Ben Hunt | Todd Tribble

Machine Learning 4378V

Digit and Symbol Recognition Using an Artificial Neural Network

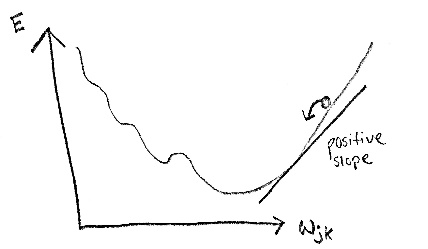
Digit recognition in the field of machine learning and specifically artificial neural networks is relatively common but acts as a great foundation for learning artificial neural networks from the ground up. To differentiate our neural network we sought to train the network on math symbols as well as the standard digits as to enable math equations to be read and solved. We used the MNIST dataset using different subsets of the original 60,000 digits as to speed up the runtime of our network. We performed various tests on the network and collected results to see the effects of parameters and various inputs such as the size of the dataset. Our neural network was programmed in Python using Jupyter and the Anaconda development suite. As a reference we read the book, “Make Your Own Neural Network”, by Tariq Rashid. The reference covers all aspects of the neural network as well as required perquisites such as linear algebra and calculus.

The goal of our network was to take in some raw input of image data and predict the expected output. To do this we needed some way of making a prediction given a high a dimensional input space such as an image. Images even scaled down can have hundreds of individual values known as pixels. We needed some way to handle such a high input space in that we needed to know how each pixel correlates to the output. With this, each one of these pixels may contribute to the image but not every pixel plays a part in the actual perception of the image. What differentiates one image from another is the features that it may contain, this is something we wanted our program to learn. For example, the core difference between a zero and the rest of the single digit numbers is that there is no black markings in the middle, it’s hollow. This is what we want our program to learn, but how does a program learn? This was the core of our problem and the centerpiece to what a neural network does, learn from what it is given, block out noise, and predict an output.

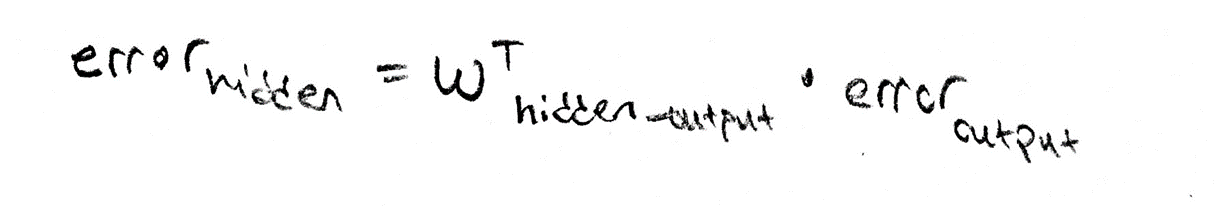
C:\Users\The Dough Boys\Desktop\MLProject\TestPlus\1.jpgAn artificial neural network contains multiple layers, the first of which is an input layer. Given that we want to predict handwritten digits and symbols, we found a dataset on handwritten digits. This dataset was the MNIST dataset that contains 60000 training examples and 10000 test examples. Unfortunately, we could not find a dataset for the math symbols so we created one using Microsoft paint and a drawing tablet. Our dataset consisted of 200 training examples of plus and minus signs. With these images, we can’t simply feed them into the input layer. Computers need to be able to read this data and make sense of it, thus it is necessary to preprocess the data. Given that our input space and number of images would already be considerably high we opted for a lower resolution set of images at 28x28 pixels. This would put less strain on our computer processors and lower the runtime of our program significantly. On the left is an example of a digit from the MNIST dataset and on the right is an example from our own dataset.

With these images, we now have 784 individual inputs, or pixels, that can be given to our network. This led to the creation of our 784 input nodes to our neural network. However, with gray scale values ranging from 0-255, we scaled these values down in the range from .01 to 1 to keep them around the same range as our activation functions.

With our inputs, we connect them to the next layer with the use of weights. These links are connected with each of the neurons in the next layer. The weights are to give the inputs a sense of significance for their contribution to the output of the activation function. Our weights were initialized from .01 – 1.0 with each link, with the expectation that they will dynamically change. Rather than randomly initializing them randomly however, we initialized the weights with a normal distribution using the numpy library to avoid zero weights and overly large weights. This is to avoid killing the signal as well as creating a gradient that’s flat so the learning ability of the network would be limited. The next layer from the inputs is the hidden layer which we chose to simply have 1, simply for a white box understanding of neural networks. Our hidden layer summed up the weighted input and applied a sigmoid function to it which would be the output to the next layer. In our case this is the final output layer which would again apply our activation function and make a final prediction. This prediction is determined to be wrong if it is less than the labeled correct answer, which we gave it to be 0.99. We determined that the correct answer should be 0.99, as the sigmoid function outputs values from 0 to 1. In addition we had a matrix of labeled outputs for every training example, with .01 being the expected on the respective incorrect nodes and 0.99 on the correct nodes.

 In receiving this output, we did not stop there as there is not much information to gain from the randomly initialized weights. Rather than feeding the network forward once we fed the network forward a minimum of n times, with n being the number of training examples. In addition we also iterated over the whole process, known as an epoch, various amounts of times. We did this to allow the neural network to learn the weights. Our neural network learned the weights by calculating the error based upon the error matrix. This error was fed backwards through the neural network by a process known as back propagation which updates the weights as well as the errors of each output from each layer. Gradient descent, in our case stochastic gradient descent, is at the core of this process in which it tries to find the correlation from each weight and the error and adjust the weight in a way that would minimize the error. It does this by taking the derivative of the error with respect to the weight in question (Wjk) and updates the weight according to the slope of that derivative.

The goal of the gradient descent is to converge at a minima, so if the slope is positive then the weight should be decreased and if the slope is negative then the slope should increase and hence the use of the negative sign in the equation above. This equation shows an update for weight a connecting node j to node k with j being the previous layer. Every hidden layer gets back propagated an error to calculate the updated weights for the links feeding into that layer. The error gets back propagated by the following formula which shows that each error from a hidden layer node can be contrived from all the outgoing link weights and their contribution to the error of the node they are linking too. One node’s error will take into account all the errors from the next layer.



Once the back propagation reaches the front of the neural network, i.e. there are no more weights to update, the next training example gets fed into the network and the process restarts. Repeating this process thousands of times results in weights that are gradually updated in such a way that minimizes the error in the prediction, and slope of the error with respect to the weights converge at a minima. However, this is not guaranteed and we chose a learning rate to update the weights with a value small enough, 0.3, not to overshoot the minima in most cases.

Now with a fully functioning neural network we tested the results of our neural network to see the effects of its parameters and its effectiveness to predict our digits and symbols. We tested the effect of the number of epochs, hidden layer nodes and different values for epochs on the accuracy of the network. In addition, we tested the accuracy of the network with the larger 60,000 digit set in one run with the 1000 digit subset in another. Due to processing constraints, we could not run the 60,000 digit set often. Figure 1 shows the results of training our network on the 60,000 digit set alongside our own 200 symbol set. The accuracy was not optimal but it was due to the (+) and (-) symbols having high variation in the way they were drawn. Input with such high variation should be done in much large quantity.

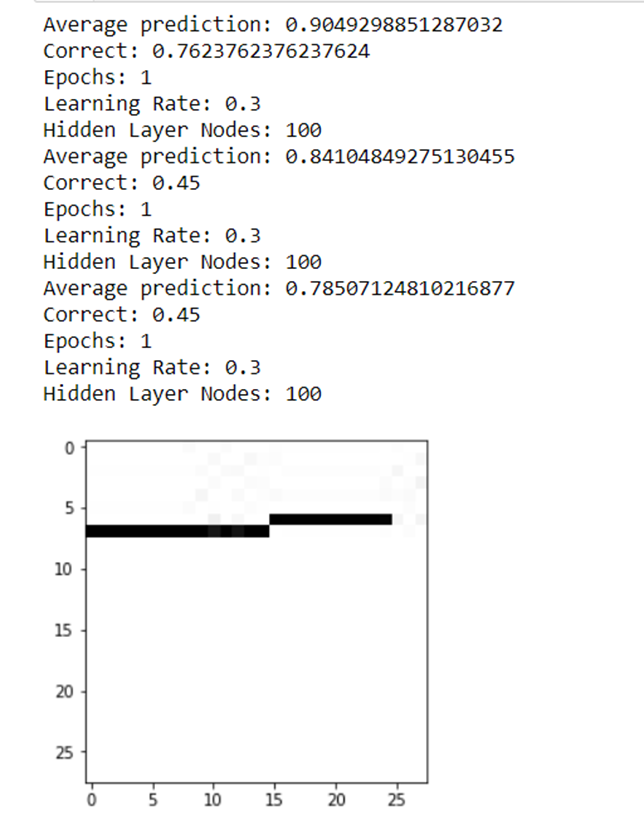


Figure 1

Once removing the symbol set from the network, our prediction of digits improved significantly, from 76% accuracy to 99% showing the effect of noise on the network.

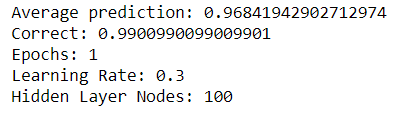


Figure 2

However, we did our following tests on the smaller 1,000 digit subset of the original 60,000 to improve runtime speed. In figure 3 we found the optimal hidden layers to be around 90 – 120, showing that too few and the network can learn any features. We didn’t draw any significant conclusions on why the accuracy began to fall when increasing the number of nodes, but it does hold that you should try to summarize the key features. My assumption would be that given such a high amount of hidden layer nodes, overfitting of the training data would occur. In figure 4, the importance of not overshooting the minima can be seen. At one end, the learning rate causes the descent to never converge and at the other end it over shoots the minima. A safe bet was to keep it around .2 to .5

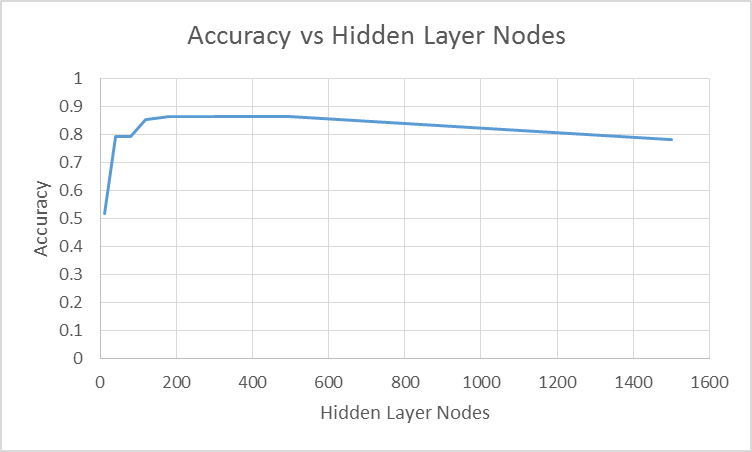


Figure 3

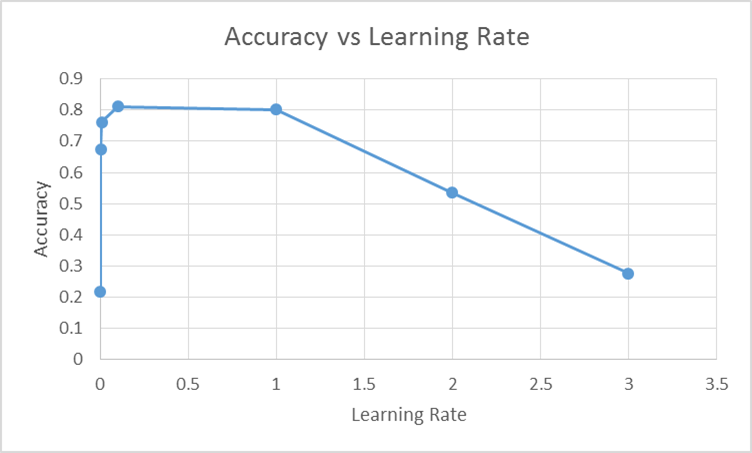


Figure 4

Figure 5 shows the accuracy vs the number of epochs, which refers to iterating over the whole network n amount of times. Around 4 epochs seems to be the optimal balance between peak performance and minimal program runtime. Each epoch can essentially double the original runtime of the network. Too few epochs and the network still has room to learn and update its weights further.

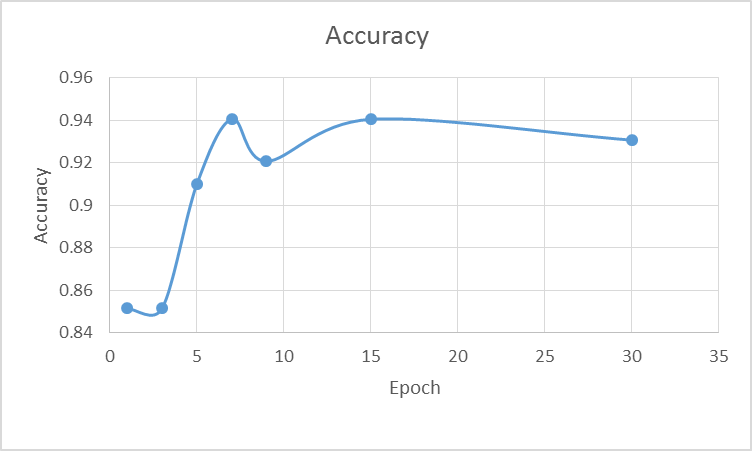


Figure 5

Looking forward there are many applications in which this type of machine learning can be applied. Digit and symbol served as an introductory to this field but we would like to take on a much bigger task with a higher input space. To do this, we would take on tasks of increasing difficulty such as classifying real images of fruit or other objects that would serve as good categories for classifiers. Ultimately, the experience we gain from this project and future related machine vision projects in between will be for the goal of achieving object recognition in videos.