

# Accelerating Artificial Neural Network Learning via Weight Predictions

Chris TANNER

Florida Institute of Technology

Melbourne, FL 32901

ctanner@fit.edu

## Abstract

In this paper, I investigate the famous, generic *BackPropagation* algorithm that is used for Artificial Neural Networks, in hopes of improving how it learns weights. Moreover, I explore a technique for learning weights faster, which I call *3BoxPrediction*. I assert that if the *BackPropagation* algorithm learns the training examples well, then the weights of the network will typically develop in a relatively well-behaved, stable path. Furthermore, at a point during learning, I attempt to predict each weight by jumping to a value to which the path seems likely to converge. Consequently, when this predicted weight value is accurate, the remaining learning only further updates the weights—therefore imitating what we would have achieved had we allowed more learning to occur. This paper discusses the obtained results and mentions the limitations and weaknesses of my proposed technique for accelerated learning.

## 1 Introduction

**Motivation:** Artificial Neural Network learning algorithms have been highly successful in learning even complex, real-world tasks, provided they are given a good, representative target function that directly relates to the learning task. The most famous of these algorithms is likely *BackPropagation*. *BackPropagation* is used on feed-forward, multi-layer neural networks—the network contains an input layer, hidden layer(s), and an output layer. The input layer has units with values based on our training data. The output layer has units that correspond to the output/prediction of each training example. The network is fully connected in a forward manner such that each unit has a weighted link to each unit in the next layer. The algorithm learns by computing errors from the output layer and appropriately working backwards to the input layer, while adjusting each weight between units. The user specifies a desired learning rate from 0 to 1, whereas the value has a direct proportionality with the degree by

which each weight should be changed. A suitable learning rate allows the weights to collaboratively reach a balance such that the error in its predicted outcome values has a minimized error in respect to the actual target values that are represented in the training examples. Typically, as the learning progresses, the weight values form paths that are well-behaved and converge near their end [2]. If one could predict the values to which the weights will converge, then the system may accelerate the learning process.

## 2 Problem

As mentioned, in order to learn well, the *Back-Propagation* algorithm requires the user to specify an appropriate learning rate and stopping criterion. In our case, the stopping criterion is the number of times to iterate through the entire training data set. Notably, the learning rate has a drastic effect on the algorithm's ability to learn because if the rate is too low, it may update the weights too minimally and thus never learn well before the stopping criterion is met. Moreover, a low learning rate makes the weights vulnerable to getting stuck in local minimums and maximums, thus preventing the weights from reaching their optimum values. Conversely, a learning rate too high may not permit the weight values to satisfactorily converge—as it will oscillate and overstep the optimum values. My proposed attempt to predict the weight values hopes not to only overcome the weakness of sensitive learning rate, but to also accelerate the learning.

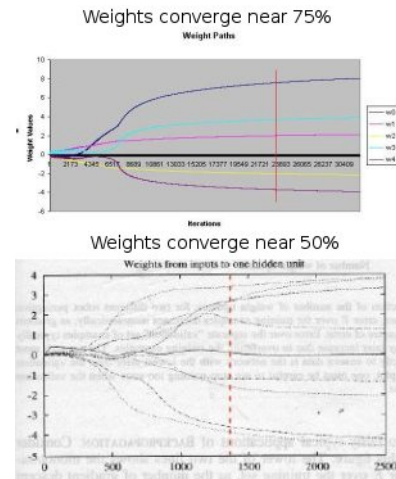


Figure 1: Weights often converge between 50% and 75%

## 3 Approach

Hoping that the learning weights eventually converge, we attempt to guess future weight values at a particular time during its learning. From my own tests and viewing others' results (Figure 1), well-learned data produces weights that start to converge between 50% and 75% of the learning time [3]. Therefore, we wish to analyze the weight values up until the point at which we make our prediction, which will occur sometime during this 50% to 75% range of training time. Implemented is a sigmoid function that determines when we will make our prediction:

$$\# \text{ to analyze} = \frac{1}{1 + e^{-\text{learning rate}}}$$

The value of the learning rate is directly proportional to when the algorithm will make its prediction; a lower learning rate will yield a prediction sooner than a higher learning rate will.

This accommodates the possibility of bad predictions, for lower learning rates will permit the weights to have time to eventually grow to desirable values.

*3BoxPrediction* now knows how many points to analyze before it predicts a value for each path. As for making a prediction, we must somehow know the behavior of the path—is the path starting to converge, diverge, or oscillate? Moreover, we need some measure as to how certain we are of our prediction, which should directly relate to how much “in the future” we are trying to approximate. One elementary way to model the path is to segment the path into regions, or “boxes.” The analyzed path is to be segmented into three evenly sized clusters. The amplitude of the last “box” will be compared in respect to the one-third of the total amplitude of the entire path that is being analyze. This simple approach provides a good idea as to the recent behavior of the path; if the amplitude of the last box is greater than one-third of the amplitude of the entire path, then the path is starting to change more than its average amount. Similarly, if the amplitude of the last box is less than one-third of the amplitude of the entire path, then the path has diminishing behavior and is hopefully converging. This ratio,  $\theta$ , is as follows:

$$\theta = \frac{\text{amplitude of } 3^{\text{rd}} \text{ box}}{\text{amplitude of entire path}/3}$$

$\theta$  yields us with a suggestion as to how severe the prediction should be—a larger ratio correlates to a prediction of larger magnitude, for the path is greatly changing. These variations can be seen in Figure 2.

This ratio  $\theta$ , however, lends itself vulnerable to making an unsafe guess, for it may suggest

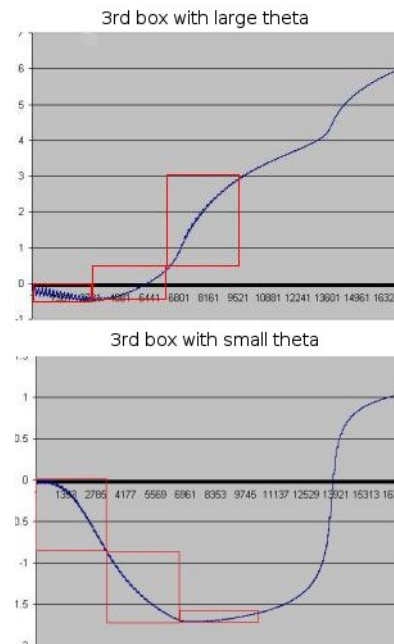


Figure 2: Varying Theta Values

making a nearly infinitely large guess if nearly all of the weight path’s movement occurred in the last box. Therefore, we “squash” this value via the already-used sigmoid function:

$$\Delta = \frac{1}{1 + e^{-\theta}}$$

This value  $\Delta$  is multiplied by the total amplitude of the path, which now provides us an actual distance of our prediction. For example, if the path has continued to grow linearly with time,  $\theta$  will equal 1 and  $\Delta$  would consequently have a value of .75. Therefore, the magnitude of our guess would be .75 of the amplitude of our entire encountered path. Additionally, we multiply this magnitude by the learning rate for the sake of taking into consideration the ability to save itself if a bad prediction was made. If a small learning

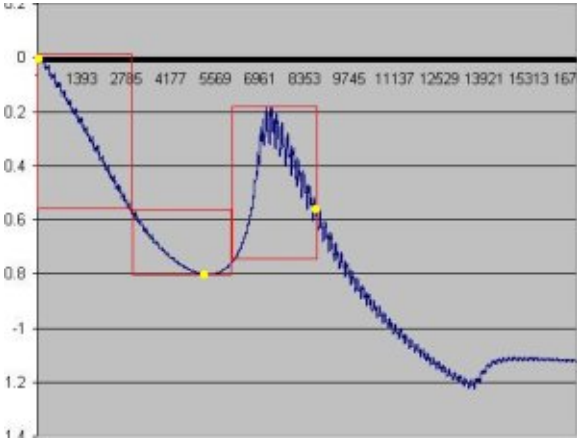


Figure 3: Oscillations affect the magnitude of a prediction

rate was specified, the prediction should be reserved enough so as not to make a prediction as large as that when having a large learning rate.

$$\omega = \Delta * A * \eta \text{ (where } A = \text{total Amplitude)}$$

Additionally, the prediction should take into consideration oscillations. Therefore, we observe the location of the last weight value in respect to the total amplitude. This provides us with a confidence  $\alpha$  of our prediction:

$$\alpha = \frac{\text{value of end point} - \text{origin point}}{A}$$

For example, the path in Figure 3 has oscillated back toward its initial value. Thus, despite its large  $\theta$  value, the prediction should not have such a large prediction because the path's oscillation suggests uncertainty.

This yields us with our final equation for predicting a given weight  $w_x$ :

$$w_x = \omega\alpha$$

Now that we have a magnitude that represents how far from the end point our prediction should be, we need to know in which direction to predict (upwards or downwards from the end point). Merely looking at the error from the unit to which the current link is forward connected would force many weights to predict values in the wrong direction. As a result, we look at each individual weight's path. Since each path was already segmented into thirds, we cheaply compare the average values of the 2<sup>nd</sup> and 3<sup>rd</sup> box as a way of deciding if the path is generally heading downwards or upwards. If the 3<sup>rd</sup> box has a higher average of weights than the 2<sup>nd</sup> box, then our path is likely heading in the upward direction. Similarly, if the 3<sup>rd</sup> box has a lower average than the 2<sup>nd</sup> box, then our path is likely heading downwards. This approach seems more insightful than simply looking at the last few values of the weight's path.

## 4 Empirical Evaluation

### 4.1 Evaluation Criteria

The goal of the devised *3BoxPrediction* algorithm was to improve *BackPropagation* via requiring less training iterations in order to achieve at least comparable results. Moreover, if training both algorithms for the same period of time, *3BoxPrediction* should have the higher accuracy of classification during testing. For that reason, I evaluated the algorithms based on these two aspects: the required number of iterations to achieve a given testing accuracy level at least 80% of the time, and the average testing accuracy level for a set number of iterations.

## 4.2 Experimental Data and Procedures

Six data sets were used for training and testing, for the sake of seeing the aforementioned classification results. The data sets used had no missing attributes, contained only discrete values for attributes, and were gathered from [1]. The summary of the testing data is found in Figure 4. As for evaluating the results, one must realize that each time the neural network is trained, it is subject to the variance of the initial random weights. Thus, multiple training and testing runs must be performed for the sake of getting a better average of the overall performance of each algorithm. As for obtaining the classification accuracy level for a given data set, each algorithm was trained and tested 100 times. Each training instance used a learning rate of .3. Each of the ‘monks’ data sets was trained with 50 iterations. The ‘lenses’ data set was trained with 700 iterations. The ‘car’ data set was trained with only 10 iterations. The average accuracy level for each algorithm, per data set, was then reported. As for determining the required number of iterations to reach a desired accuracy level, each algorithm was trained and tested until at least 40 of 50 runs produced 90% accuracy (learning rate was set to .3).

## 4.3 Results and Analysis

The extensive testing on the chosen six data set illustrated that the algorithm made good predictions (see Figure 5), yet it is disappointing in that no significant improvements appeared. Figure 6 shows the complete results. It should be noticed that although there is no noticeable im-

name	# of cases	# attributes	# classifications
monks1	432	6	2
monks2	432	6	2
monks3	432	6	2
car	1728	6	4
lenses	24	4	3
votes	435	16	2

Figure 4: Training Data

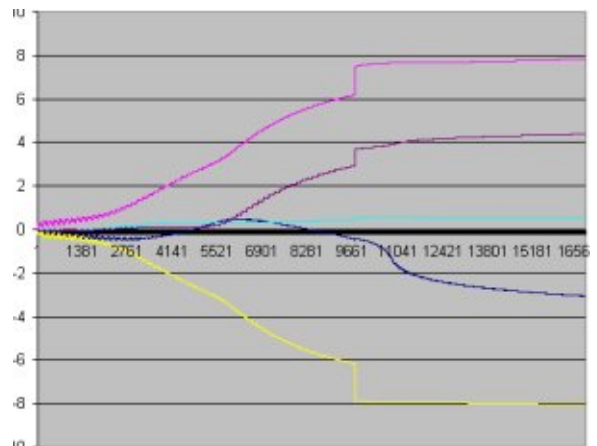


Figure 5: Predicted Weight Values

provement from *BackPropagation*, the results suggest that *3BoxPrediction* also seems not to be much worse. I believe that the collection of ‘monks’ data is very similar, and thus provides little information about the algorithms. Notably, *3BoxPrediction* seems almost identical to *BackPropagation*, for their values are relatively the same and neither appears to be strongly superior. The ‘lenses’ data suggest that *BackPropagation* is superior, as it has an overall higher classification accuracy and requires less iterations in order to achieve a 90% accuracy level. Lastly, *BackPropagation* appears superior again according to the ‘car’ data, as it also has higher classification accuracy and comparable accuracy with fewer required iterations. I believe these less-than-desired results are justified because making a weight prediction, even if it is good, does not entirely imply that the overall system has well-learned the training data. I had already considered this idea, and I had accepted the idea that the strongest factor for learning well is how the weights grow *together*. Yet, I believed that if good weight predictions were made, then it would take a few iterations for the system to find optimum weight values; thus, I thought it would be possible to easily surpass the accuracy results that the original algorithm yielded. Another possibility is that *3BoxPrediction* occasionally overfits the data. Regardless, *3BoxPrediction* in general appears to be an elementary, and possibly unorthodoxed, method for trying to accelerate learning weights of a neural network.

Training	Algorithm	Req. # of Iterations	Avg. Accuracy
monks1	backProp.	70	85.0276
monks1	3BoxPred.	70	84.5492
monks2	backProp.	100	77.2606
monks2	3BoxPred.	110	73.1336
monks3	backProp.	40	94.3867
monks3	3BoxPred.	30	94.3058
lenses	backProp.	690	92.1639
lenses	3BoxPred.	740	88.8312
car	backProp.	150	79.1060
car	3BoxPred.	170	70.0299

Figure 6: Results

## 5 Conclusion

### 5.1 Summary of Findings

Overall, the results from *3BoxPrediction* were somewhat disappointing in that although the weight predictions appeared relatively accurate, the algorithm overall seemed to be slightly worse than the original *BackPropagation*. Furthermore, I concluded that predicting future weight values at one given point during the training time is an elementary and probably unorthodoxed method for accelerating learning. I hypothesize that the only time at which *3BoxPrediction* is superior to *BackPropagation* is shortly after the iteration when *3BoxPrediction* makes its prediction; as the iterations progress toward the stopping criterion, both algorithms converge near similar points. However, *3BoxPrediction* makes a prediction that should imitate what we would have achieved had training been normally carried out to completion. Because the predic-

tions are not 100% accurate, I believe it takes some time for the weights to find their global good niche of stability. Another possibility for having worse results is that it may be subject to overfitting.

## 5.2 Limitations and Possible Improvements

*3BoxPrediction* is limited in that its magnitude of prediction is reliant on the behavior of the weight paths; if the weights grow highly chaotically, then the prediction will be very little magnitude and thus be of little value. In other words, its usefulness is generally directly related to how to stable a weight path is, yet if a weight path is highly stable, then it likely would have converged nicely had the original algorithm been used. A possible improvement is if the predictions or weight adjustments were of a more continuous nature, rather than merely making one hopefully good prediction. This seemingly more orthodox method would hopefully allow the weights to increasingly minimize error and become a well-learned system faster than the traditional *BackPropagation* algorithm. In summary, this devised algorithm appears limiting and to be of no novel basis.

[3] Mitchell. *Machine Learning*. McGraw-Hill, 1997.

## References

- [1] C.L. Blake and C.J. Merz. Uci - repository of machine learning databases, 1998.
- [2] Ham. *Principles of Neurocomputing for Science and Engineering*. McGraw-Hill, 2001.