In this project, I implemented a multi-layer perceptron and used it to classify points on a Cartesian plane drawn from two non-intersecting Archimedean spirals. The MLP takes two inputs, the x coordinate and y coordinate of the point to be classified, and outputs the probability that the point is from one of the spirals. After some trial and error, I settled on using 8 hidden layers with 15 hidden nodes. And for the activation function for my hidden layers I found that the ReLu activation function was causing the output at the final hidden layer to almost always be 0. I tried different ways of initializing the weights and biases for the MLP like initializations where positive weights are more likely to be chosen than negative ones. However, I wasn't getting much success so I decided to switch to the exponential linear unit. That way negative outputs to the next hidden layer wouldn't just get dropped to 0. For the activation function for the output layer, I used the sigmoid function with the idea that since the output of the neural network would be between 0 and 1, it would be able to more quickly converge since the labels are 0 and 1.

To determine the functional form for my classifier, my first approach was to find the values of x and y that make the classifier output 0.5. If we for simplicity say that the classifier has one hidden layer with hidden activation function g and output activation h, then we want to find the points (x, y) such that h(g(W2*(W1*[x, y] + b1) + b2)) = 0.5.

In our case, h is the sigmoid function so h^-1(0.5) = 0 and we get that g(W2*(W1*[x,y] + b1) + b2) = -b2. b2 in our case is a scalar and g is strictly increasing and therefore invertible. So we have W2*(W1*[x,y] + b1) = g^-1(-b2). But now we see a problem with this approach and it's that if the weight matrices aren't invertible then we can't find the decision boundary curve this way. Additionally, since g is a piecewise function for us, with more layers, the expression for our boundary curve will become more and more complicated.

We could instead try to find all of the points that the classifier gave a probability between .5-epsilon and .5 + epsilon. We could then convert the points to polar coordinates and train a 1D linear regression model to find the curve of best fit. The decision boundary I think is just another spiral so we should be able to just find the "line of best fit" in polar coordinates to get the function.

With an epsilon of 0.1, I obtained the decision boundary shown below. Unfortunately, what I didn't realize was that for the spiral dataset, there are two decision boundary curves as can be seen in Figure 1. So I added some if statements to take out the points corresponding to one of the curves (in polar coordinates so it was a lot easier to filter out since it was just two lines). After doing that, I was able to get a line of best fit that pretty closely matched one side of the boundary for my model as can be seen in Figure 2.

But this still wouldn't allow me to get the decision boundary for my model because it's not a function. Instead, I can only say that the boundary points are the points that roughly satisfy either r = 1.112*theta + 7.568 or r = .821*theta + 4.266.

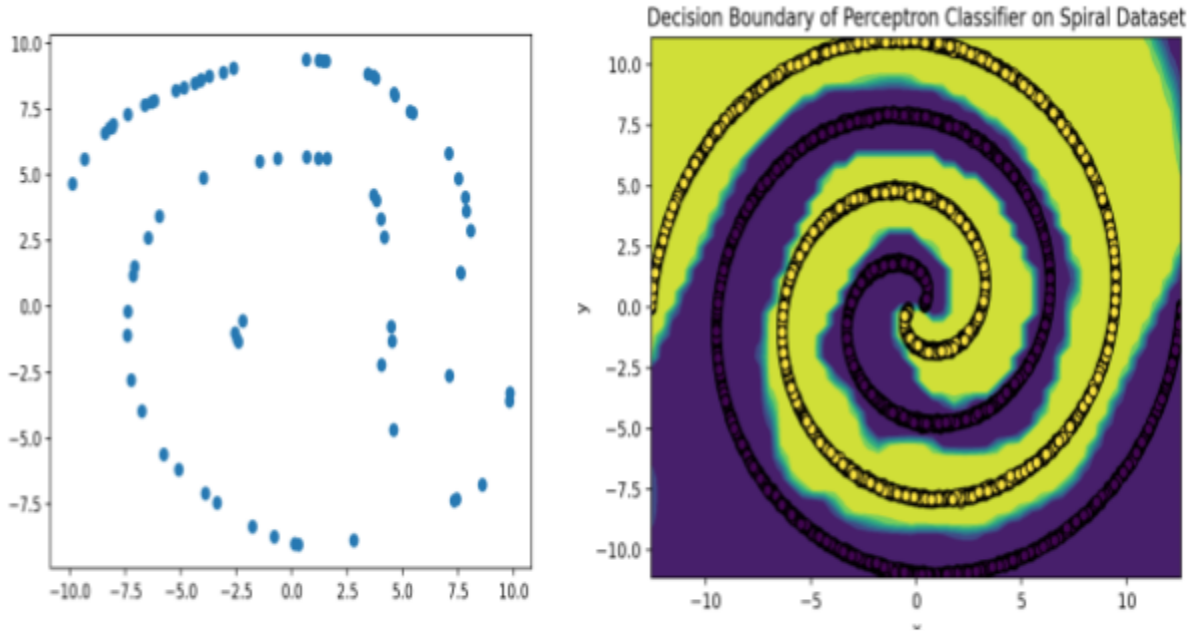

Figure 1: Decision Boundary of Classification Model

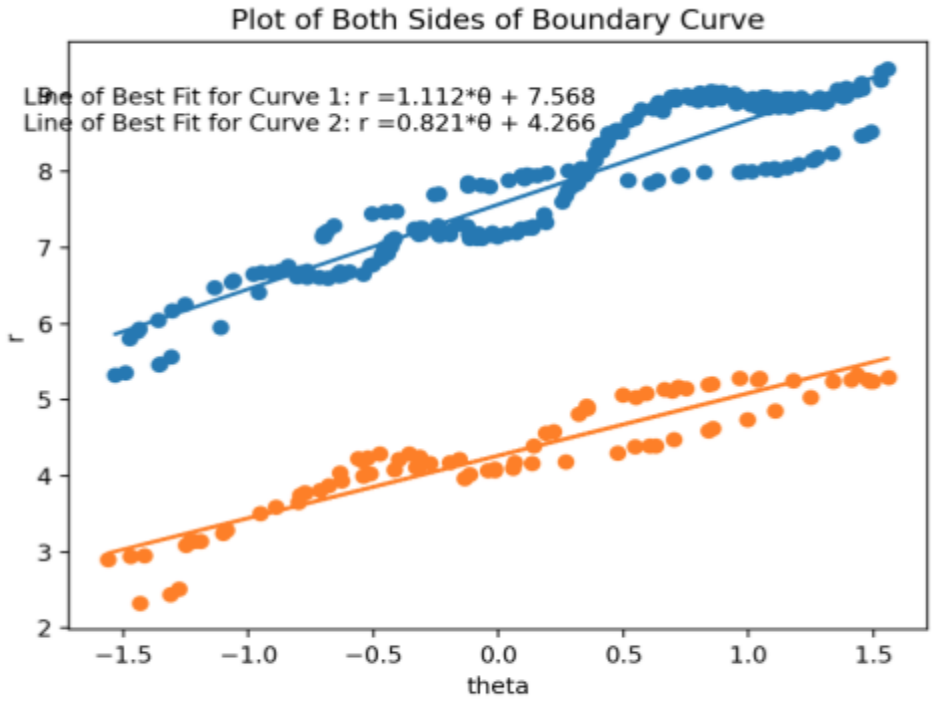

Figure 2: Line of Best Fit for Half of Boundary Curve