Predicting Absenteeism At Work using ANN, Effects of Pruning By Badness, and Deep NNs

Aditya Sinha

Research School of Computer Science
College of Engineering and Computer Science
Australian National University
ACT 0200 Australia
u6010425@anu.edu.au

Abstract. Feed forward networks that train using back-propagation have been a much discussed subject in industry and research. Such networks facilitate systems that given the right type (and size) of data, predict results or classify objects with relative ease.

One common complaint with such systems has been the following - there seems to be no reliable way of knowing in advance, the number of hidden layer neurons needed for optimum functioning of the network. And often, satisfactory results are obtained after overestimating this number, in which case, the system can be made more efficient if these neurons are pruned. This paper studies the effects of one such pruning strategy: pruning by Badness factor.

Our network predicts absenteeism at work using factors such as employee weight, height, BMI, distance from workplace, prior record of social smoking/drinking and so on. We do this by training our network on a record of employee absenteeism from a courier company in Brazil. The ANN classifies the absenteeism into one of three categories, and we assess the accuracy of the system using various means. Then we prune one of the excessive nodes, and re-evaluate our system's accuracy. While accuracy is seen to marginally improve in 60% of the cases, it stays the same in 10% of the runs and decreases in the remaining 30%. Pruning by badness is found to be a not too reliable strategy for reducing extra neurons. The paper also discusses the application of Deep Neural Networks to the same problem, although the implementation ends up not working.

Keywords: Artificial Neural Networks, Pruning, Badness, Deep Neural Networks

1 Introduction

Absenteeism is the absence of an employee from their workplace. Predicting the extent and frequency of the absenteeism of employees can be beneficial to companies

and organisations in terms of productivity and cost.

We worked with the Absenteeism at work data set from the UC Irvine Machine Learning Repository. It contains 21 attributes, one of them being absenteeism in hours. The 740 instances present in the dataset are sufficient to train and test a neural network that predicts the absenteeism of a given employee by looking at their data. The dataset is a record of employee absenteeism from a courier company in Brazil. It is well organised and has no missing values. The aforementioned reasons make the dataset suitable for our task. The last attribute in the table was the Absenteeism time in hours, this is what we aimed to predict. The other 20 attributes were: Individual Identification (ID), Reason for absence (a categorical variable in numerical form, one number denoting each of the 28 possible categories), Month of absence, Day of The Week, Seasons, Transportation Expense, Distance from residence to work, Service Time, Age, Work load Average/day, Hit target (achievement of periodic goals in percentage), Disciplinary failure, Education, Son, Social Drinker, Social smoker, Pet, Weight, Height and Body Mass Index. Although the dataset had the target attribute in numerical value (hours of absenteeism), for a more comprehensive analysis of results, it was decided that this was best turned into a classification problem. So the absenteeism hours were divided into three classes. (More discussion about this in next section). Once the neural network was ready and running, we analysed the results of the test runs and worked on ways to improve the system. A simple neural network can be improved in roughly two ways - the first being accuracy (in terms of its results), the second being efficiency (in terms of number of hidden nodes, or time/training taken to reach the results). We use pruning for this purpose and measure the results primarily in terms of classification accuracy.

1.1 Badness

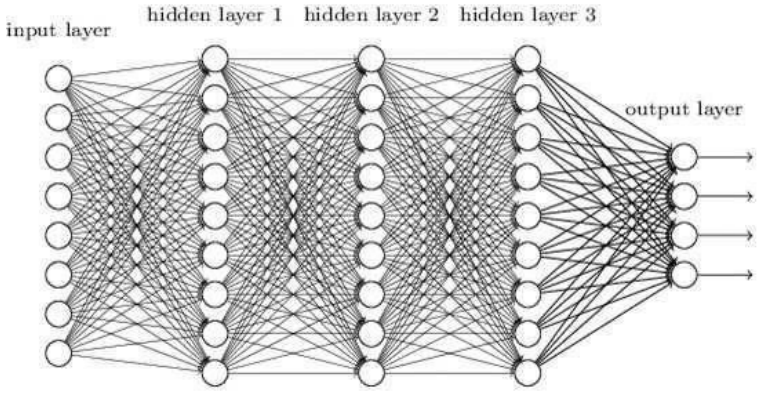
Badness factor for a hidden unit is the sum of back propagated error components over all patterns [2]. So for a dataset with k number of patterns, the badness factor for each node in layer n would be calculated by:

- Feeding the data forward pattern by pattern (1 to k).
- Finding the error component from each node in layer n+1 to each node in layer n.
- Adding the above error components node-wise, so we have the net error for each node in layer n.
- Storing these values and adding to them again and again, the node-wise error components for each pattern.
- Once all k patterns have been iterated through, we find the absolute value of these accumulated error components. These are the badness factors for each node in layer n.
- We eliminate the node with the highest badness factor.

Although pruning by badness may be simple in approach, it's arguable whether it's a good strategy or not. Removing the node with the highest badness factor is akin to removing the node that has done the maximum amount of learning, and is an approach susceptible to backfiring in terms of accuracy.

1.2 Deep Neural Networks

A deep neural network is an artificial neural network with multiple hidden layers between the input and output layers. It's simply a 'deeper' feed forward network with a greater number of hidden layers. There is no such agreed number of hidden layers beyond which a network is considered to be deep, so the 'depth' is a rather soft property. It is often argued that a simple neural network (with a single hidden layer) with a finite number of hidden nodes can approximate almost any function, and can in principle learn anything. However, deep neural networks have been seen performing better for certain kinds of problems. The extra layers may enable composition of features from lower layers, potentially modelling complex data with fewer nodes than a similarly performing shallow network. DNNs have seen a massive surge in popularity over the last five years.



2 Method

2.1 Preprocessing

Our first step was to make the data ready for use. This involved taking a look at the dataset, and encoding and formatting the data for the best results. Because the identification number of an employee did not play into their likelihood of absence, this column was dropped. This left the table with one less attribute than it started with. Next, we looked upon the attribute 'Reason of absence'. Being a key causal factor in the absence of employees, this was definitely an important attribute. Our first step would be to convert the categories into numerical values. But fortunately, the dataset already contains the categories in numerical form. But this would mean that a reason corresponding to a high numerical value would affect the network more than the a reason with lower value. But there was no such order in the importance of these reasons. In order to make sure these numerical values did not affect our results, we used one hot encoding.

One hot encoding transforms categorical attributes into a format that works better with classification neural networks. This is done by representing each category with a boolean attribute. The attribute is true only for rows which belong to the corresponding category. For example:

ID	Species	Numerical value
1	Human	0
2	Tuna	1
3	Dog	2
4	Tuna	1

ID	Human	Tuna	Dog
1	1	0	0
2	0	1	0
3	0	0	1

4	0	1	0

Once this is done, all input nodes corresponding to these categories carry the same input value. Our next step was to normalise input data. Although it's not compulsory, normalising helps reach convergence faster. For all data in each column in the dataset except the target attribute (Absenteeism time in hours), we subtracted the mean of the column, and divided this difference by the standard deviation. This scaled down all values to relatively small numbers in close range with each other, making it easier to approach the global minima of the error function.

2.2 Splitting the data

Once the data has been encoded and normalised, we move onto our target attribute. Because we're making a classifier for the purpose of this task, we divide and encode the target values into three categories:

- Less than or equal to 10 hours => 0
- Between 10 and 40 hours => 1
- Greater than 40 hours => 2

Other intervals were tried, but the results showed a lack of balance. Because a vast majority of employees show an absence of less than 10 hours, these intervals were chosen to allow as many entries as possible into the other two categories for better balance. This was not made a binary classification because that would leave too much to interpretation, reducing the utility of the system. For example: If we divided the data into two classes on each side of a set point (say 60 hours), about 95% of the entries would still fall to the left of 60 hours. Furthermore, this would categorise an employee absent for 120 hours and a different employee absent for 61 hours in the same class.

The dataset was shuffled and then randomly split 80:20 for training and testing sets. Both these sets were further split into input and target values.

2.3 The neural network

The neural network was created using PyTorch. We PyTorch.nn module was extended to make a simple two layer network. The number of nodes in the input layer was equal to the final number of attributes after preprocessing (46 in total). There were three output nodes one for each of the possible categories. There's no set way to determine an optimum number of hidden nodes for the hidden layer. In a good

number of cases, the number of hidden nodes is found to lie somewhere between the number of input and output nodes. So for our network, we started with the mean of the two. This is a simple feed forward back propagation neural net, so we used sigmoid function with softmax.

2.4 Pruning

The next bit involves finding the badness factors for each node in the hidden layer, and pruning the excessive ones. Although pruning by badness itself is a rather simple approach, its implementation through pyTorch is deceptively tricky. The training input is split into patterns to be fed forward into the network. It's important that this be done, since the forward() function in pyTorch does not work with a Tensor of less than one dimensions. This means feeding the data row by row requires a different (manual) approach, that would make the usage of pyTorch rather pointless, thus eliminating that possibility.

Once these patterns are being fed, the net loss is calculated which is the total error of the network for each pattern. This net loss is split into the hidden layer connections in proportions of the weights that these connections have been assigned. This can either be done by manual calculation or by accessing the gradient of the weight parameter. The split error is stored in a one dimensional Tensor to be added to the next batch of split errors from the next pattern. All these error components are accumulated while iterating through the the patterns. Once this is finished, the node-wise stored error components go through an abs() function to make sure we're measuring the badness regardless of the sign. The maximum of these values is the node with the maximum accumulated error.

Although there are many ways to eliminate nodes, for simplicity's sake in our study we pruned by setting all connection weights of the desired node to zero. After this, the requires_grad variable was set to False, making sure that the node weights would undergo no further changes.

2.5 Deep Neural Network

The simple two layer network had a blind spot when training with instances of the second and third categories. (This can be seen in the confusion matrices in the Results section). Which is why, this part of the study, it was expected that a deeper network with fewer nodes in each layer (and roughly the same number of hidden nodes in total) would perform better classification, especially for instances which belong to the second and third categories. The structure was built and connected using PyTorch's built-in torch.nn.Module. However, when feeding the data into this network, some issues were encountered in the feed-forward function of this structure, which could not be resolved in the given time-frame.

3 Results and discussion

The In the training set, the network showed an increasing accuracy over 5000 epochs (increasing beyond 99% in the end). In the testing set, an accuracy of between 86 and 90 percent was observed.

Having known the skew in the target data, it's very much possible that the high accuracy was a result of a majority of data belonging to one class. In order to test for this we used confusion matrices instead.

Table 1: Testing and training matrices

Confusion matrix for training:	Confusion matrix for testing:
549 0 1 5 33 0 1 0 10	124 3 0 8 1 1 2 1 1

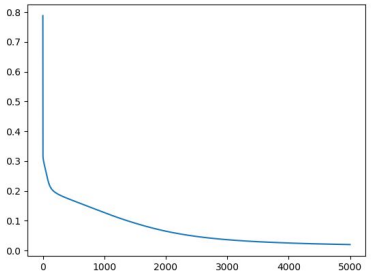
Table 2: Testing matrices before and after pruning

Before: Testing Accuracy: 90.14 % Confusion matrix for testing:	**After:** Testing Accuracy: 91.55 % Confusion matrix for testing:
124 5 1 6 3 1 1 0 1	127 3 0 7 2 1 1 0 1

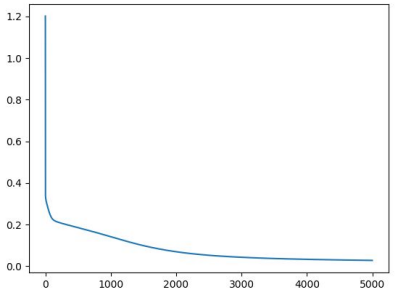
In case of pruning, the results were mixed. Out of 10 runs, 6 yielded marginally increased accuracy. In 3, accuracy was lost, and in 1, there was no change. This is not too surprising given that badness is not the most comprehensive metric for removing excessive nodes. To remove the node with maximum errors also means to lose on a huge chunk of the learning that the system has done so far.

4 Conclusion and future work

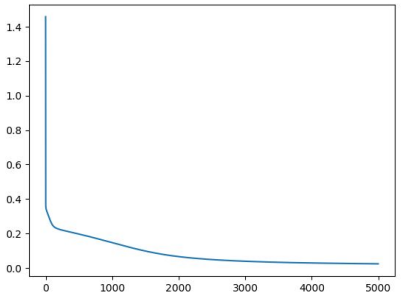
The performance of the system can be improved by tinkering with the number of hidden neurons, number of epochs and learning rate. But even despite playing with these variables, the system doesn't do very well in classifying things into the third category. This could be remedied by working on the categories themselves. Oversampling is a potential solution. Another way would be to set categories in a way that facilitates better distribution. The following graphs show plots of loss vs epochs for varying number of nodes in the hidden layer.



Number of hidden neurons: 24



Number of hidden neurons: 20



Number of hidden neurons: 28

The performance of the system can also be improved by pruning neurons that are not helpful. Although Badness didn't show reliable improvements, other criterion like Relevance and Distinctiveness might be more helpful. Better solutions can be sought in future work by expanding upon this network using evolutionary and genetic algorithms. The comparison of the simple classifying network, the network after pruning and the network aided by genetic algorithms may yield interesting results.

Some issues could also have been resolved by finer preprocessing. Perhaps some attributes could have been encoded in a manner that ensures faster convergence and fewer mistakes. Perhaps, if the deep network was successfully run, the results may have been better than what the simple network yielded with/without pruning. This possibility serves as a good avenue for future work.

A research paper that features the same dataset [2] uses a more complex network, with two hidden layers. It also has reduces the number of attributes to 17 in the preprocessing phase, using The Rough Sets theory. There also seem to be more instances in the dataset that the paper uses. This could have been achieved using some sampling methods. The results of that paper cannot be compared to this work easily as the authors predicted absenteeism instead of categorising it into classes. But the graph of their network output maps pretty closely to the desired output for a vast majority of instances. This indicates that our work can improve from looking into other types of (deeper?) network structures and optimisation methods. If done well, a system like ours could help predict absenteeism in workplaces and contribute to improving productivity for individuals and organisations alike.

5 References

[1] Hagiwara, Masafumi. "Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection." *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*. IEEE, 1990.

[2] Ricardo Pinto Ferreira et al.2018, Artificial Neural Network And Their Application In The Prediction of Absenteeism At Work. Int J Recent Sci Res. 9(1), pp. 23332-23334.