ANN Visualization and

Image Recognition Demo

Casey Johnson

12/9/19

Introduction

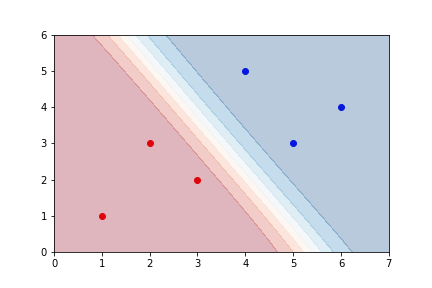
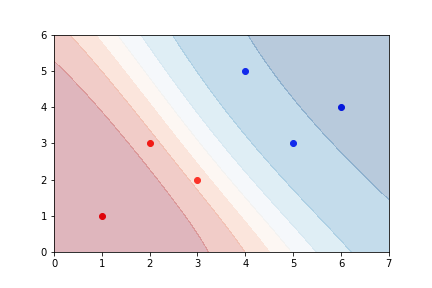
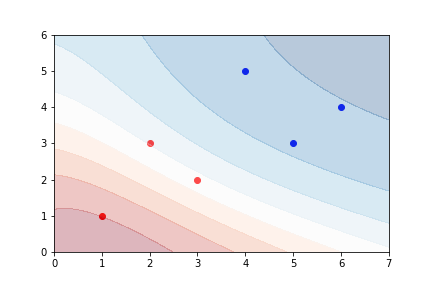
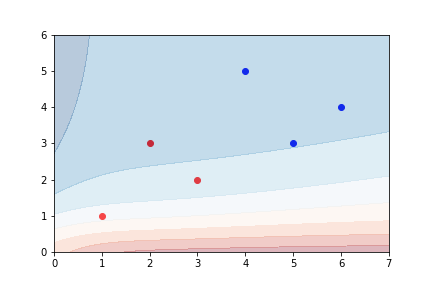
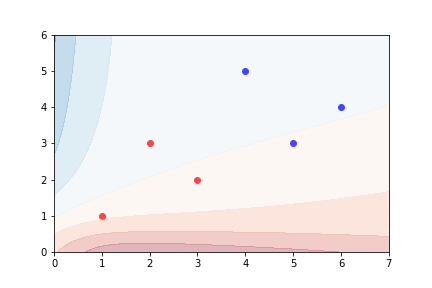
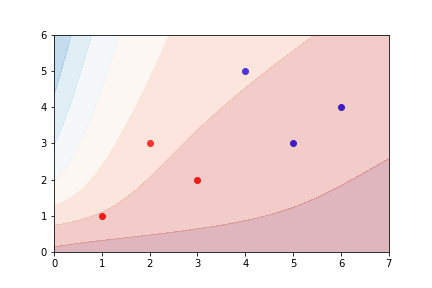
There are many Python libraries which allow programmers to create their own Artificial Neural Network (ANN) in a handful of lines such as “scikit-learn” and “keras”. While these libraries work just fine for analyzing data, they function as a black box of the general mathematical procedures used to build and train ANNs. The goal of this project was to create an ANN from scratch with the intent of building a more intuitive sense of how ANNs work. The two subtasks of this goal are 1) create a visual which encapsulates the learning process of an ANN and 2) apply an ANN to image recognition involving hand-written numbers.

The Structure of the ANN

All of the features of my ANN are contained inside of a class called “NN”. Upon declaring an NN object, the layers of the object are initialized according to a “layer\_sizes” parameter passed into the constructor. Each layer is a numpy array containing the values of each neuron within that layer. By default, the neurons are left uninitialized until data is fed into the NN object. Along with the layers, the weights and biases of the NN object are initialized with random values. Each layer is a dense layer which means that every neuron from one layer connects with every neuron in the next layer. The class has two main functions used to train itself and predict data. The first function is the feed\_forward(X) function which takes in an array of data and fills in every neuron with a value depending on the weights and biases. Since the weights and biases are initialized with random values, the neurons in the last layer (the output layer) will not match the labels of the inputted data. After the feed\_forward(X) function is called, the NN can use the back\_propagation(y, alpha) function to compute a cost function and take a step towards a minimum on this function with respect to the weights and biases. As a result of this process, the weights and biases are modified in order to reduce the cost function and strive towards a more accurate output. The activation function used for the NN is a sigmoid function. The NN also includes a train function which calls the feed\_forward and back\_propagation functions a given number of times in an alternating fashion. This has the effect of training the entire NN in one continuous stream of commands such that it is “fully” trained when the function terminates. Finally, the NN includes a predict function which calls the feed\_forward function for multiple inputs of data and returns an array of each resulting output.

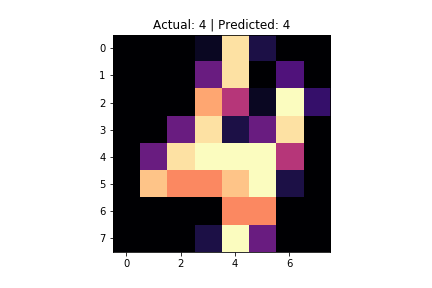
The Visual

Once the NN class was finished, I could then proceed with visualizing the training process. The data used in the visualization seen in Figure 1 is simply a hand chosen set of data points under no context. The idea of the visual was to show how the NN would evolve to learn how to classify two separate regions using labelled points (labelled as either red or blue). The sequence of images corresponds to some iteration of the feed\_forward-back\_propagation loop. As is shown in Figure 1, each iteration corresponds to a more accurate separation of the red and blue points.



(Figure 1) The top left frame represents the random starting point in which the ANN classifies red and blue points. The frames coming after the initial frame are future iterations of the training process. At the bottom right frame, the ANN stopped training and has found a reasonable solution to classifying red points from blue points.

Image Recognition

The final task of the ANN was to accurately read off the value from hand-written digits. The images of these digits used to train the ANN were 8 x 8 arrays of values ranging from 1 – 16 and were available from the scikit-learn library. The ANN was setup to have 64 neurons in the input layer each for the values which make up the image, and the final layer contained 10 neurons each to represent the probability that the image represented one of the ten possible digits. I also included two hidden layers with 20 neurons each and normalized the values making up the image to be between 0 and 1 in order to increase the probability of the ANN reaching a lower cost value. The results are as follow:

(Figure 2) Example of an image the ANN tested on. Here the ANN correctly predicted the image as a 4.

Number of training images: 1000

Number of testing images: 797

Percentage of images predicted correctly: 92.97%

Conclusion

From these results, I have successfully created a simple, functioning ANN. Not only can my ANN be used to recognize numbers to some degree, but the visualization of the ANN demonstrates the mechanism at work for training the ANN. Although there are more detailed topics on the matter such as the architecture of the neural network, activation functions, and optimization methods, this project at least demonstrates the general procedure behind ANNs.