *n-1* and *j* th unit of layer *n* which is at the behind is called w_{nij}. The bias of *i* th unit of layer *n* is called b_{ni}. The output of *i* th unit of layer *n* is called z_{ni} and its corresponding value after activation function is applied is called a_{ni}. Dimension of layer *n* is called d_n.

As is suggested above, the output of each unit and the output modified with activation function is applied can be calculated using formulas below.

$$z_{ni} = \sum_{j=0}^{d_{n-1}} w_{nji} * a_{(n-1)j} + b_{ni}$$

$$a_{ni} = acitvation\ function(z_{ni}) \tag{1}$$

2.2 Prune hidden units with zero or fixed output and parameters (weights) with small contribution to the output

Prune hidden units with output close to zero

If a_{ni} is too small compared to other a_{nj} (j = 0 … d_n), I can judge that $z_{(n+1)i}$ will not change a lot if the hidden units corresponding to a_{ni} do not exist [2]. This hidden units will be marked as unnecessary hidden units and can be pruned. This will reduce training time because in the further forward propagation and backward propagation process, hidden units are reduced. This means less calculations.

Prune hidden units with fixed output

Cause the output of each unit z_{ni} equals to the sum of prothe duct of weights and output of lasthe t layer $\sum_{j=0}^{d_{n-1}} w_{nji} * a_{(n-1)j}$ plus bias b_{ni} and bias b_{ni} is a fixed number with no relationship with output of last layethe r $a_{(n-1)the\ j}$. If the sum of product $\sum_{j=0}^{d_{n-1}} w_{nji} * a_{(n-1)j}$ is too small compared to bias b_{ni}, z_{ni} can be viewed as a fixed number b_{ni} ($z_{ni} = b_{ni}$). Its *contribution* in the next layer is calculated as below.

$$contribution = w_{(n+1)ik} * a_{ni} = w_{(n+1)ik} * acitvation\ function(z_{ni}) \approx w_{(n+1)ik} * acitvation\ function(b_{ni})\ (k \text{ in } 0 \dots d_{n+1}) \tag{2}$$

This is also a fixed number and can be replaced by bias in the next layer $b_{(n+1)k}$. Only thing we need to do is to add $w_{(n+1)ik} * acitvation\ function(b_{ni})$ to $b_{(n+1)k}$. Then we can prune this hidden unit.

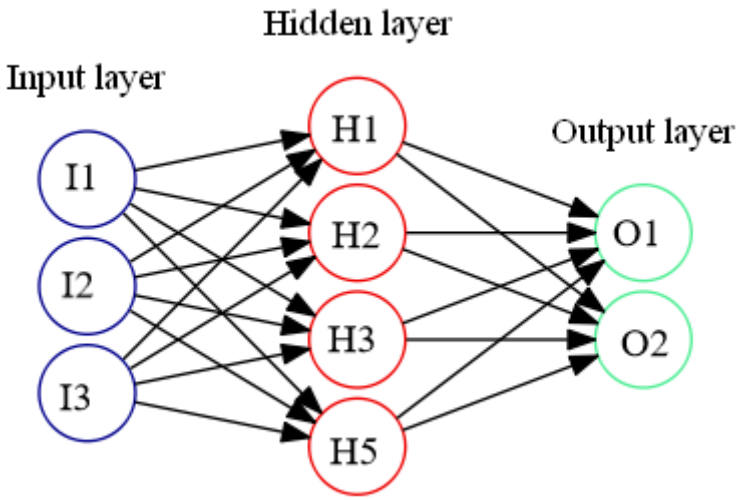

Figure 2 – A neural network after pruning unnecessary hidden unit H4 (Figure is generated with

Graphviz)

Prune parameters (weights) with little contributions to output

Similar to the method stated above, if we extend this to bias in the hidden units, we can prune parameters (weights) that have little *contribution* to the output of its hidden unit it is in. For example, if $w_{nki} * a_{(n-1)k}$ $(d_{n-1} \geq k)$ is always very small compared to other $w_{nji} * a_{(n-1)j}$. Then the output of this hidden unit can be calculated as below.

$$z_{ni} = \sum_{j=0}^{d_{n-1}} w_{nji} * a_{(n-1)j} + b_{ni} \approx \sum_{j=0}^{k-1} w_{nji} * a_{(n-1)j} + \sum_{j=k+1}^{d_{n-1}} w_{nji} * a_{(n-1)j} + b_{ni} \quad (3)$$

But how can I prune parameters? I use *partially connected* neural networks [1]. This is a method similar to *"sparse connectivity"* in convolutional neural networks. Compared to traditional fully connected neural network, not every unit between adjacent layers have weights. Figure 3 shows how a partially neural network is like.

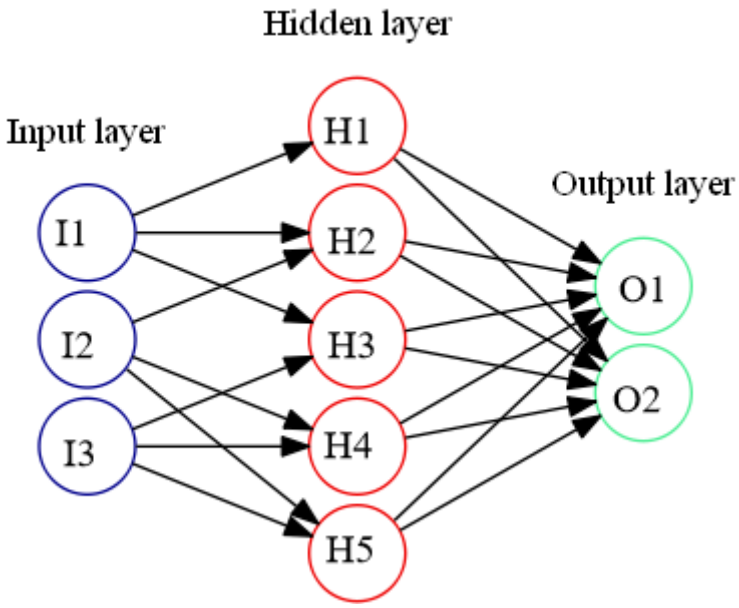

Figure 3 – A partially connected neural network with several weights removed (Figure is generated with Graphviz)

The benefit of partially connected neural network is that it with less parameters and takes less memory to store the network. It takes less time to train the neural network [1].

2.3 Combine hidden units similar to other hidden units in the same layer

Measuring distinctiveness using vectors consist of output of hidden layers

Suggested we have *n* examples of input, for a hidden layer consists of d_{hid} hidden units, I can create d_{hid} vectors with *n* dimension. I can calculate the angle between vectors using formula below.

$$vector1 = (v_{1.1}, v_{1.2}, \dots, v_{1.n}), vector2 = (v_{2.1}, v_{2.2}, \dots, v_{2.n}),$$

$$angle < vector1, vector2 > = \frac{v_{1.1} * v_{2.1} + v_{2.1} * v_{2.2} + \cdots + v_{1.n} * v_{2.n}}{\sqrt{v_{1.1}^2 + v_{1.2}^2 + \cdots + v_{1.n}^2} * \sqrt{v_{2.1}^2 + v_{2.2}^2 + \cdots + v_{2.n}^2}} \quad (4)$$

The hidden units that are similar will have the same process of forward propagation and back propagation in further training [2]. This makes no sense if two hidden units are almost same and training them separately.

The process of combination is based on the feature of the activation function. Different activation functions have a different way to combine. Activation functions like *tanh* or *sigmoid* function which have only one center of symmetry can combine the hidden units that angle is 0 degrees or 180 degrees and whose norm $\sqrt{v_{1.1}^2 + v_{1.2}^2 + \cdots + v_{1.n}^2}$ and $\sqrt{v_{2.1}^2 + v_{2.2}^2 + \cdots + v_{2.n}^2}$ are the same. Linear functions like *ReLU* or *Leaky ReLU* do not have to ensure they have same northe m because they have unlimited centers of symmetry. I only have to ensure their angles are 0 degrees or 180 degrees.

Hidden unit index	0	1	2	3	4	5
11	4.547818e+01	43.436955	134.339205	90.596161	142.836400	137.665853
12	1.025003e+02	105.116329	128.906264	5.221897	83.908833	75.523460
13	1.603309e+02	164.458962	59.659853	64.105617	24.065815	15.510054
14	1.478021e+02	148.079714	51.208168	77.079853	30.654025	32.778888
15	5.052720e+01	50.606886	73.611691	161.574199	118.340211	131.154020
16	1.596214e+01	16.923466	119.128559	119.431443	154.514514	165.756427

Table 1 – An example of the angle between vectors consist of the output of different hidden units from the same layer

As can be seen from the table, the angle between hidden unit 5 and hidden unit 16 is 165.75 degrees which are close to 180 degrees. If these two hidden units are combined, both two hidden layers will be pruned. The angle between hidden unit 3 and hidden unit 12 is 5.22 degrees which are close to 0 degrees. If these two hidden units are combined, the weight of one hidden units will be doubled and another will be pruned.

Measuring distinctiveness using vectors consist of weights of hidden layers

In the previous method, I use vectors consist of the output of hidden layers to measure their *distinctiveness*. But there are some problems. Some vectors consist of output may not show the *distinctiveness*. For example, below is a series of data.

$$z_{1.1} = 1 * a_{0.1} + 2 * a_{0.2} + 3 * a_{0.3} + 3$$
$$z_{1.2} = 3 * a_{0.1} + 2 * a_{0.2} + 1 * a_{0.3} + 3$$
$$\text{example1: } a_{0.1} = 1,\ a_{0.2} = 1, a_{0.3} = 1$$
$$\text{example2: } a_{0.1} = -1,\ a_{0.2} = -1, a_{0.3} = -1$$
$$vector_{1.1} = (9, -3),\ vector_{1.2} = (9, -3)$$
$$angle < verctor_{1.1}, vector_{1.2} >= \arccos\left(\frac{9 * 9 + (-3) * (-3)}{\sqrt{9^2 + (-3)^2} * \sqrt{9^2 + (-3)^2}}\right) = \arccos(1) = 0 \quad (5)$$

The angle between two hidden units is 0 degrees. Accordingly, I think their *distinctiveness* is 0. However, if I investigate how $z_{1.1}$ and $z_{1.2}$ are calculated, they are totally different. Using vector consist of output can be disturbed by special data.

An improvement is using vectors consist of weights and bias. Because weights and bias will not be disturbed by special data and can represent the features of hidden units. Another benefit is that, usually, the number of examples is larger than the number of hidden units. Using weights and bias can reduce the dimension of vectors, saving run-time and memory.

3. Dataset

The data used in the experiment is *Breast Cancer Wisconsin (Diagnostic) Data Set* collected by University of Wisconsin. It has 30 real-valued input features and the diagnosis and 569 instances. 70% of instances are used for training and 30% for testing. The purpose of this dataset to classify whether tumor is benign one or malignant one.

Preprocess of this dataset is to change "Malignant" to “1” and "Benign" to “0” and rescale (normalize) the data from 0 to 1.

4. Results and Discussion

This experiment runs on a computer with Intel i7-7700HQ, 16GB main memory, windows 10 operating system and runs with jupyter notebook.

The experiment runs on 4 models, one without any new methods used [3], one with the first method, one with the second method and one with both methods. Four models are all built with a 4-layer DNN. Dimension of input and output layer are 30 and 1. Every dimension of hidden layer is 30. So, there are 90 hidden units and 2730 weights. I will examine how well models work by comparing their training time (run-time) and accuracy. This is only 1 hidden layer in the network with a dimension of 30 (the dimension of input).

	Model 1 [3] (without new method)	Model 2 (with two methods)	Model 3 (with the method stated in 2.2)	Model 4 (with the method stated in 2.3)
Hidden units pruned because low contribution	0	31	31	0
Hidden units pruned because low distinctiveness	0	19	0	47
Parameters (weights) because low contribution	0	47	50	0
Run-time (training time) (seconds)	10.199704647064209	5.117308616638184	8.51022982597351	5.577071189880371
Time saved	NA	49.828855 %	16.563958 %	45.321248 %
Accuracy	2.339181 %	2.923977 %	2.339181 %	2.923977 %
Change on accuracy	0 %	0.584796 %	0 %	0.584796

Table 2 – Result of an experiment using different models applied to same DNN

The model 1 is used the neural network stated in the paper of W.N. Street, W.H. Wolberg and O.L. Mangasarian published in 1993. But I modified the neural network and make it in the same structure of neural network so that the comparison makes sense.

The result shows that run-time is reduced with only limited influence on accuracy. The methods work for this classification problem and dataset.

5. Conclusion and Future Work

The result of the experiment shows that pruning hidden units with zero or fixed output, parameters (weights) with a small *contribution* to the output, and similar with other hidden units (low *distinctiveness*) can reduce training time without having great change on accuracy. Using *partial connection* and vectors consist of weight and bias make these two methods to function better.

Because how is the network shaped, reduction of parameters has a limited improvement in training time (only 50 out of 2730 weights are pruned) although the *partially connected* neural network is applied. Future work can be work on how to improve the network and pruning algorithms to suit this method better.

6. References

[1] Elizondo, D., & Fiesler, E. (1997). A Survey of Partially Connected Neural Networks. *International journal of neural systems, 8*(5-6), pp. 535 - 558. doi:10.1142/S0129065797000513

[2] Gedeon, D. T., & Harris, D. (1991). Network reduction techniques. *Conference on Neural Networks Methodologies and Applications, 1*, pp. 119-126. doi:10.1371/journal.pone.0103006.g001

[3] Mangasarian, O., Street, W., & Wolberg, W. (1993). Nuclear feature extraction for breast tumor diagnosis. *International Symposium on Electronic Imaging: Science and Technology, 1905*, pp. 861-870. doi:10.1117/12.148698