ITP 259, Fall 2023

Homework 4
30 points
Neural Networks (NN)

1. Tensorflow playground (10 points)

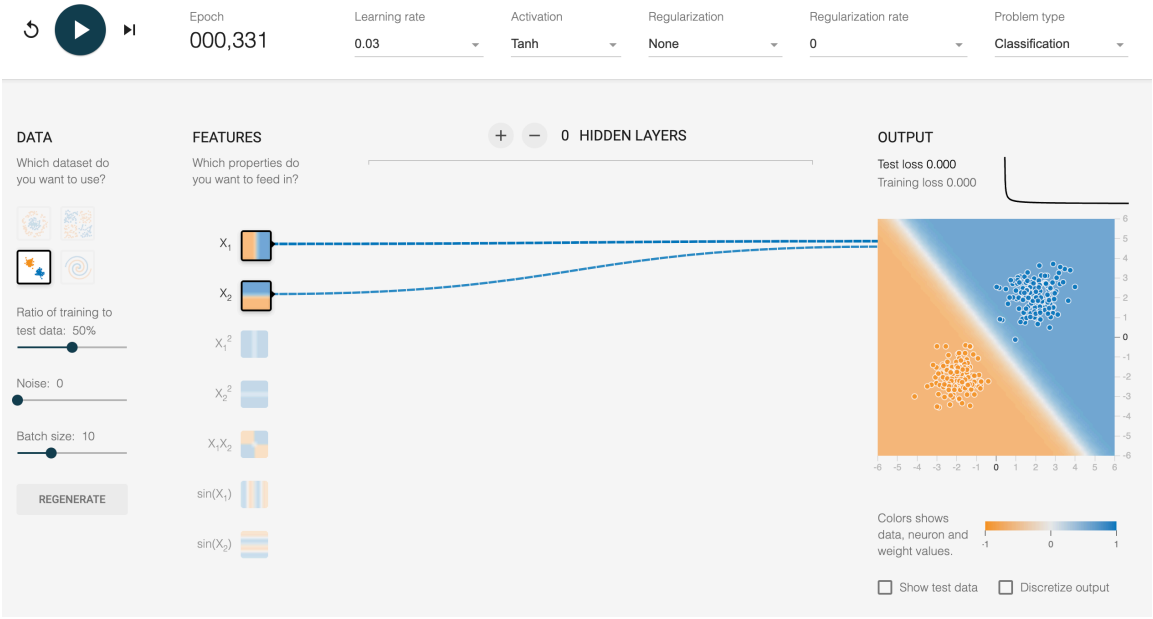
Go to https://playground.tensorflow.org/ and tinker with various NN scenarios.

Explore various parameters for NNs to answer the following questions.

Insert screenshots to support answers for this problem.

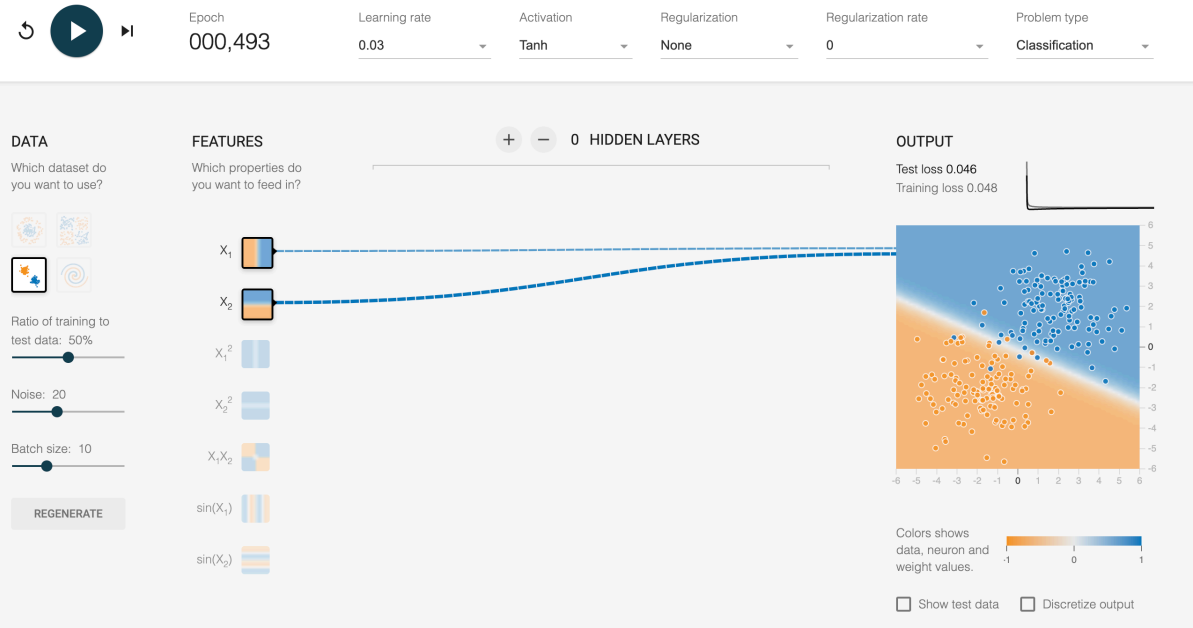
1.1. What is the simplest classification neural network can you build (for what dataset?) in this playground? What is the test loss? (3)

This simplest classification neural network you can build is one that linearly splits a Gaussian distribution with zero noise. You can do this classification with zero hidden layers. In only 331 epochs, the network was able to achieve a training loss and test loss of 0.



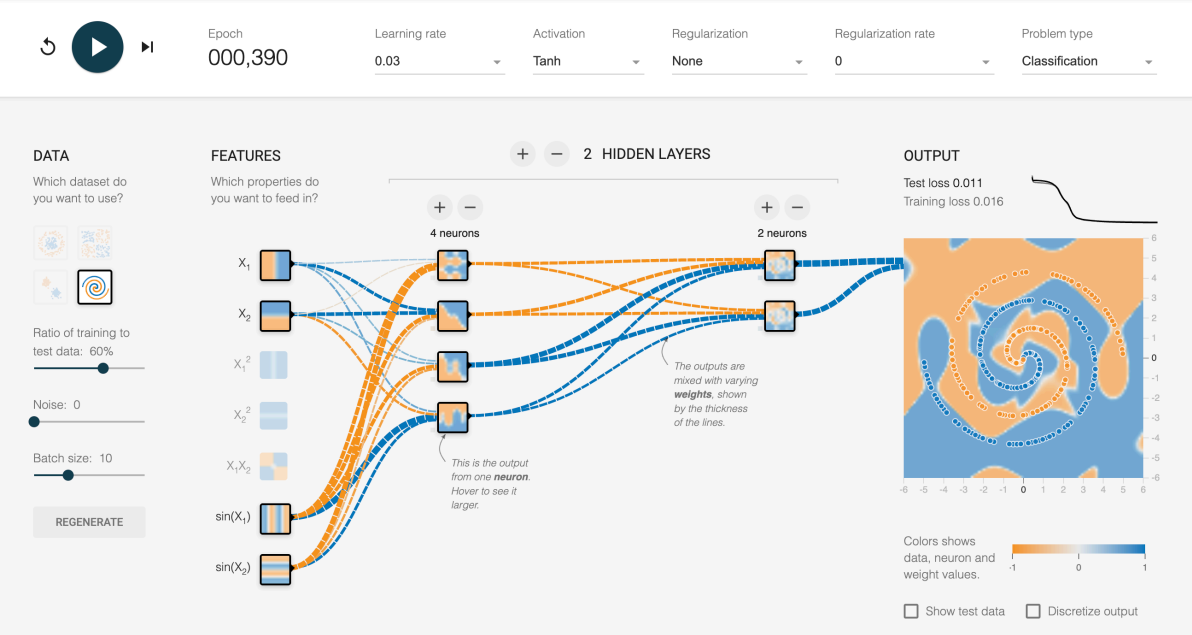
1.2. Add *noise* of 20 to the model in 1.1 How does the output change? Test loss? (2)

After adding noise of 20 to the previous model, it was no longer able to achieve a test loss of 0. After 493 epochs, the test loss was 0.046 and the training loss was 0.048.



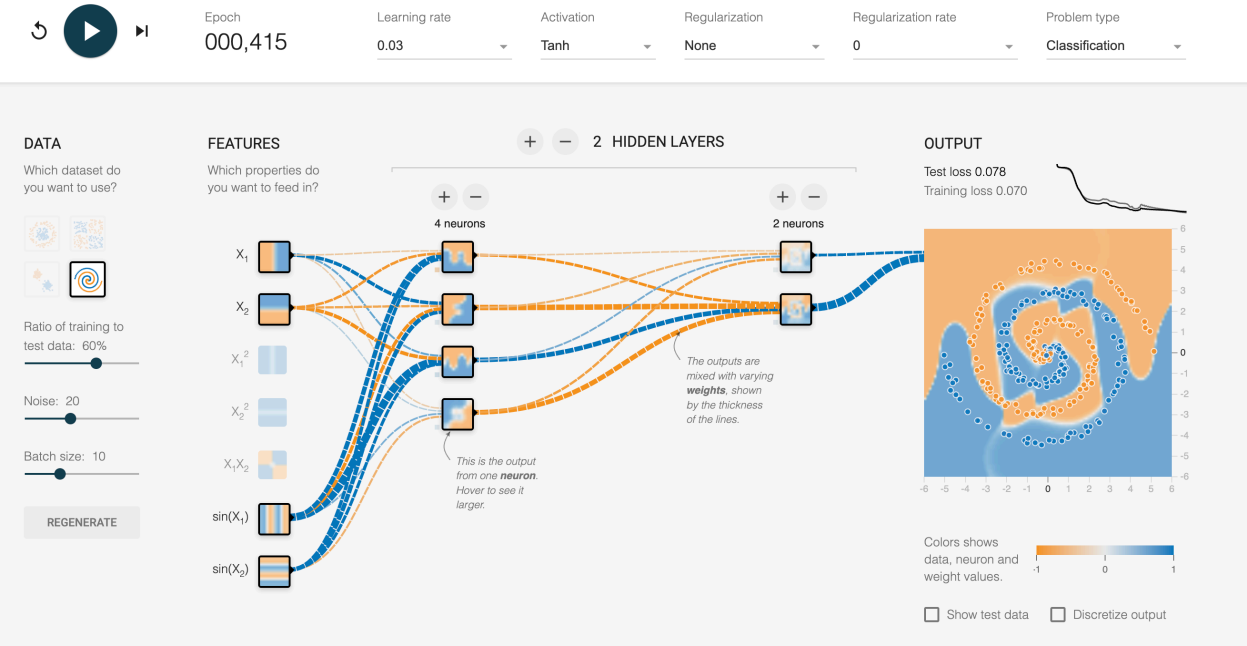
1.3. Which classification dataset do you think is the most difficult to train? Why? Show what parameters you chose to train it. What was the test loss? (3)

The dataset that is the most difficult to train is the spiral distribution since it can not be separated by a simple circle, line or lines, but rather needs a more complex function, feature values and network to separate the classes. In order to minimize loss, I added the features sin(X1) and sin(X2), used 2 hidden layers, the first with 4 neurons and the second with 2. After 390 epochs, this network was able to achieve a test loss of 0.011 and a training loss of 0.016.



1.4. Add *noise* of 20 to the model in 1.3. How does the output change? Test loss? (2)

After adding noise of 20 to the previous model, the output still did a pretty good classification but had a slightly higher error: test loss was 0.078 and training loss was 0.070 (after 415 epochs).



2. MLPClassifier Neural Network (20 points)

Only submit the Python notebook from Colab or the Python file for this problem. No need for screenshots for this problem.

The goal is to train a neural network to do *binary classification* of *spirally* distributed 2-dimensional data.

2.1. Generate *x, y* coordinates of spirally distributed blobs in two colors. See figure below. *You can search for code online to do this.* (2)

2.1.1. Note that the spirals should complete at least one full turn.

2.1.2. Add some noise to *x* and *y*.

2.1.3. The equations for the spirals are:

$$\theta \, should \, vary \, between \, 0 \, and \, 2\pi.$$

For the first spiral,

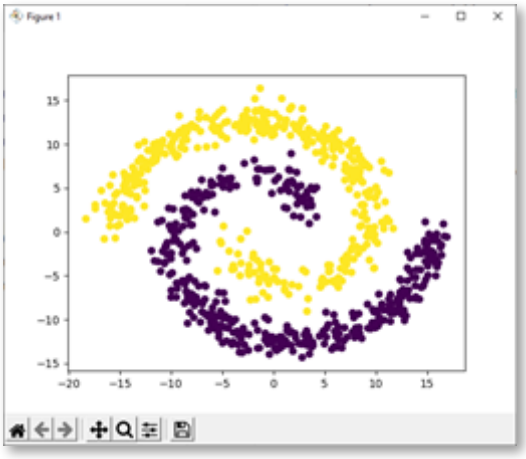
$$r = 2\theta + \pi$$
$$x = r * \cos\theta$$
$$y = r * \sin\theta$$

For the second spiral,

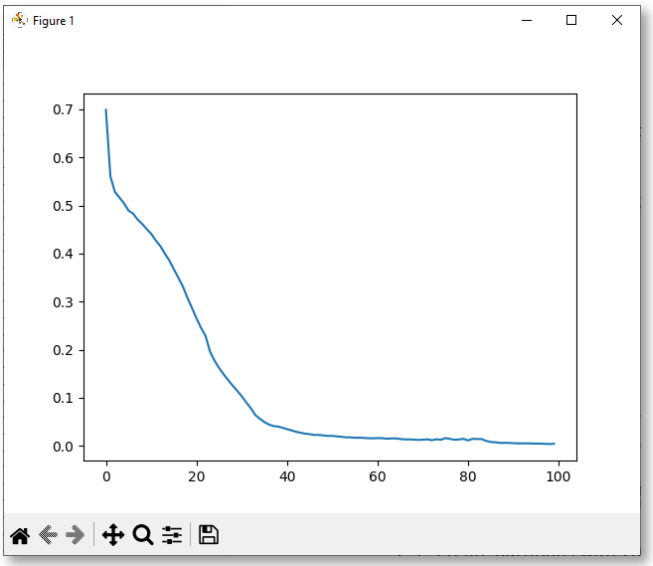
$$r = -2\theta - \pi$$

$$x = r * \cos\theta$$
$$y = r * \sin\theta$$

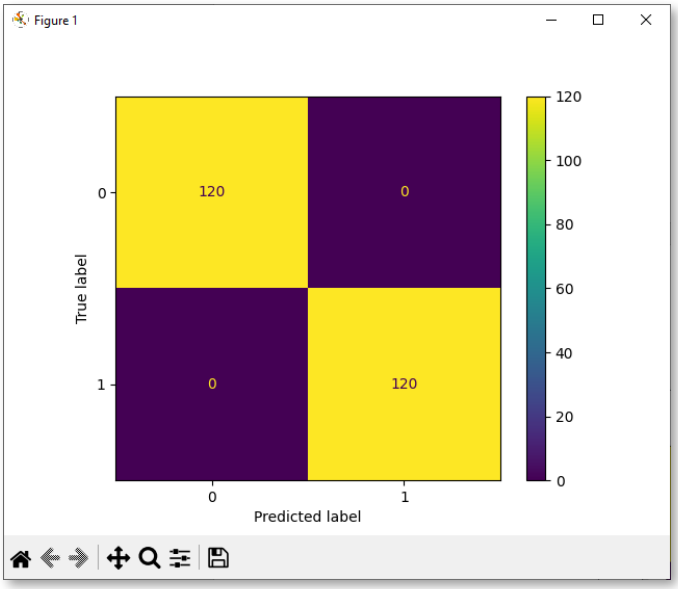
2.2. Display a *scatter* plot of the *x* and *y* coordinates using the *label* as color. Label is the spiral number such as 0 and 1. You may use any color map i.e., the colors corresponding to 0 and 1. (2)



2.3. Create *partitions* with 70% train dataset. *Stratify* the split. Use *random state* of 2023. (2)
2.4. Now train the network using *MLP Classifier* from scikit learn. The parameters are your choice. (2)
2.5. Plot the loss curve. (2)



2.6. Print the *accuracy* of the test partition (2)
2.7. Display the confusion matrix (2)



2.8. Plot the *decision boundary* (along with the original spirals). The decision boundary is the line where samples of one class are on one side and samples of another class are on the other side. (6)

Hint:

a. To plot the decision boundary, create a mesh of x and y coordinates that cover the entire field (e.g., -20 to 20 for both x and y coordinates).
b. You can make the mesh points 0.1 apart. So, you will have a 400x400 mesh grid.
c. Then reshape the meshgrid to a dataframe that has two columns and 160000 rows (each row is a mesh point).
d. Then classify each point using the trained model (*model.predict)*
e. Then plot both the original data points (spirals) and the mesh data points. This generates the decision boundary as shown below (green vs light blue). Use *color maps* of your choice.