Artificial Intelligence COSC 6368
Spring 2018

Classifying Numbers using a Perceptron

Yeh-Wern Chiam – 0991594

Navleen Kaur – 1641012

1. About Neural Network

1.1. Introduction to Neural Network

Inspired by the central nervous system of a human, particularly the brain, Artificial Neural Networks was born. It was a major research area for neuroscience and computer science until 1969 when Minksy and Papert published a book titled “Perceptrons” that diminished research activity for neural networks. It was not until the 1980s that researches developed algorithms that modified the Neural Network weights and threshold more efficiently. This efficiency addressed the limitations brought up by Minsky and Papert. And with the help of the Computer Game Industry, neural network was resurged. Video games have complex imagery that require hardware than keep with its pace. To do so, Graphics Processing Units (GPU) are used. GPUs have a similar architecture to neural networks. It packs thousands of simple processing cores on a single chip. While in the past there were 1 to 3 layer networks, now modern GPUs allowed the growth to 15 to 50 layer networks. Neural Networks can also be seen in many other real world applications such speech recognition, text translations and face identifications.

Modeled loosely after the brain, Neural Networks consists of thousands or millions of simple processing nodes that are densely interconnected. These nodes model after the biological neuron of the brain. Also referred to neurons, the nodes are the building blocks of the neural networks.

Like the brain, the neural network receives an input signal, processes it and sends the corresponding output signal. To mimic this, the topology of neural network is layered structures. It consists of an Input Layer, Hidden Layer(s) and an Output Layer. The number of hidden layers is arbitrary and are the computational engine of the multilayer neural network.

The input layer sometimes also referred to as the bottom layer consists of a layer of nodes. This layer if fed with training data and picks up the input signal. It then passes the signal to the next layer, the hidden layer. There can be one or more hidden layers in a neural network. The processed signal is then passed to the output layer that delivers the result.

To process the input, the individual signal is multiplied with the weight assigned to each respective node. As the data is passed to the successive layers, it gets added and multiplied until it arrives at the output layer. The weights and thresholds for the nodes are continually adjusted until the training data yields the same output labels as the target labels.

1.2. Perceptron Theory

The perceptron is a single layer neural network. Invented by psychologist Frank Rosenblatt, it was intended to be a machine instead of a program. As a binary classifier, the perceptron is capable of classifying linearly separable data. The perceptron predicts if the input belongs to the class above the line or below the line. There are four parts to a perceptron: the input layer, the weights and biases, the sum and the activation function. This is depicted in the following figure.

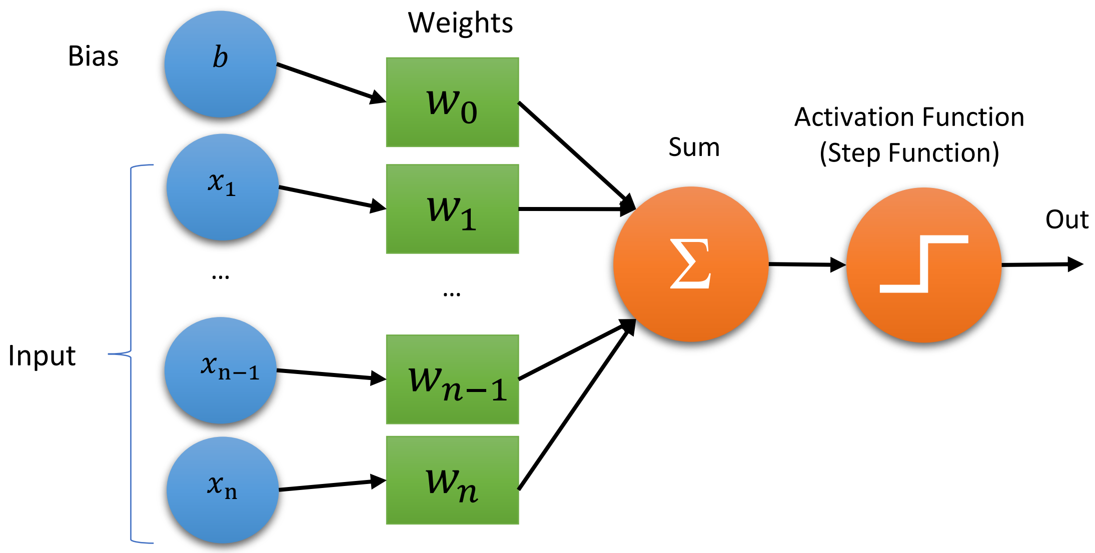


Figure 1. Perceptron Structure

The inputs x are multiplied by their respective weights w. The multiplied values are then added together to yield the weighted sum, equation 3. The multiplication and addition step are also referred to as taking the dot product of the weight vector(W), equation 1, and the input vector(X), equation 2. An offset called the bias is added at this step, equation 3. The weighted sum is then applied to the correct activation function. In this case, the Heaviside step function is used, equation 5. It returns 1 if the input to it is positive or zero and returns 0 if it is negative.

$$W = (w_1, w_2, \dots, w_n) \tag{1}$$

$$X = (x_1, x_2, \dots, x_n) \tag{2}$$

$$\sum_{i=0}^{n} w_i x_i \; = WX = (x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n) \tag{3}$$

$$y = WX + b = (x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n) + b \tag{4}$$

$$\begin{gathered} O(X) = sgn(WX) \; where \\ sgn(y) = \begin{cases} 1 & if \; y > 0 \\ 0 & otherwise \end{cases} \end{gathered} \tag{5}$$

The weights represent the strength of each particular node. Initially, the weights and thresholds are all set to random values. They are adjusted in the learning phase based on the error of the last test result. With each example and iteration, the weights are updated according to the Perceptron Rule in equation 6. The perceptron stops when all the training examples are correctly classified.

$$\begin{gathered} w_i = w_i + \Delta w_i \\ Where \; \Delta w_i = \; \eta(t - o)x_i \\ t : target \; output \\ o : output \; generated \; by \; the \; perceptron \\ \eta : learning \; rate \; (e.g. \; 0.1) \end{gathered} \tag{6}$$

The perceptron uses the step function as the activation function as it is a linear function. Examples of non-linear activation functions for other neural networks are: sigmoid/logistic activation function, tanh/hyperbolic tangent activation function and Rectified Linear Unit(ReLu) to name a few. Figure 2 below depict a few examples of activation functions.

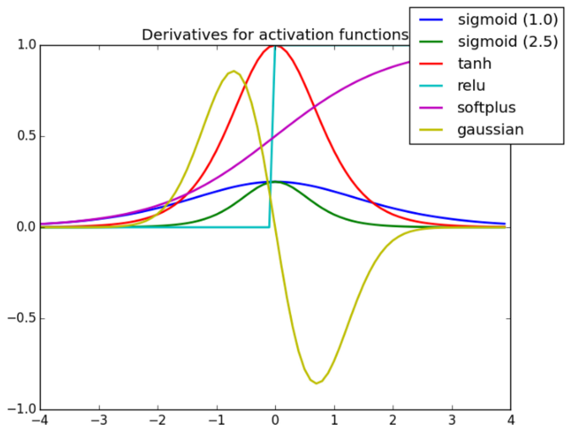


Figure 2. Activation Functions

1.3. Multilayer Perceptron

Multilayer Perceptrons(MLP) are capable of handling non-linear data. They are composed of more than one perceptron. It consists of the input layer, an arbitrary number of hidden layer and an output layer that makes a prediction about the input. They are often applied to supervised learning problems. The MLP is trained on input-output pair data to learn to model the dependencies between the two. In training, the weights and biases are adjusted to minimize error. Error is measured in many different ways. One example is the Root Mean Square Error, equation 7.

$$Error = \frac{1}{2}\sum(target - output)^2 \tag{7}$$

MLPs are feed forward networks. In the forward pass, the signal flows from the input layer to the hidden layer then to the output layer. This yields a guess by the network. The guess is then compared against the truth label or alternately known as the target output. In the backward pass, the network utilizes backpropagation, chain rule, partial derivatives of the error function with respect to the weights and biases and then back propagates it to the network. The differentiation gives a gradient that the parameters can be adjusted to minimize error. This can be done by using gradient based optimization like stochastic gradient descent. The network does forward pass and backward pass repeatedly until the error converges.

2. Implementation

For the experimental portion of the project, the most basic form of an Artificial Neural Network, the perceptron, was implemented. The Perceptron was trained to learn to classify images of 0 or 1 for different fonts and tested to see how the size of input features, learning rate as well as amount of noise affected it.

2.1. Building the Perceptron

The Perceptron was coded in Python following the design of the primitive units. A perceptron uses a step function based on the linear combination of real valued inputs as shown in the equations 1-6 above. Since the perceptron here is trained to classify images of 0 and 1. The step function will output 1 if the linear combination outputs greater than the threshold, otherwise it outputs a 0.

With each example and iteration, the weight is updated according to the Perceptron Rule. To train the perceptron to find the right values for W, the following algorithm was used:

1. Assign random values to the weight vector
2. Apply the perceptron rule to every training example
3. Check if all training examples are correctly classified
    a. If all training examples are correctly classified, Quit
    b. Else, go back to Step 2.

The implementation of this algorithm can be seen in Perceptron.py python file under the function name "perceptron"

Processing the Inputs

The perceptron was trained to classify images of 0 and 1. For this project, 4 different font styles were used to train the perceptron.

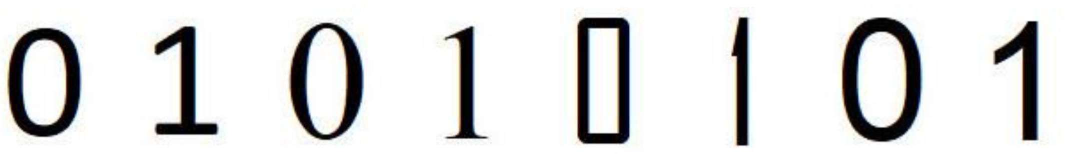


Figure 3. Different images of different font types used to train the perceptron

Each image was imported and converted into an array of values. Each element in the array represents a pixel of the image where the value 0 represents black while the value 1 represents white.

2.2. Training the Perceptron

The perceptron was trained with 8 different images for each image size, 150 pixels, 500 pixels, 1050 pixels and 6800 pixels. It was also trained over several iterations to see how long it would take to correctly classify all the training examples correctly. The following figure shows the number of iterations it took for the perceptron to classify all the training examples correctly.

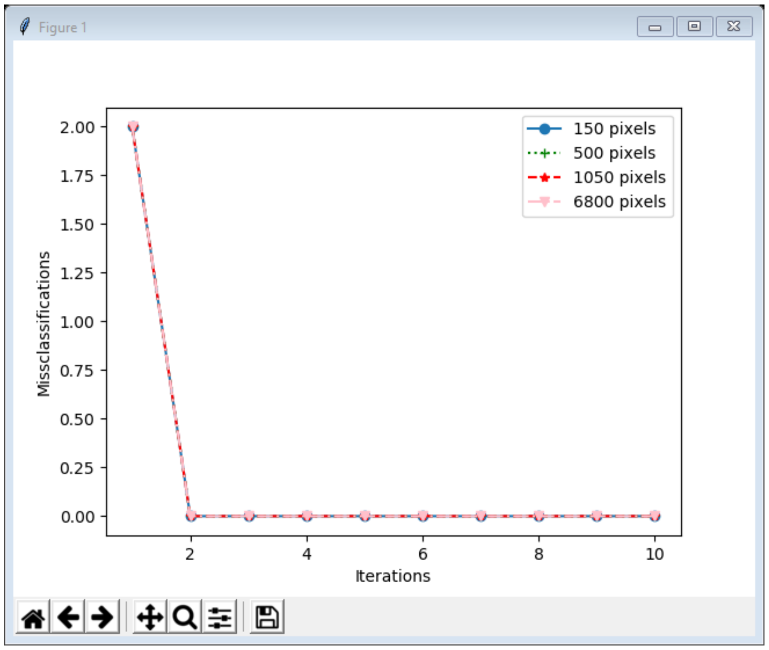
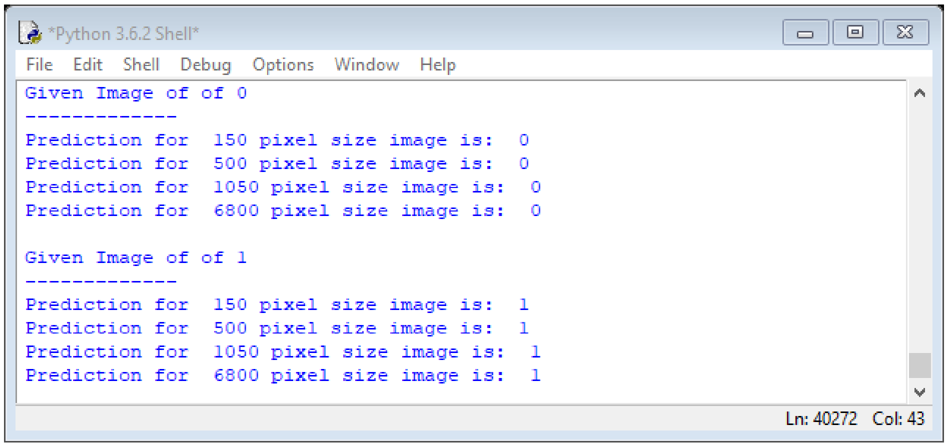


Figure 4. Iterations needed to train the Perceptron for different image sizes.

As seen above, for all 4 different sizes of the image, the perceptron took the same amount of iterations classify all the training examples. Since the perceptron only has to learn to classify between 2 types of classes, it does not take it long to learn the weights. As the size of the image increases, there are more pixels which means more features and weights that the perceptron has to update. Based on the figure above, the size of the image did not affect the number of iterations needed to train all the training examples correctly.

2.3. Testing the Perceptron

The trained perceptron was then tested with a fifth type of font that it has never seen before. The following figure shows the prediction of the perceptron after it has been trained for 10 iterations.



```
Given Image of of 0
-------------
Prediction for  150 pixel size image is:  0
Prediction for  500 pixel size image is:  0
Prediction for  1050 pixel size image is:  0
Prediction for  6800 pixel size image is:  0

Given Image of of 1
-------------
Prediction for  150 pixel size image is:  1
Prediction for  500 pixel size image is:  1
Prediction for  1050 pixel size image is:  1
Prediction for  6800 pixel size image is:  1
```

Figure 5. Perceptron classification output for a font it has never seen

As seen in the figure above, when given an image of 0, the perceptron correctly classifies the respective 0 images for each size. The perceptron also correctly classifies the images when given an image of 1 for each size.

2.4. Learning Rate effect

To see the effect of the learning rate on the perceptron, the perceptron was trained with varying learning rates from 0.1 to 1.0. In this test, the 6800 pixel size images were used for the training examples. The number of misclassifications for each learning rate with each iteration is show in the following figure.

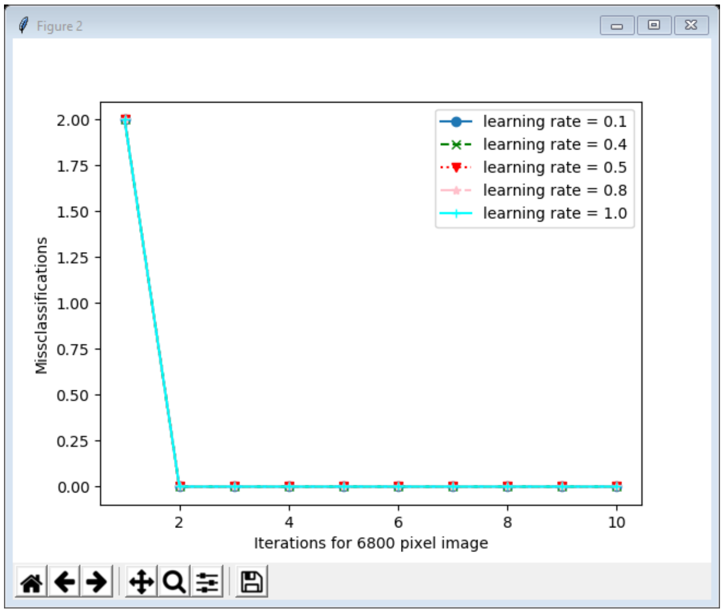
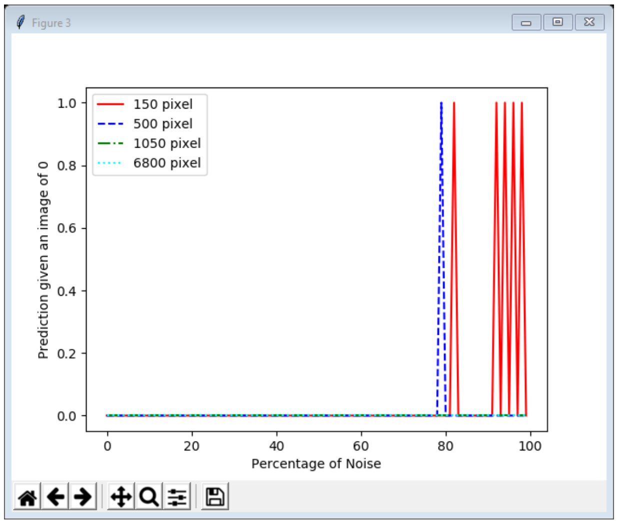


Figure 6. Number of misclassifications with different learning rates for the perceptron

As seen in the figure above, the learning rate did not affect number of misclassifications. There was neither an improvement or deterioration in the performance of the perceptron. This could be due to the simplicity of the images as well as the small variability between the different font types.

2.5. Noise effect

To test the effect of noise on the perceptron, after training the perceptron with the training example images, the perceptron was tested with noisy version of the training examples. The images were modified with varying percentages. To mimic noise, the value of the pixel is flipped. If it was a 0, it was changed to 1 and vice versa. The prediction of the perceptron is plotted against the percentage of the image that is noisy as seen in the following figure.



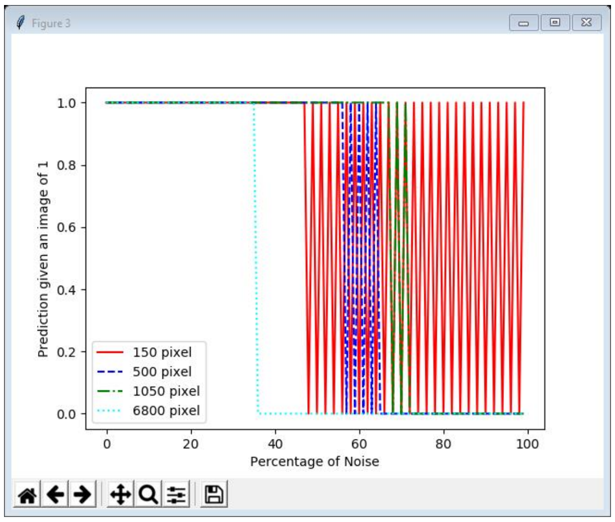
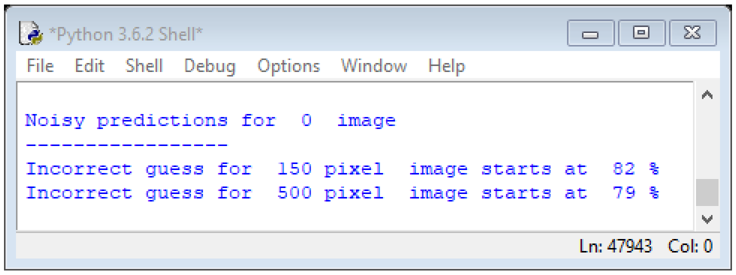
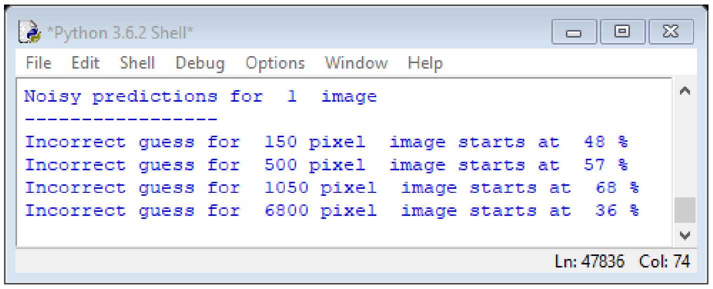


Figure 7 Perceptron prediction as a function of varying percentage of image that is noisy for a ‘0’ image (Left) and for a ‘1’ image (Right)

The following figure shows the percentage value of noise for the first occurrence of an incorrect guess by the perceptron.



```
Noisy predictions for  0  image
-----------------
Incorrect guess for  150 pixel  image starts at  82 %
Incorrect guess for  500 pixel  image starts at  79 %
```



```
Noisy predictions for  1  image
-----------------
Incorrect guess for  150 pixel  image starts at  48 %
Incorrect guess for  500 pixel  image starts at  57 %
Incorrect guess for  1050 pixel  image starts at  68 %
Incorrect guess for  6800 pixel  image starts at  36 %
```

Figure 8 Occurrence of incorrect guesses by the perceptron as a function of percentage of the image that is noisy given a ‘0’ image (top) and for a ‘1’ image (bottom)

From the figures above, the perceptron had incorrect guesses starting at 82% and 79% of noise in the ‘0’ image for the 150pixel and 500pixel respectively. While the perceptron was had incorrect guesses starting at 48%, 57%, 68% and 36% of noise in the ‘1’ image for the 150pixel, 500pixel, 1050pixel and 6800pixel respectively. The perceptron is capable of classifying the noisy training examples correctly with some minor noise. As the noise increased, the perceptron gets more confused how to classify them.

3. Conclusions

Perceptron is straightforward to implement and capable of classifying linearly separable data. For non-linear and complex data, Multilayer Perceptrons can be implemented. Different activation function can be used to suit the data needs.

The perceptron can classify the images when given an input image of 0 or 1. It took 2 iterations to train to correctly classify all the training examples. From this experiment, the size of the image did not factor how many iterations needed to train the perceptron and correctly classify all the data. When given a test example that the perceptron has not scene, the perceptron is able to correctly classify it.

When tested on the same image size but with varying learning rates, the perceptron took the same amount of iterations to train. Since the provided data is linearly separable, the perceptron is guaranteed to find the solution. Thus, when the learning rate is changes, it only scales the weight. However for other neural networks, the learning rate will play a huge role as those implementations as a smaller learning rate will take longer time for convergence and too large of a learning rate can cause it to overshoot the minima.

When tested on the images with varying noise percentages, the perceptron is capable of handling some amount of noises. Generally, the more noise there is, the more confused the perceptron gets. Since the perceptron performs better in the 0 case when it has noise in comparison to the 1 case, it is difficult to conclude a pattern. Further testing with a larger set of training data or more different types of numbers can be done to help draw a better conclusion.

References

1395550283894582. “Activation Functions: Neural Networks – Towards Data Science.” *Towards Data Science*, Towards Data Science, 6 Sept. 2017, towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Hardesty, Larry, and MIT News Office. “Explained: Neural Networks.” *MIT News*, 14 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

Nicholson, Chris V., and Adam Gibson. “A Beginner's Guide to Multilayer Perceptrons.” *A Beginner's Guide to Multilayer Perceptrons - Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM*, deeplearning4j.org/multilayerperceptron.

“Perceptrons - the Most Basic Form of a Neural Network.” *Applied Go*, 9 June 2016, appliedgo.net/perceptron/.

1395550283894582. “What the Hell Is Perceptron? – Towards Data Science.” *Towards Data Science*, Towards Data Science, 9 Sept. 2017, towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.