Understanding & Developing Artificial Neural Networks

Deerfield Academy Advance Computer Science Research

Yongyang (Neil) Nie

Feb, 2016

Contents

1 Abstract:

Machine learning is a subset of artificial intelligence [1] that provides computers with the ability to learn without being explicitly programmed. [2] Machine learning focuses on the development of computer programs that can change when exposed to new data.

There are many methods and algorithms for machine learning, from Support Vector Machine[3] to Artificial Neural Networks[4]. One of the common purpose for learning algorithms is to classify complex data. For example, It can be used in analyzing human genetics and genomics. [5]

2 Introduction:

There are two types of machine learning, one is supervised machine learning[6]. Supervised learning is the machine learning task of inferring a function from labeled training data. Each example is a pair consisting of an input object and a desired output value. By doing some calculation, the network can make rudimentary predictions and correct itself based on the training data to make more desired prediction. In this research, we will mainly focus on artificial neural networks and supervised machine learning.

This paper will discuss the principles behind designing, building, and debugging an artificial neural network. All of the data are collect from a project I built that can recognize handwritten digits. The projects is written mainly in Objective-C and Python. You can find the resources on Github.

2.1 What's neural networks

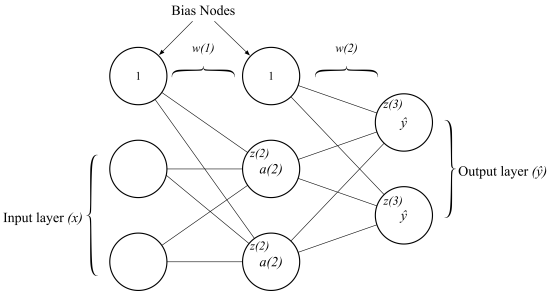


Figure 1

In the diagram above, you can find the important components in a neural network. First of all, most neural networks have a input and a output layer. Many neural networks will have hidden layer(s), when there numerous hidden layers, we consider that as deep learning [7], which is out of our scope.

Secondly for neural networks in this paper, there are synapses connecting every node to every node in the next layer. All synapses have weights, a floating point number that will govern how the network behaves. Since the input is often static, the commonly used way to improve the prediction of a network is to modify the weights on synapses.

Thirdly, there is one bias node connected to every layer in the network except the first layer. The bias will horizontally shift the activation function. This behavior will likely help the network learn more accurately. [8]

2.2 Why neural networks

According to Christos Stergiou and Dimitrios Siganos' Neural Networks[9] "Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in

[1] Artificial Intelligence A Modern Approach http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.8854&rep=rep1&type=pdf

[2] Samuel, Arthur L. (1959). "Some studies in machine learning using the game of checkers". IBM Journal of research and development.

[3] https://en.wikipedia.org/wiki/Support_vector_machine

[4] https://en.wikipedia.org/wiki/Artificial_neural_network

[5] Machine learning applications in genetics and genomics, Libbrecht, Maxwell W. http://dx.doi.org/10.1038/nrg3920

[6] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) *Foundations of Machine Learning*, The MIT Press ISBN 9780262018258.

[7] Deep learning Yann LeCun, Yoshua Bengio & Geoffrey Hinton http://dx.doi.org/10.1038/nature14539

[8] Role of Bias in Neural networks http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

[9] This section is reference from NEURAL NETWORKS by Christos Stergiou and Dimitrios Siganos https://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol4/cs11/report.html#Why%20use%20neural%20networks

the category of information it has been given to analyze."

3 Forward Feed:

The neural network will begin be taking in some inputs and making some predictions based on the inputs and the weights between nodes. We can treat the inputs and weights as matrices. This process can be seen as consecutive matrices multiplications. The matrix calculation should yield us some result

$$\left[input\right]\ \left[weights\right] = \left[output\right]$$

If we give the matrices some names, call inputs x, weights $w_{(l)}$ and product of the matrix multiplication $z_{(l)}$. In our case, l indicate the layer, for example, the first layer weights are $w_{(1)}$. $z_{(1)}$ represent the output matrix with layer *1*.

$$\mathrm{x}w_{(n)} = \ z_{(n)} \tag{1}$$

Afterwards, we have to apply an activation function[10]. The activation function that I used in this research and this paper will be the sigmoid function.

$$\sigma\left(z\right) = \ \frac{1}{1+e^{-z}}$$

$$\mathrm{a}_{(l)} = \sigma(z_{(l)}) \tag{2}$$

Multiplying $a_{previous}$ with connection weights, then applying activation function is one complete cycle. In a multilayer neural network, this process will be repeated until we reach the output layer. In a simple three layer neural network, this process only have to execute twice.

4 Quantifying and Minimizing Cost:

Now the neural network can make calculation/predictions, however, the result is far from desired. In almost all learning algorithms, the input data cannot be altered, therefore the x term is constant in equation one. In order to change the output z the only option is to change the weights w.

First of all, we have to come up with ways to quantify the cost.

$$(3)\ \ C = \sum_{j} \frac{1}{2}(\hat{y} - y)^2 \tag{3}$$

C is the cost, which equals to the sum of all the differences between calculated result and actual result squared and times one half.

In a simple three layer neural network, we can take advantage of the equation derived above and substitute for some of the variable.

$$\hat{y} = f\left(f\left(xw_{(1)}\right)w_{(2)}\right)$$

$$C = \ \sum_{J} \frac{1}{2}\left(y - f\left(f\left(xw_{(1)}\right)w_{(2)}\right)\right)^2 \tag{4}$$

t

Here we have it, a way to quantify the cost of the neural network. This function will be referred to as the **cost function**. Now, we have to solve the problem of minimizing C. Will brute force work? It turns out, no, because in a three-node neural network, there are millions of possible weights that we have to compute. But there are better solutions.

We can think of the equation above as a function of cost in terms of all possible weights. There will be one set of weights that will bring the cost to the lowest. Then, this becomes a minimization problem.

5 Gradient Descent:

The best way to minimization the cost is to use gradient descent, a very fast and classic way to solve problems like this. In fact, gradient descent is widely used in math, image process and machine learning.

Gradient descent is "a first-order iterative optimization algorithm". To find a local/-global minimum of a function using gradient descent, we have to take steps proportional to the gradient (or of the approximate gradient) of the function at the current point.

The starting point of gradient descent in our case is random, since the initial set of weights are randomly generated. Then, the algorithm can take steps toward the minimum The process can be seen as a ball rolling down a hill[11] and trying to find the lowest point. Note that actual physics doesn't apply here and we will define our own movement of the ball.

[10] https://en.wikipedia.org/wiki/Activation_function

[11] https://iamtrask.github.io/2015/07/27/python-network-part2/ by Andrew

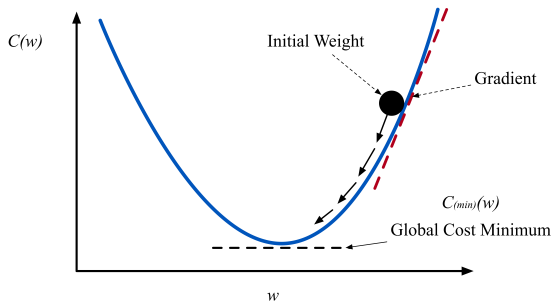


Figure 2

There are limitations to this method. First of all, what if we are stuck in a local minimum? Our goal is to find the global minimum for the cost function. In another word, this method will not work properly for a non-convex function.

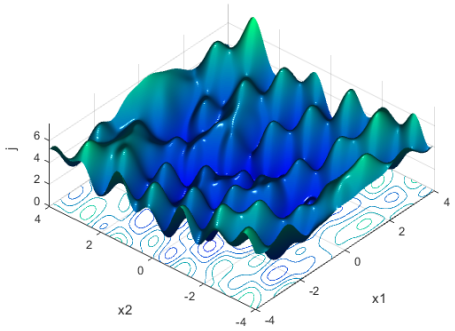


Figure 3

In fact, this problem is solved in equation (3), by squaring the difference in $(\hat{y} - y)^2$, we are using the quadratic cost function, which is a convex function for any number of dimensions.

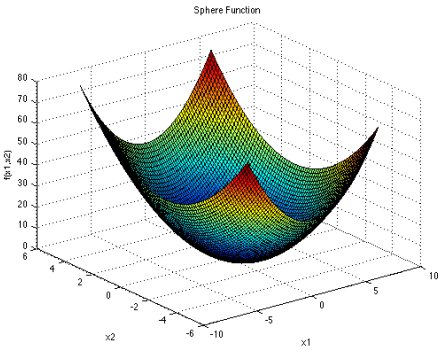


Figure 4

Thus, we can apply gradient descent without worrying about local minimums [12].

There are other types and variation of gradient descent as well. One of the most commonly used one is Stochastic gradient descent (SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration. [13]

6 Back propagation:

With gradient descent, we can create a set of routines that can help us to change the weights of the network to minimize the cost. This is call backprogation, which is the core of most of the sophisticated learning algorithms.

6.1 Back propagation overview

Phase 1: Propagation: Each propagation includes forward and backward propagation: [14]

1. Forward propagation of a training pattern's input through the neural network in order to generate the network's output value(s).

2. Backward propagation of the propagation's output activations through the neural network using the target result in order to calculate the deltas (the difference between targeted and actual outputs) of all output and hidden neurons.

Phase 2: Weight update: For each weight:

1. The weight's output delta and input activation are multiplied to find the gradient of the weight.

2. A ratio (percentage) of the weight's gradient is subtracted from the weight.

6.2 Mathematics behind back-propagation

Backpropagation is based around four fundamental equations. Together, those equations give us a way of computing both the error $\delta^L$and the

[12] Proof and definition of convex functions: http://mathworld.wolfram.com/ConvexFunction.html

[13] http://www.mit.edu/~dimitrib/Incremental_Survey_LIDS.pdf Dimitri P. Bertsekas Report LIDS - 2848

[14] A Gentle Introduction to Backpropagation - An intuitive tutorial by Shashi Sathyanarayana The article contains pseudocode ("Training Wheels for Training Neural Networks") for implementing the algorithm.

gradient cost of the function. [15] I didn't create these equations, in fact, a machine learning expert explained these commonly accepted equations in his book – Neural Networks and Deep Learning.

6.2.1 An equation for the error in the output layer δ^L

$$\delta_J^L = \frac{\partial C}{\partial a_J^L}\sigma'(z_J^L) \qquad \text{(BP1)}$$

The first term on the right, $\frac{\partial J}{\partial a_J^L}$ measures how fast the cost is changing as a function of j$^{\text{th}}$ output activation. In simpler terms, this term calculates the derivative of C (cost) with respect to a (equation 2) at layer J. The terms in (BP1) are easily calculated. We computed z_J^L during forward feeding. Depending on the cost function, in our case, the quadratic cost function is relatively easy to compute.

Equation (BP1) is a good expression, however, it's not matrix based, form that backpropagation desires. But, we can transform it into a matrix form, which becomes:

$$\delta_J^L = (a_{(l)} - y)(\sigma'(z_{(l)}) \qquad (5)$$

6.2.2 An equation for the error δ^l in terms of the error in the next layer, δ^{l+1} in particular:

$$\delta^l = ((w_{(l+1)})^T \delta_{(l+1)}) \odot \sigma'(z_{(l)}) \qquad \text{(BP2)}$$

In the equation above, $(w_{(l+1)})^T$ is the transpose of the weight matrix $(w_{(l+1)})$ for the *(l+1)th* layer. When the transpose weight matrix is applied, this can be seem as moving the error backward through the network, giving us some sort of measure of the error at the output of the l$^{\text{th}}$ layer. Finally, we take the Hadamard product [16] $\odot\sigma'(z^l)$.

Equation (5) above can help us calculate (BP2), which is matrix based. Hadamard product can be understood as:

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \odot \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_1 * a_2 & b_1 * b_2 \\ c_1 * c_2 & d_1 * d_2 \end{bmatrix}$$

The incredible power of the equation above becomes apparent when (BP1) and (BP2) are working together. We start by using (BP1) to compute δ^l, then apply Equation (BP2) to compute δ^{L-1}, then Equation (BP2) again to compute δ^{L-2}, and so on, all the way back through the network. In the subsection below, there will be an example of using (BP1) with (BP2) in a simple three layer neural network.

6.2.3 An equation for the rate of change of the cost with respect to any bias in the network

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \text{[17]} \qquad \text{(BP3)}$$

This is the error δ_j^l is exactly equal to the rate of change $\frac{\partial C}{\partial b_j^l}$. This equation can be rewritten as

$$\frac{\partial C}{\partial b} = \delta$$

Bias nodes are not necessary in neural networks. However, they are helpful in improving the network's learning. In a linear function, $y = mx + b$, the b term will effect the vertical shift of the function. Bias nodes for neural networks behaves similarly. It can translate the sigmoid function on the coordinate plane. [18]

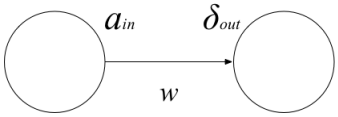
6.2.4 An equation for the rate of change of the cost with respect to any weights in the network

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad \text{(BP4)}$$

This equation can help us to compute the partial derivative $\frac{\partial C}{\partial w_{jk}^l}$ in terms of the quantity δ^l and a^{l-1}. It can be rewritten in a less index-heavy notation:

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \qquad (6)$$

In equation (6), a_{in} is the output with weights w. And $delta_{out}$ is the error of the neural output from the weight w.



[15] The equations were created by Michael A. Neilson "Neural Networks and Deep Learning", Determination Press, 2015

[16] https://en.wikipedia.org/wiki/Hadamard_product_(matrices)

[17] The equation was reference from Michael A. Neilson "Neural Networks and Deep Learning", Determination Press, 2015

[18] Make Your Own Neural Network – Tariq Rashid

Equation (6) is another critical component of backpropagation. Now, we know exactly how much to change the cost regarding the weights. If we apply the gradient to current weights, then, cost of the network should decrease, and we are closer to the global minimum of the cost function.

6.3 Backpropagation with simple neural network example

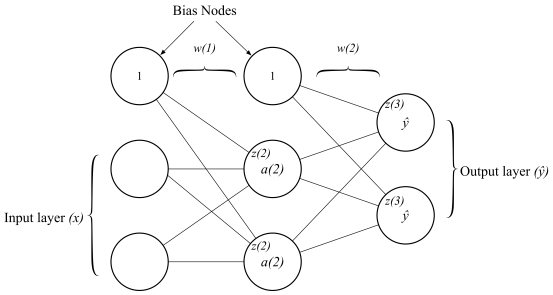


Figure 5

In a simple three-layer neural network, below are the four equations that we derive with the principle of gradient descend that will help us minimize the error.[19]

$$\delta_{(3)} = -(y - \hat{y})(\sigma'(z_{(3)}))$$

Calculate the δ for the third layer of the network. This looks quite similar to Equation (6).

$$\frac{\partial C}{\partial w_{(2)}} = (a_{(2)})^T \delta_{(3)}$$

The equation yields the partial derivative of C in terms of second layer weights $w_{(2)}$, which connect the hidden layer to the output layer. Transposing the hidden layer output, $a_{(2)}$ and multiplying $\delta_{(3)}$ we can calculate the rate of change, which will be applied to $w_{(2)}$

$$\delta_{(2)} = \delta_{(3)} \left(w_{(2)}\right)^T \sigma'(z_{(2)})$$

Calculate the δ for the second layer of the network.

$$\frac{\partial C}{\partial w_{(1)}} = x^T \delta_{(2)}$$

Finally, we can modify the first layer of weights in the network. This process is often repeated until the accuracy of the network reaches a threshold. The backpropagation process above can be represented with this simple diagram of a neural network with one hidden layer.

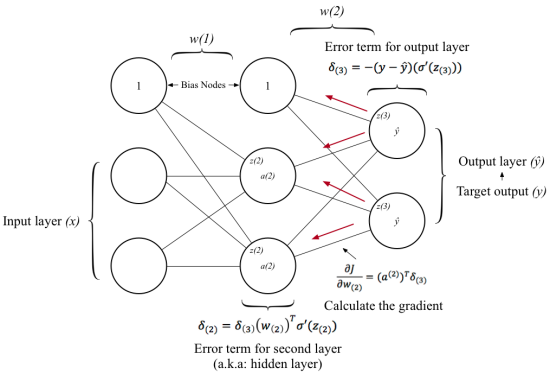


Figure 6

Here is a visualization of the change in weight effect the cost of output nodes. Line 0-3 are lowering as we train the network. This figure displayed that the error of the network C is being minimized.

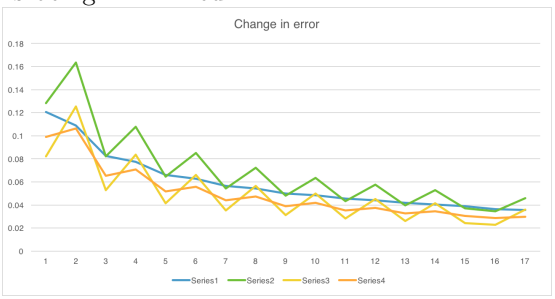


Figure 7

Here is a graph of the correction of the neural network over times of training. Each time the network will train with 1000 out of 60,000 training data. After, The program will shuffle the training data and repeat the training process for 197 times, or until the network reaches 95% accuracy.

[19]The equation was referenced from Michael A. Neilson "Neural Networks and Deep Learning", Determination Press, 2015

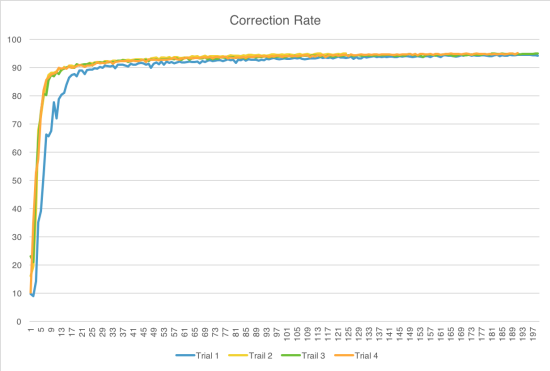


Figure 8

The neural network is trained four times with the same training and testing data. The results are similar and the network is steadily improving, thanks to gradient descent. There are many factors that contribute to a successful learning process. If these things are ignored or incorrect, the network will not be able to learn as well.

7 Improve neural network training result

Not all neural network behaves flawlessly. Generally, there are a few factors that will effect the training result of the network: hyperparameter, number of hidden nodes, human error, training data/testing data problem and more. Among them, hyperparameter, number of hidden nodes and be determined by estimation and trial and error. Huamn error can be eliminated by debugging. Training data and testing data problem is not easy to solve. That often contributes to the source of your data and the quality of your data, which are beyond the scope of this research. It's safe to assume that the training data and testing data that we are using are carefully chosen and considered.

7.1 Making good decision

In a pre-trained network, there are several critical hyperparameters and numbers that will help the network to improve. I mainly focused on learning rate and number of hidden nodes.

7.1.1 Hyperparameters

Learning rate is the hyperparameter that we will focus on. To make gradient descent work correctly, we need to choose the learning rate r to be small enough that Equation (9) is a good approximation. If we don't, we might end up with $\Delta C > 0$, which will take us to the opposite of minimizing. Meanwhile, r can't be too small, since that will make the changes Δv tiny, and thus the gradient descent algorithm will work very slowly. In practical implementations, r is often varied so that Equation (9) remains a good approximation, but the algorithm isn't too slow. [20]

This is a graph of the training result with a different learning rate. The batch size is 1000 and the training is repeated 100 times.

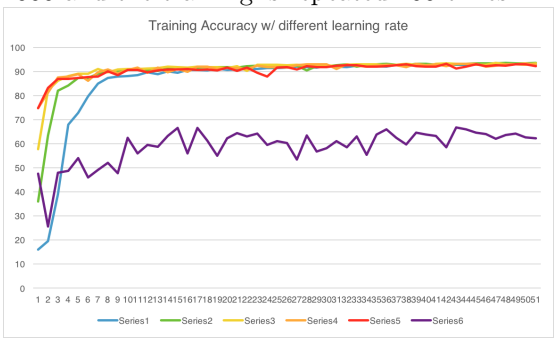


Figure 9

The range of the learning rate that I chose is from [0.1, 5]. When the learning rate is relatively low, the network began at a lower accuracy comparing to higher training rates. However, especially when the learning rate is 0.9, the training result became unstable, and the graph starts to oscillate.

On the other hand, when we set the training rate to 5.0, the network doesn't improve after 40%-50%. Then general rule of thumb for choosing the learning rate is somewhere between [0.1-1].

With this observation in mind, we might be able to alter the learning rate as the network progress. The training rate should we high in the beginning to quickly bring us to the desired place. Then, we can lower the rate so the network learning result will not have oscillation.

7.1.2 Hidden nodes

In a multilayer neural network, you will likely encounter the problem of how many hidden layers and hidden nodes should the network have. Seemingly simple, but complex question will help you improve the learning rate of the network. In

[20] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015. Chapter 1 Using neural nets to recognize handwritten digits

this research, we will only focus on one hidden layer. Multiple hidden layer is known as deep learning, which is out of our scope.

It's difficult to form a good network topology just from the number of inputs and outputs. It depends critically on the number of training examples and the complexity of the classification you are trying to learn. There are problems with one input and one output that require millions of hidden units, and problems with a million inputs and a million outputs that require only one hidden unit, or none at all.

Below if a graph of a neural network with different number of hidden nodes. There is a healthy range of number of hidden nodes. If there is one hidden node, the accuracy will not be more than 25% and also defeat the purpose of the neural network. If there are too many nodes, calculation becomes gruesome and the accuracy is similar to lower number of nodes. Therefore, the number of hidden nodes highly depend your data, network, situation and needs.

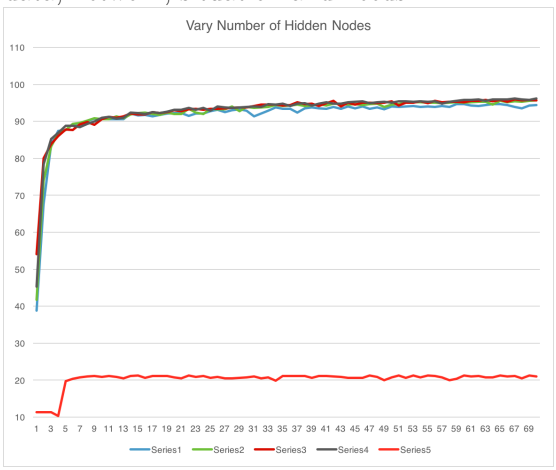


Figure 10

7.2 Improve training result with dynamic hyperparameter

In section 7.1.1, I stated that the learning rate will effect how the network learns in the beginning. Higher learning rate means that it will start at a higher success rate and gradually improve. However, high learning rate will lead to noticeable fluctuations in the learning rate. This observation gives us the possibility to change the learning rate as the network learns. In the beginning, we will set the learning rate 0.8, then slowly lower the the learning rate as the network improves.

This concept will seem intuitive in that you want the network slow down the learning rate when it's close to the desired result. Otherwise, a high learning rate will likely create some fluctuation like we have seen figure 2.

Below is a accuracy graph of a neural network with 0.8 learning rate. The network is trained with batches of 1000 data and shuffled before the next training. Training is stopped after the accuracy on test data reaches 95%. All of its hyparameters were static during training.

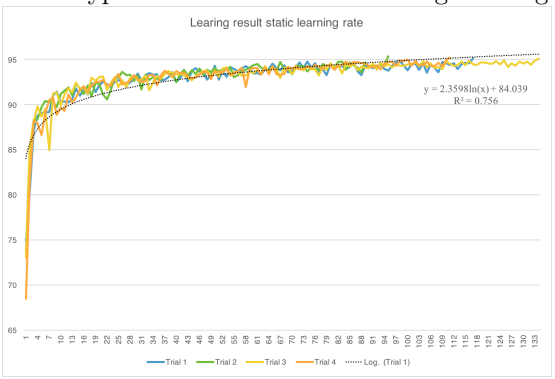


Figure 11

The training time is mediocre comparing to a well optimized network. The average training time is around 360 second and the number of epoch is around 110. In another word, the network took 6 minutes and 12,000 data to reach 95% accuracy.

We can improve this result by lowering the learning rate of the network as it improves. Specially, the network begins to lower the learning rate when the accuracy reaches 93%. The new accuracy is calculated by

$$r_{(new)} = r_{(old)} * 0.9 \tag{7}$$

Once again, the network is trained with batches of 1000 data and shuffled before the next training. Training is stopped after the accuracy on test data reaches 95%.

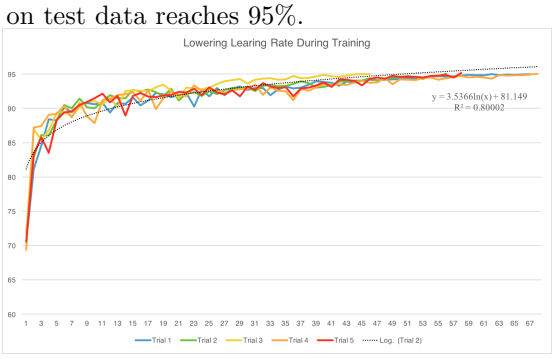


Figure 12

Clearly, the network took much less time and epoch to reach 95%.

Figure 13

The training time difference between dynamic and static learning rate is more than 300 seconds.

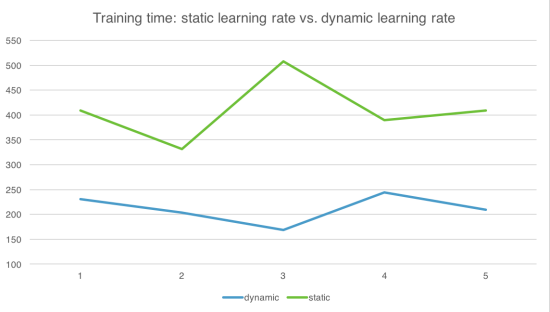


Figure 14

Comparing two results, we can calculate a LBF (line of best fit) for the accuracy rate. By comparing the two ln functions, the function in Figure (12) has a higher coefficient, which means that the growth of the curve will be greater. In another word, by adjusting the learning rate, the network was able to learn more quickly and efficiently.

8 Future studies

Finally, you have made it to the end. Thank you for staying with me throughout the paper. I hope you find neural networks amazing. It will bring tremendous change and power to computing technologies. The current computing power is far from simulating a biological brain. However, our understanding of it has helped us to develop technologies such as neural network and deep learning.

Specifically, in this research, I only touched the surface of neural networks. The field is enormous and requires all types of scientist to improve it. I avoided topics such as: different types of activation functions, some mathematical proves, details on improving neural network and finally, deep learning. Further development in the areas above will enhance current technologies and open doors to new opportunities.

In fact, many of my dedicated friends and colleges are working on these hard problems. Mr. Meng focused deeply on the mathematics behind neural networks. Mr. McAvoy developed other learning algorithms. In the larger realm, Google DeepMind is trying to create a comprehensive A.I. system; Google translate team made important breakthroughs in 2016 on their multilingual neural machine translation system.

Thank you again for taking your time to read this paper. I hope you are also excited to jump into the field and be apart of this great journey of studying, developing and exploring the world of machine learning, and eventually artificial intelligence.