An Evaluation of Neural Network Performance For Digit Recognition

Ethan Zhou, Jake Adicoff, and Mac Groves

Abstract: In this paper, we analyze a single-layer feed-forward Neural Network (a *perceptron*) and how the learning rate affects its performance in optical digit recognition. We test on two types of perceptron inputs: 32x32 bitmap images and 8x8 downsampled images. Additionally, we evaluate two perceptron outputs formats: a single node whose values correspond to different integers, and 10 nodes that represent each of the 10 digits. The learning rate parameter will be changed and the performance of the perceptron will measured for each epoch in order to evaluate the performance of the perceptron. Our results suggest that a perceptron network with 10 output nodes operating on 32x32 bitmaps is the best way to identify handwritten digits.

1. Introduction

A perceptron neural network is a function approximation algorithm that is inspired by the function of neurons. A perceptron receives inputs through an *input node layer*, which is fully connected to an *output layer*. The output node(s) produce output based on the input node values and the *edge weights* connecting the input nodes to the output nodes. The output of a perceptron is its "best guess" of the output of a specific function.

Before a neural network is able to approximate a function, it must first undergo a "learning" phase. During this phase, the perceptron is given pairs of known inputs and outputs, and all the edge weights are adjusted so that the network outputs may more closely resemble the known output values. The learning rate and the number of training *epochs* of a neural network have an important effect on the accuracy of the algorithm. Ideally, once a perceptron is trained, it can operate on arbitrary inputs not present in the training set, and produce proper outputs.

Digit recognition is a widely used standard for machine learning algorithms. In this study, we apply the perceptron to optical digit recognition. The digits are given as image bitmaps of integers between 0 and 9. In these bitmaps, values of 1 denote areas with ink and values of 0 denote blank space (see Figure 1). The perceptron will be trained to take a bitmap and return the

digit the image is supposed to represent. We will test the performance of the perceptron with two types of image inputs formats and two types of output node strategies.

We will compare the error rate of the four combinations of input-output problem representations, and evaluate the quality of the resulting neural nets. In section 2 we discuss the two input formats that our perceptrons were trained on. In section 3, we discuss the perceptron in detail, and the two two different perceptron architectures we tested. In section 4, we describe the methods used to evaluate our four different perceptron combinations. In section 5 we present our results and discuss their implications. In Section 6, we suggest further research in the use of neural networks. In Section 7, we conclude our study with a summary of our findings.

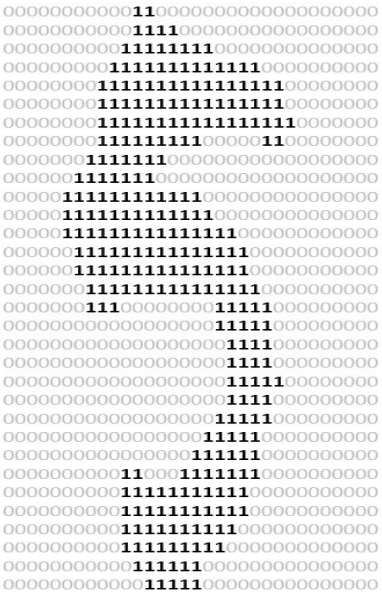


Figure 1: A sample of a 32x32 bitmap of a handwritten '5'. The 0's have been desaturated and the 1's have been put in boldface to emphasize the relevant information.

2. Optical Digit Recognition

The Optical Digit Recognition (ODR) problem is subset of the Optical Character Recognition (OCR) problem. OCR is concerned with identifying characters in images of

handwritten or printed characters. While this may seem like a trivial problem for humans, OCR is a difficult problem for machines to solve. A major hurdle is that any given character can appear very different from two different sources. Among printed sources, different fonts can display a lot of variance (e.g. “a” versus “ɑ”), and this problem is even more extreme among handwritten characters. Furthermore, OCR typically processes scans or photographs of words which can introduce artefacts to the data, like pixelation, blurs, alignment issues, etc. These complications make it difficult and unfeasible to hard-code a set of rules to identify every character accurately. Instead, a Neural Network is more well-suited for the OCR problem, since they can learn to identify patterns to accurately classify a large variety of input data into categories.

We tested our perceptron two different input representations of the handwritten character data. The first is the 32x32 bitmap, illustrated in figure 1 above. This format essentially divides an image of a character up into a 32x32 grid, and assigns a 1 to every cell that has “writing” present, and 0 to empty cells. The corresponding target value is, of course, the value of the digits written in these images. The second format is a compressed 8x8 version of the original 32x32 bitmap. These 8x8 maps can be visualized as a lower-resolution grid that is placed directly over the higher resolution 32x32 bitmap, such that each cell in our 8x8 grid overlays a 4x4 grid in the 32x32 bitmap. Each cell in this 8x8 map contains the sum of these 16 bits (resulting in any value between 0 and 15). The target of these compressed character data is the same value as the uncompressed digit.

3. Perceptron Neural Networks

Neural Networks are loosely inspired by the behavior of animal brains. Brains contain a large collection of interconnected neurons, each of which can interact with the neurons it's connected to. Neurons can both receive and send signals to other neurons. Neurons have an activation threshold that is a function of the their input signals, and when the threshold is met, the neuron will produce a signal of its own to send to downstream neurons. Neural networks mimic this behavior, and replaces the activation threshold with an *activation function*, which will be explained later. By adjusting the strengths of the connections between the artificial neurons in the network, the algorithm can "learn" to identify patterns from inputs to make classification decisions.

The perceptron consists of an *input node layer* that is connected to an *output node layer*. The two node layers are fully connected, i.e. there exists an *edge* between every input node and every output node. The two perceptron architectures we test will have a different number of nodes in the output layer: one with just 1 output node, and one with 10 output nodes. Each node can output a value in the range [0,1] inclusive. In 1-node version, the final output is interpreted by multiplying the output node value by 10 and rounding down to the nearest integer. In the 10-node version, each node represents a digit, and the final output is the digit associated with the node with the largest value. We will just discuss the one output node model in detail, and extend the concept to the 10 node model.

The input of a perceptron is a vector of numbers. For our ODR problem, the input layer vector is a flattened (1-dimensional) version of the digits' bitmap. The j^{th} input is denoted as I_j. Every edge in the network (connecting I_j to the output node) has a weight W_j. These weights are

initially set to a random value between -1 and 1. The output node takes the dot product of the input vector with the weight vector, and feeds that value to an activation function to produce some output. The activation function we use is a sigmoid function:

$$\vec{I} \cdot \vec{W} = x; g(x) = \frac{1}{1 + e^{0.5-x}}$$

We refer to the output of the net *g(x)* as *O*. The goal of a neural net is to set the values in the weights vector *W* such that given an arbitrary input vector *I* with a target value *T*, *O* is as close to *T* as possible. We also include an extra 'bias node' as part of the input vector. The value of this always 1, but its edge weight is updated along with all other edges during the training process. This bias node essentially allows the neural net to "tune" the slope of the output node's activation function, yielding better results.

The process of setting the values of the weight vectors is called *training*. A Neural Network is trained on a large set of inputs with known target values *T*. These inputs are first fed into the untrained net, which will not produce good output values *O*. However, this error *T-O* can be used to update the network's edge weights, so that future similar inputs may have a smaller error. Each edge weight W_j connected to the output node is updated as follows:

$$W_j = W_j + (\alpha \times I_j \times Err \times g'(I \cdot W))$$

In this function, α is some learning rate, usually set between 0 and 1, and *Err* is equal to *T-O*. With each target/output pair (*T,O)*, there is an associated mean squared error surface. The above edge weight update equation is equivalent to a negative gradient descent on this error surface, so that error is minimized. One total iteration of this weight update process constitutes an iteration

of training, or an *epoch.* To achieve the best results, a neural net is trained over many epochs, but not too many so that the neural net *overfits* the problem. Overfitting occurs when the neural net outputs values that reflect the training set well, but fails to generalize to new inputs.

If there are multiple output nodes, a $\vec{I} \bullet \vec{W}$ is calculated for each output node, where each output node has its own weights vector. The update is also done for each output node, and the *T* value may be different for each node. With these differences, each output node will respond differently to the same inputs, and the final output is encoded in some way by all of the nodes as a whole.

After training, we are able to evaluate the neural net using a new set of test inputs that is different than the training set. Input vectors are fed into the neural net, and we compare *T* and *O* values. Edge weights are not updated in this phase. We measure error as:

$$E = \frac{n - \sum_{k=0}^{n} a}{n}; \text{where } a \text{ is 1 } \textit{iff } T{=}O, \text{ 0 otherwise}$$

Where *n* is the number of test inputs. The more times the neural net's output matches the target output, the lower the error is.

4. Experimental Methods

We sought to understand how the input representation, number of outputs, and learning rate affected the digit recognition performance of the perceptron. We tested the perceptron using 32x32 bitmaps of handwritten digits as well as 8x8 downsized integer maps of handwritten digits (described in Section 2). We also tested two different perceptron architectures: a network with one output node, and another with ten output nodes. Finally, we tested learning rates of 0.01, 0.1,

0.5, and 1.0. We collected data for perceptrons using all possible permutations of input type, number of outputs, and learning rate, resulting in 16 total perceptron tests.

To assess the accuracy of each perceptron, they are trained on one set of inputs and tested on a completely different set of inputs. For the 32x32 bitmap inputs, the optdigits-32x32.tra inputs were used to *train* the perceptron and the optdigits-32x32.tes inputs were used to *test* the perceptron. For the 8x8 bitmap inputs, the optdigits-8x8-int.tra inputs were used to *train* the perceptron and the optdigits-8x8-int.tes inputs were used to *test* the perceptron. The perceptrons were trained for 50 epochs, and were tested after every epoch so that the performance could be tracked throughout the training of the perceptron.

5. Results and Discussion

Perceptron performanced ranged greatly depending on the neural network architecture (input and output design) and learning rate parameters. The best perceptron configuration used a 32x32 bitmap input, 10 output nodes, and a learning rate of 0.1, which correctly identified 94.6% of the test digits. Perceptrons with the same architecture with learning rates of 0.01 and 0.5 also had competitive performances, correctly identifying over 93% of the test digits. Many configurations could only identify around 10% of the digits correctly, which is equivalent to randomly guessing a digit from a set of 10 numbers.

The 32x32 bitmap inputs greatly outperformed the 8x8 integer map inputs. The best 32x32 bitmap perceptron correctly identified 94.6% of the test digits, while the best 8x8 integer perceptron with a learning rate of 0.01 and 10 output nodes correctly identified only 65.8% of the test digits. We believe that this is due to the lossy nature of the 8x8 integer maps. Many 32x32

bitmaps would reduce to the same 8x8 map, since the smaller integer maps simply sum the bits in a given reason, losing information about the specific positioning of the bits within the 4x4 compression sector. Additionally, the 32x32 bitmap inputs have 1024 input nodes, while the 8x8 integer inputs only have 64 input nodes. This means the 32x32 bitmap perceptrons have over 10 times as many weights as the 8x8 integer map perceptrons, and therefore could make finer adjustments and identify more nuanced characteristics to correctly identify digits. It is possible that a neural network with a hidden layer may improve the performance of 8x8 integer map inputs.

The better performance of the 32x32 inputs came at a computational cost: the best 32x32 perceptron completed 50 epochs in 7.7 seconds while the best 8x8 perceptron completed 50 epochs in 0.63 seconds. This is illustrated in Figure 4. However, the best 32x32 perceptron took fewer epochs to reach peak performance than the best 8x8 perceptron. The larger perceptron hit a performance plateau almost immediately, while the 8x8 perceptron required 7 epochs to reach peak performance. This behavior can be seen in Figure 3.

The learning rate affected both how well the perceptron performed and how quickly the perceptron was trained. When the learning rate was too large, the perceptron required more training epochs to reach optimal performance. In the worst cases, performance often suffered compared to other trials with lower learning rates. For the perceptrons with 32x32 inputs and 10 output nodes, learning rates of 0.01 and 0.1 quickly trained the perceptron to correctly identify over 90% of the test digits. A learning rate of 0.5 caused the perceptron to require more epochs before the performance was over 90%, and the perceptron with a learning rate of 1.0 never surpassed a performance rate of 50%. The importance of the learning rate is diagramed in Figure

2. We believe that when the learning rate is too big, we get divergent performance similar to the the rightmost plot in Figure 2. The weights jump around too quickly for the perceptron to achieve accurate digit recognition.

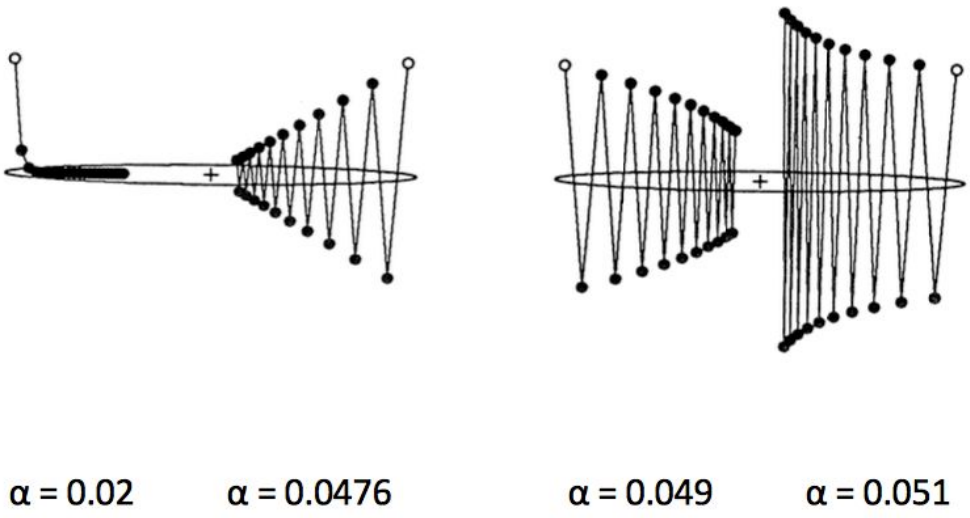
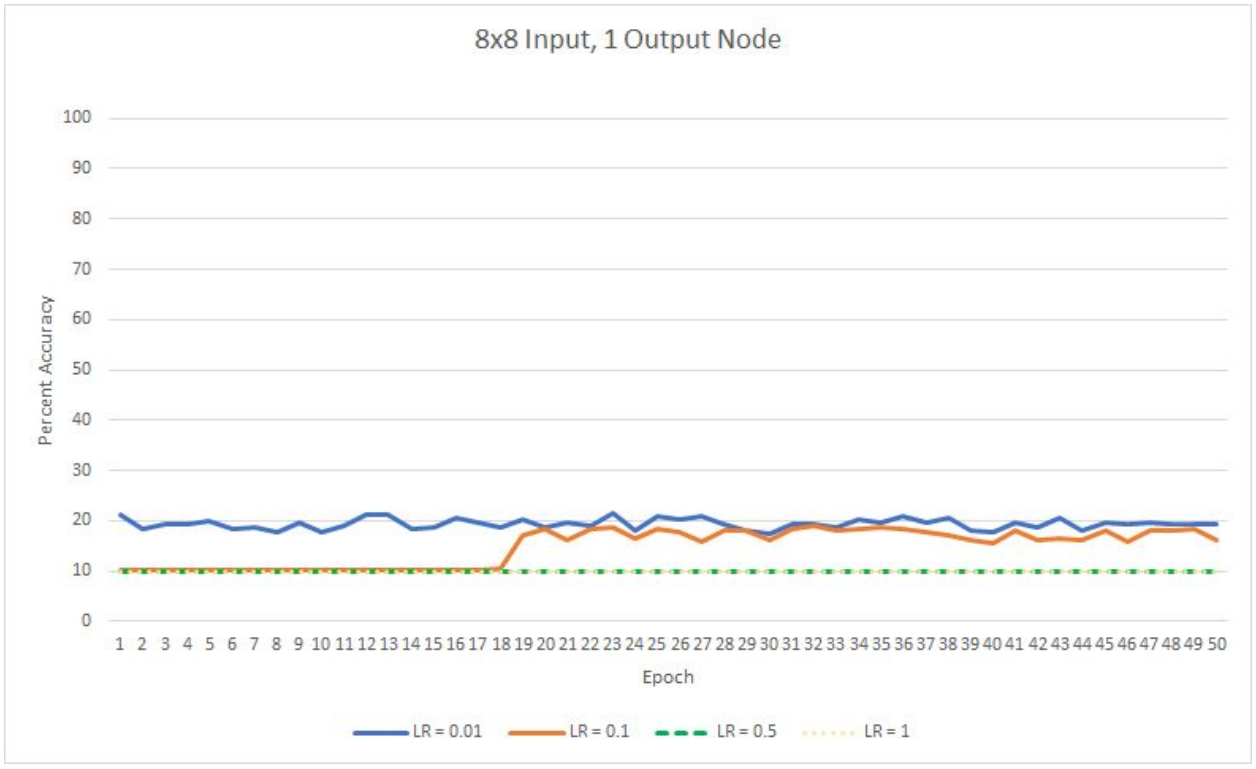


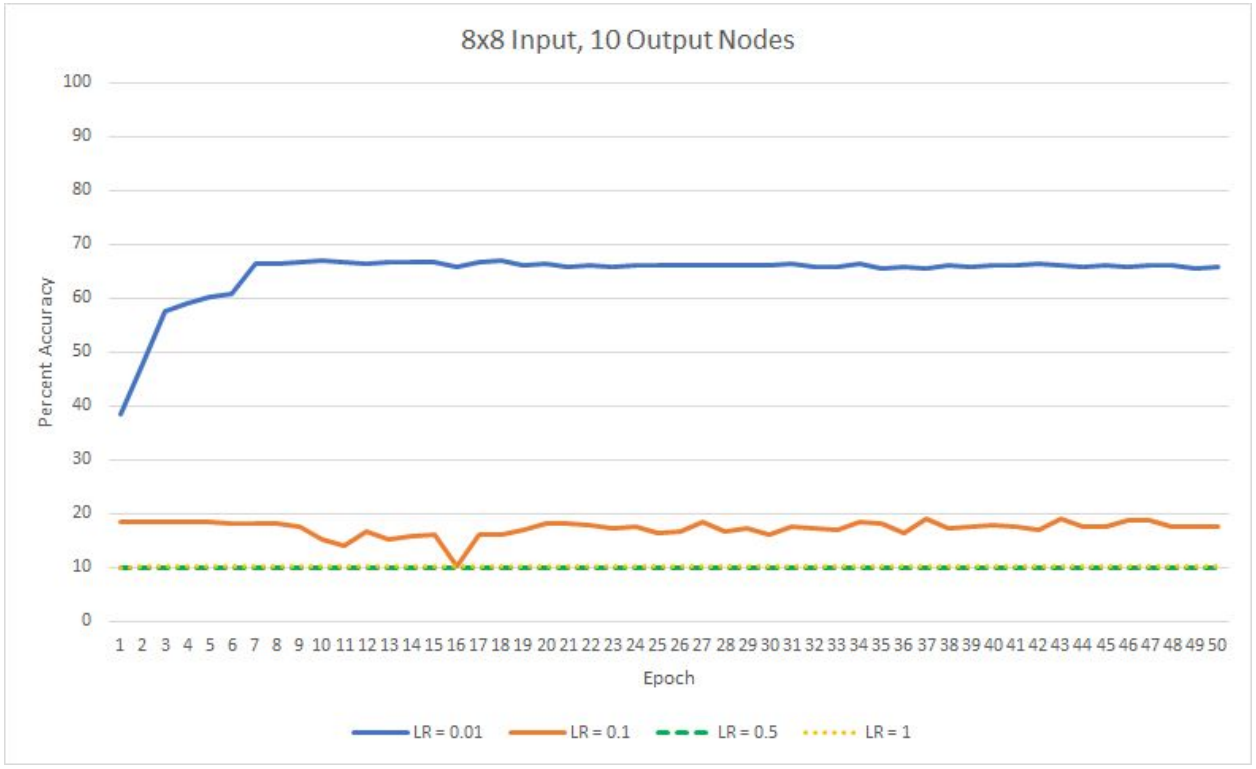
Figure 2: A diagram of the neural network search function behavior at different learning rates. The ellipses represents a surface with low error, and each set of connected lines represent several epochs of a hypothetical neural network.
This diagram shows that if the learning rate is too low, the network may avoid the problem of over correcting for the error, but fail to approach the absolute minimum error. On the other hand, networks with too large a learning rate may exhibit divergent behavior, and never reach the minimum of the error surface. (taken from Stephen Majercik's slides for Nature-Inspired Computation, Bowdoin College 2017)

The output nodes representation in the perceptron was one of the most important factors affecting the performance of the perceptron. None of the perceptrons with only one output node surpassed a performance rate of 25%. Generally, the performances of the one-output node perceptrons localized around 10% and 20% correct. We believe that 10% performance means that the perceptron is not functional and is randomly guessing numbers or selecting the same number for every input, succeeding only one out of 10 tries. An analysis of the perceptron digit

recognition guesses reveals that when the perceptron has a 10% performance, it guesses that the digit is “0” the majority of the time. We believe that 20% performance means that the perceptron is only successfully guessing digits at the end ranges of the activation function. An analysis of the perceptron digit recognition guesses reveals that when the perceptron has a 10% performance, it guesses that the digits is “0” or “9” almost all the time. This makes sense for the perceptrons with one output node, because our activation function is a sigmoid, which levels off at the values that correspond to the digits “0” and “9”. The inflection point of the sigmoid activation function occurs at a very narrow range, and is thus difficult to have values away from the two extremes.

Perceptrons with 10 output nodes were far more successful at recognizing the handwritten digits. Three of the perceptrons with 8x8 inputs and 10 output nodes had performances around 10%, which corresponds to random or constant guessing. One 8x8 perceptron was able to achieve 66% accuracy. Three of the perceptrons with 32x32 inputs and 10 outputs had performances over 93%, and the other had 48% accuracy. All the perceptron performance values per epoch can be found in Figure 3. When the perceptron has 10 output nodes, each node corresponds to a digit and the digit guess is the output node with the largest value. The activation function is designed to be stable at function values of 0 and 1, so it makes sense that the networks work better with an output format such that the nodes are supposed to return binary values, compared to a format that attempts to utilize the transition slope of the sigmoid function.


8x8 Input, 1 Output Node
Percent Accuracy
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Epoch
LR = 0.01
LR = 0.1
LR = 0.5
LR = 1


8x8 Input, 10 Output Nodes
Percent Accuracy
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Epoch
LR = 0.01
LR = 0.1
LR = 0.5
LR = 1

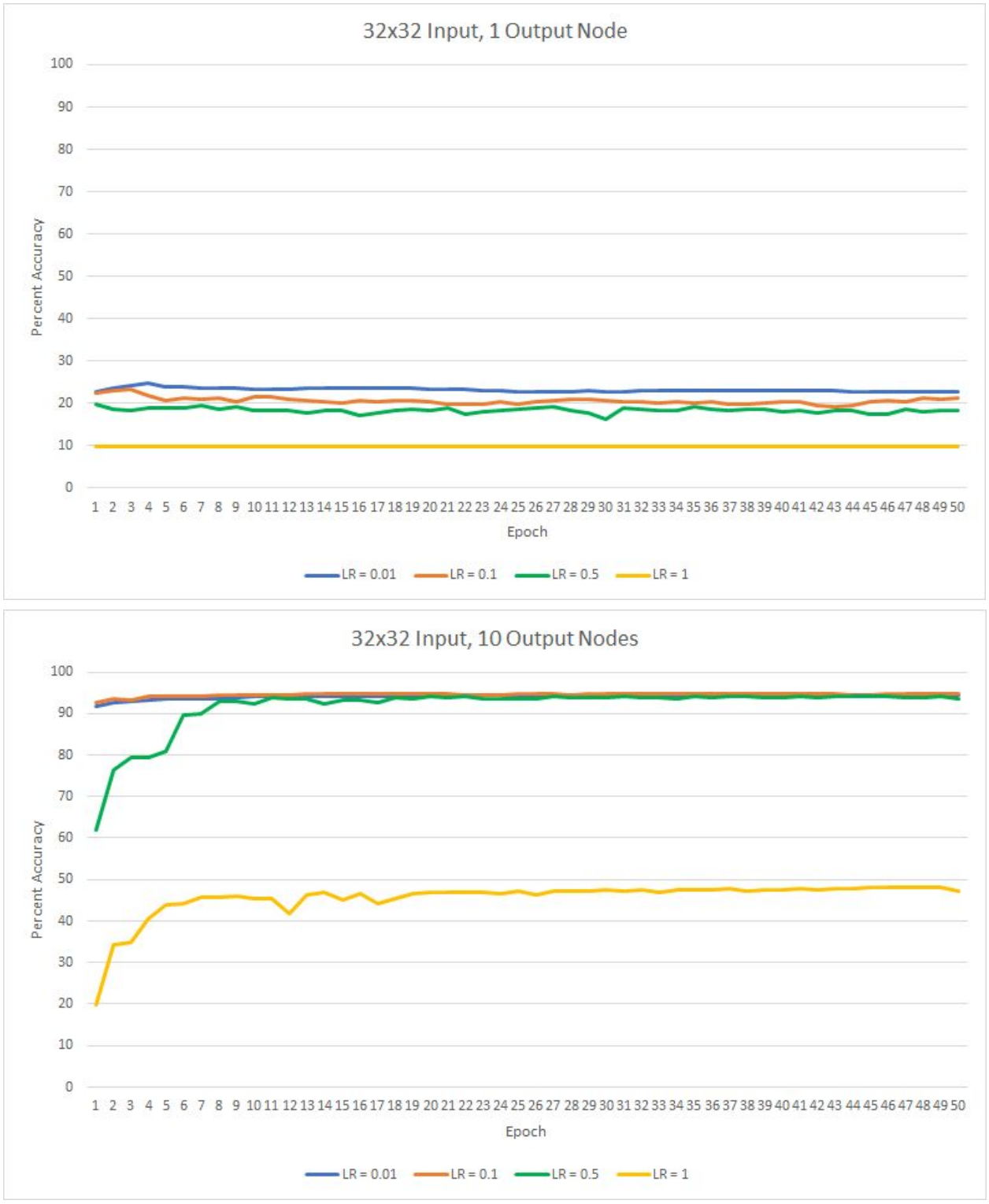


Figure 3: Graphs of Perceptron Accuracy Rate at each learning Epoch, at different learning rates. Each graph displays the performance of a different input/output configuration (input map dimensions, number of output nodes). For all configurations, a learning rate of 0.01 has the best performance. Ten output nodes outperforms a single output node, and a larger, more detailed input format allows for higher character identification accuracy.

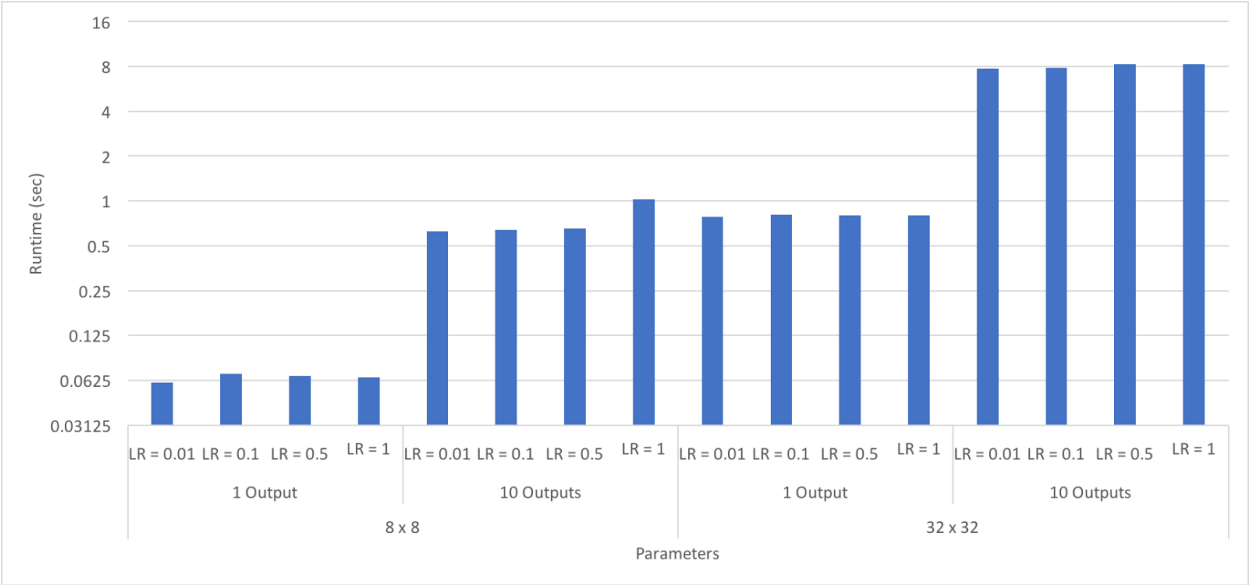


Figure 4: The runtime of the different perceptron configurations. Runtime is mainly dependent on input map size and number of output nodes, with larger inputs and outputs taking more time. Learning rate has a very slight effect on runtime, and there does not appear to be a pattern.

6. Further Research

It would be worthwhile to investigate the performance of other possible perceptron architectures. One possible output format may have just four output nodes, where each output node value would be rounded to 0 or 1 and the values would be interpreted as binary digits to give an integer output. Perceptrons with one output node were able to run faster than the perceptrons with 10 output nodes because they only had to implement 10% of the number of weights. However, the activation function does not work well for digit recognition with one output node because the inflection point is very unstable. Four output nodes would balance the smaller number of weights with the limitations of the activation function. Output node values could still be 0 or 1, but there would be 60% fewer weights to adjust than perceptrons with 10 output nodes.

Additionally, we would like to try different image compression methods. The 8x8 integer bitmaps proved to be a lossy compression method that did not work very well with the perceptron. However, there may be other compression functions that are less lossy, but with the added benefit of a smaller input size that decreases the runtime of the perceptron. We will explore this in our final project!

Finally, it would be worthwhile to implement the neural network with hidden nodes. In particular, the performance of the 8x8 input perceptrons was not competitive in this study (<70% correct tests) but the hidden nodes may make it possible to balance small input size with good digit recognition.

7. Conclusions

The perceptron neural network is an effective implementation of digit recognition that can correctly identify handwritten digits over 94% of the time. The perceptron works best with large uncompressed 32x32 bitmap inputs of the digit, a 10 output node format, and small learning rates around 0.01 and 0.1. Compressed 8x8 integer map inputs and one output node architectures did not perform as well. Since the time differential between 1 node and 10 nodes in each input map type is not remarkable while the difference in accuracy is, we recommend strict use of the of the perceptron network with 10 output nodes.