Neural Networks

Russel O-Brien, Isaac Kabuika, Elliot Ketchel
Bowdoin College, Computer Science

Keywords: Learning, Neural, Pattern, Brain, Training

1. Introduction

A neural network is a system that mimics the human brain. It is composed of pseudo neurons, and is used for classifying inputs into categories. In a neural network, an input is split into component parts, and each part is given to an input node. These input nodes represent dendrites in a human neuron, which collect input from other neurons in the form of electro-chemical signals. The inputs of each input node are summed in the output nodes according to the weights of the connections between the input and output nodes. These connections represent axons, and their strength changes as the neural network learns.

Just like a human learns new skills and knowledge by strengthening and pruning connections in their brains, a neural network learns by changing the weights between input and output nodes. Weights are updated according to the errors between the values expected at the output nodes for a given set of input nodes, and the actual value observed at the output nodes.

The specific type of neural network we implemented for this project is called a perceptron. It is especially simple as it contains no hidden nodes, or nodes in between the input and output nodes that create a second set of connections. Without hidden nodes, our network will be less sophisticated, but also much easier to implement, as we are spared of the problem of how to calculate errors at these intermediate nodes.

This style of algorithm is commonly used for classification as solving classification problems by more conventional means would be nearly impossible. For complicated classification tasks such as identifying a picture of a dog versus a picture of a cat, there are way to many subtle variables that need to be taken into account. For example, an algorithm could not just focus on if the animal has a tail or what color it is, as some dogs and cats have tails, some don't, and both can be many different colors. Our brains can do this task easily because they have a network of neurons that handle all subtle associations we have with cats and dogs, and have been trained by learning to condense them into a single answer.

2. Problem

The specific problem we will be solving for this lab is that of identifying handwritten numbers. Handwritten numbers cannot be classified easily without neural networks, as just as with the cats and dogs example, there are way too many variations in how any single number could be written depending on an individual's handwriting.

The handwritten numbers were represented for us in two different ways. The first representation of handwritten numbers was a 32x32 bit array. The array represents a rectangle in which someone drew a number from 0 to 9, and the positions in the array that were shaded by the pencil or pen were set to 1. In other words, if the array was printed out, it would show the number "drawn" in 1's over a background of 0's.

The second representation of handwritten numbers was more concise but also more crude. With this representation, the 32 bit array was divided into 64 4x4 bit subsections. Each subsection was then represented by a single integer in the range of 1-16 according to how many of the bits in the 4x4 array had a value of 1.

3. Neural Networks

In this section we will go over how we implemented our neural network in more depth. As discussed in the introduction, a neural network is composed of a set of input nodes that take in the stimulus info, each of which have connections to every output node. The output nodes are assigned values based on the values of the input nodes and the weights of their connections to the output nodes.

We'll start by describing the data structures we use to represent these key components. First, we use arrays of doubles to represent our input nodes and output nodes. Each cell in these arrays will represent the value of a single input or output node. Next, we represent the connections between input and output nodes with a custom Connection object. Each connection object carries an input and output node, which are integer values which represent the indices in the input and output arrays of the nodes to which this connection attaches, and a double value containing the weight of the connection. All connections will be initialized with random weights in the range of -0.15 to 0.15.

The different representations we use for our handwritten letters will determine the topology of the input nodes. More specifically, if we are using a 32x32 bit array for our input, we will have 1024 input nodes), each assigned a value of 0 or 1. In the case of the 64 integer representation, we will have 64 input nodes, each assigned a value of 1-16.

We also use two different output topologies for our neural net. The first one only requires a single node with a connection to all the input nodes. At this node, the output classification is taken to be the sum of all its inputs passed through a sigmoid function(equation 2, figure 1), multiplied by 10 and rounded to the nearest integer. The second of the output topologies involves 10 output nodes, each with connections to all of the input nodes. The value of each of these nodes is calculated in the same manner as with the single output node representation. However, the classification of this output topology is taken to be equal to the index of the node with the highest value. For example, if the first output node has the highest value, then the "answer" given by the network was that the input given represented a hand drawn 0.

As we will discuss below, we also need to have optimal values for our output nodes in order to calculate error. For the first output representation that uses a single node, the optimal output value is simply the number that the input data is representing. For the output representation consisting of 10 nodes, the optimal solution is an array of 10 nodes all set to 0 except for the index corresponding to the number represented by the input, which is set to 1. For example, if the input data represents a 3, the optimal output would be [0,0,0,1,0,0,0,0,0,0].

The values of each of the output node are calculated using the summation described in equation 1. In words, this equation says that for each connection to an output node, the output node will add to its value the weight of that connection multiplied by the value of the input node on the other end of the connection. After this summation has been calculated, each input node will pass this value through the sigmoid function. The purpose of the sigmoid function, as shown in equation 2, is to assure that the value of each output function lies somewhere between 0 and 1 to allow for classification of all output values.

In order to “train” our neural network so that it can give the proper outputs for a set of inputs describing a number, we need to run a series of adjustment steps or epochs during which we will adjust the weights of all connections in the network. One epoch goes as follows. First we set all the values of our input nodes according to the input data(either 1024 bits or 64 integers). Next, from this input data and the current weights in the network, we calculate the values of all output nodes. Then, for each connection, we calculate the error in its output node by subtracting its actual value from the optimal value of that same node in the optimal solution for the current input. This quantity is referred to as Err in equation 3, which describes how we set the new value for the weight of a connection. Once we have Err for the output node of a connection, we add to the weight of that connection the product of Err, the derivative of the sigmoid evaluated at that connections output value, the connections input value, and our learning rate.

The final noteworthy bit of our implementation is the use of a bias node. This is a single input node we added onto the end of our input node array, whose value is always set to one. This node has connections to all the output nodes, which are trained just like all the other connections.

$$(1)\quad Output_i = \sum_{k \varepsilon j} I_k W_k \textit{ where } j \textit{ is set of all input nodes, } I_k \textit{ is the value of input node } k, \textit{ and } W_k \textit{ is cxn weight to } i$$

$$(2)\quad g(x) = \frac{1}{1+e^{-x+0.5}}$$

$$(3)\quad W_k = W_k + (Err * g'(Output_k) * I_k * Lr)$$

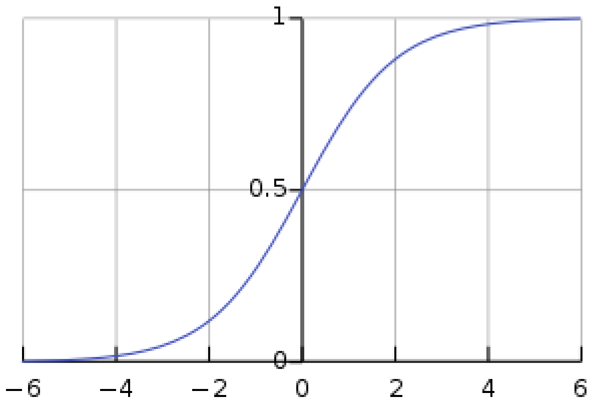

Figure 1: Graph of sigmoid function.

4. Experimental Methodology

With our experiments we hoped to determine several things about our neural network. First we wanted to test for which topology of input and output nodes gave the best percentage of correctly classified inputs. Second, we wanted to see how quickly/how many training epochs each topology needed to be able to produce good results. Finally, we wanted to see the effect on classification accuracy changing the learning rate had.

Before describing the specific tests we ran, we'll go over how a single test works. After the neural network has gone through its training epochs, we enter a different protocol for running a test. This protocol goes as follows. First, we fill the input nodes with using test data which is in the same format as training data. Then we calculate the output node values the same as we do when training. The one difference between training and testing is the next step, in which we simply get the classification based on the heuristic discussed in section 3, and do not update any connection weights. If this classification matches the expected value for this input data, we count this test as passed. We will be reporting the results of our data in terms of the percentage of tests that were correctly classified.

The first round of tests we will run will measure how different topologies perform after different numbers of training epochs. To do this we will run a set of 4 tests for each of the 4 possible network topologies(ie 64 input and 1 output, 64 input and 10 output, 1024 input and 1 output, 1024 input and 10 output). The four tests done in each for each of these topologies will be getting and evaluating the correctness of 100 classifications after training the network for 10, 50, 100, and 500 epochs. Each of these 16 test settings will return the average percentage of correct classifications across 10 separate trials. Finally, for these tests, the learning rate will be set to 0.1.

Next, we wanted to determine what effect the learning rate had on our neural network. For this round of testing we again ran tests for all 4 combinations of topologies. We ran 4 different sets of tests on each topology, taking the average percent correct(out of 100 input tests) for 10 trials. The learning rates we used for these trials were 0.01, 0.1, 0.5 and 1, and for each trial we trained the network for 1000 iterations, just to make sure that the number of iterations would not be a limiting factor.

5. Results

From our tests on the number of training epochs, we can see a clear increase in the number of correct classifications for the topology of 10 output nodes and 1024 input nodes. As far as the other topologies, however, there is only a slight increase in performance when training epochs are increased for the 10x64 and 1x1024 topologies, and no increase in performance for the 1x64 topology. In terms of absolute performance, the 10x1024 topology vastly outperformed the other topologies with a maximum classification rate of around 80, whereas the other topologies hovered in the low teens for all trials(figure 2).

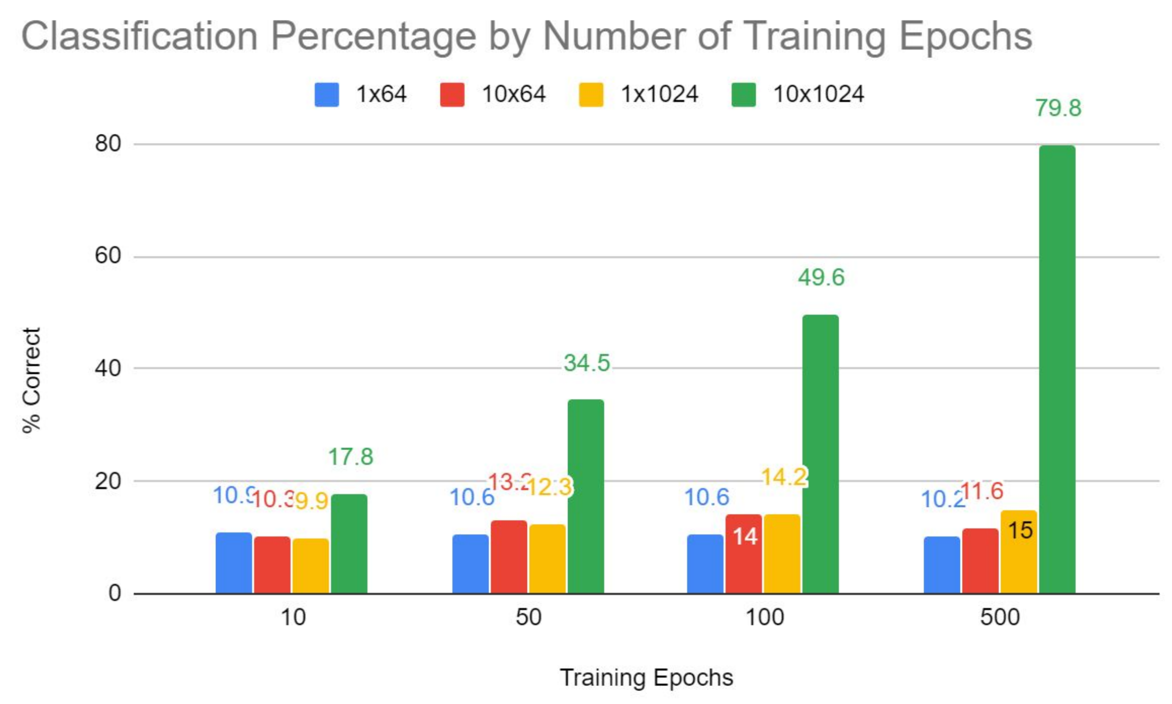

Figure 2: Shows the change in correct classification rate of the neural network for different topologies after varied number of training epochs

As for the tests regarding the learning rate of the network, the data indicates two different learning rates as optimal for different topologies. For the 10x1024 topology as well as the 1x1024 topology, the learning rate that yielded the best results was 0.1. However for the topologies with 64 input nodes, the best learning rate was 0.01. Once again the 10x1024 topology yielded the best results, with a peak classification rate of 88.6. With the exception of the 10x64 topology, which showed a significantly better peak classification rate of 36.3, the other topologies showed similar classification rates as they did in the previous round of experiments(figure 3).

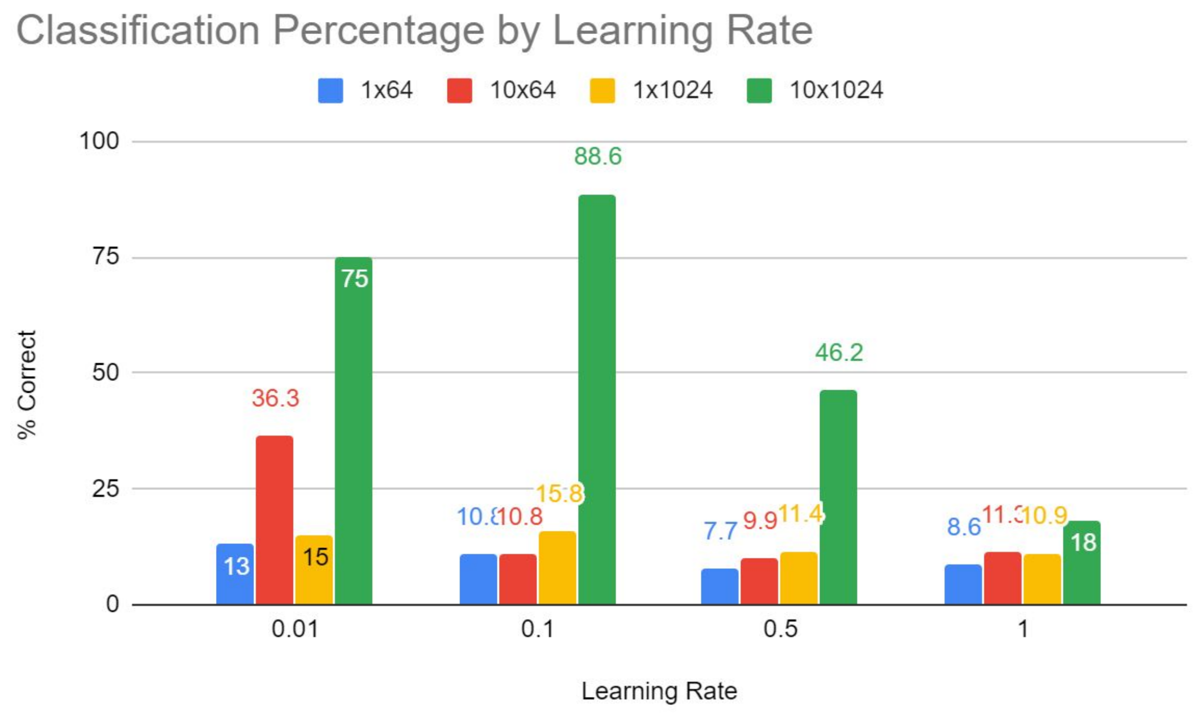

Figure 3: Shows the percentage of correctly classified tests by learning rate and network topology.

6. Discussion

The most obvious conclusion to draw from these results is that the topology of 10 input nodes and 1024 output nodes performed significantly better than did all the other topologies. Its peak classification rate was almost 90 percent, whereas the other topologies never cracked 40(figures 2,3).

This is in part because the 1024 bit input has much more precision than the 64 integer input. More specifically, the 64 integer input does not specify in detail the position of points in the larger bit array, as it only looks at the number of colored bits in smaller chunks of arrays. It is essentially a low resolution version of the 1024 bit array, which specifies down to the bit the shape of the hand drawn number. Therefore, it is no surprise that the neural network has more trouble honing in on the correct weights for the 64 input node topologies.

Furthermore, since the 10x1024 topology clearly outperforms the 1x1024 topology(figures 2,3), and the 10x64 topology has a higher peak classification rate than the 1x64 topology(figure 3), a 10 node output must be better than a 1 node output. One likely reason for this is that the space of possible output node configurations is evenly distributed among all 10 classification possibilities in the 10 output node topology, but with the single output node, there are disproportionately more configurations that result in classifications toward 0 or 9. More specifically the sigmoid function(figure 1) has asymptotes at y = 1 and y = 0. Therefore, there are only a small amount of values near x=0 that would result in classifications of 5, but many more values of x in the positive and negative direction that would all give a classification of 9 or 0 respectively. This biasing does not occur with the 10 output node setup, as each individual node experiences this effect independently, so when comparing them to find the max the effect is negated.

Therefore, the single output node will be biased towards the extreme solutions, whereas the 10 output nodes will not. This likely accounts for the fact that the 10 output nodes tend to perform better.

As far as learning rate, the data suggests that any value of 0.5 or above performs worse for all topologies(figure 3). This is likely due to the fact that such a large learning rate would cause the network to continually over adjust the weights of connections, preventing any equilibrium from being reached.

The other observation we took from the tests on the learning rate was that the 1024 input node topologies performed best with a learning rate of 0.1, whereas the 64 input node topologies performed best with a learning rate of 0.01. The most likely explanation for this is that we are dealing with smaller numbers(either 0 or 1) in the case of the 1024 node input, and larger numbers(in range of 1 to 16) in the case of the 64 node input. Recall that the inputs of each node are used as part of the product that calculates the new weight of a connection. Therefore, in order to update the weights of the nodes so that the change in weight is significant for the 1024 bit input, we need a larger learning rate. Similarly, since the inputs for the 64 integer input are larger, the learning rate is likely too large at 0.1, causing the network to overcorrect and fail to converge. Overall, these observations indicate that there is a sweet spot for learning rate that changes based on the size of the inputs.

7. Further Work

Any further work spent on this project would likely be in getting the training with 64 input nodes to run smoother. In our testing process, we noticed that topologies with these inputs tended to converge towards a classification of 0 or 9. We are unsure about what is causing this, but we suspect that our results could be significantly better for these topologies if we figure out what the problem is there.

As far as aspirational projects with neural networks go, one that we would enjoy exploring would be an application that could identify the person who wrote a letter or number. In theory, we could do this by training our network with a single number represented as a bit array, where the different data sets are written by different people. We could assign each person an index in an output array or a decimal range of 0.1(depending on which topology we use), then train the network to identify a person's handwriting. If we had enough data, such an application could potentially be used to produce evidence in criminal trials or any number of other uses.

8. Conclusion

Our findings indicated that we were indeed able to implement a simple neural network that could learn to classify handwritten numbers with decent accuracy(with some topologies). We learned that the higher resolution representation of the handwritten number, as well as an output that counteracted the bias of the sigmoid towards extreme solutions led to better classification results. Finally, we learned that the learning rate has a sweet spot, where it neither over or under adjusts the weights in the network, leading to a higher rate of correct classifications.