Name: Nathan Dang

DSCI 225 – Applied Machine Learning

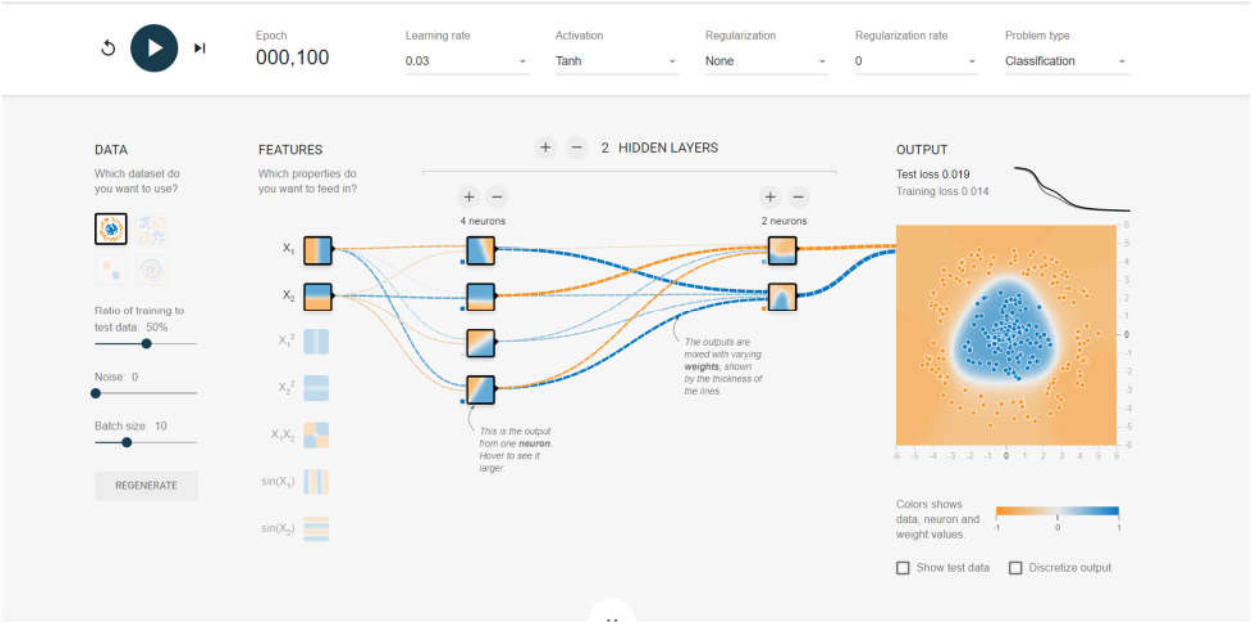
Date: 03/28/2020

Instructor: Prof. Basye

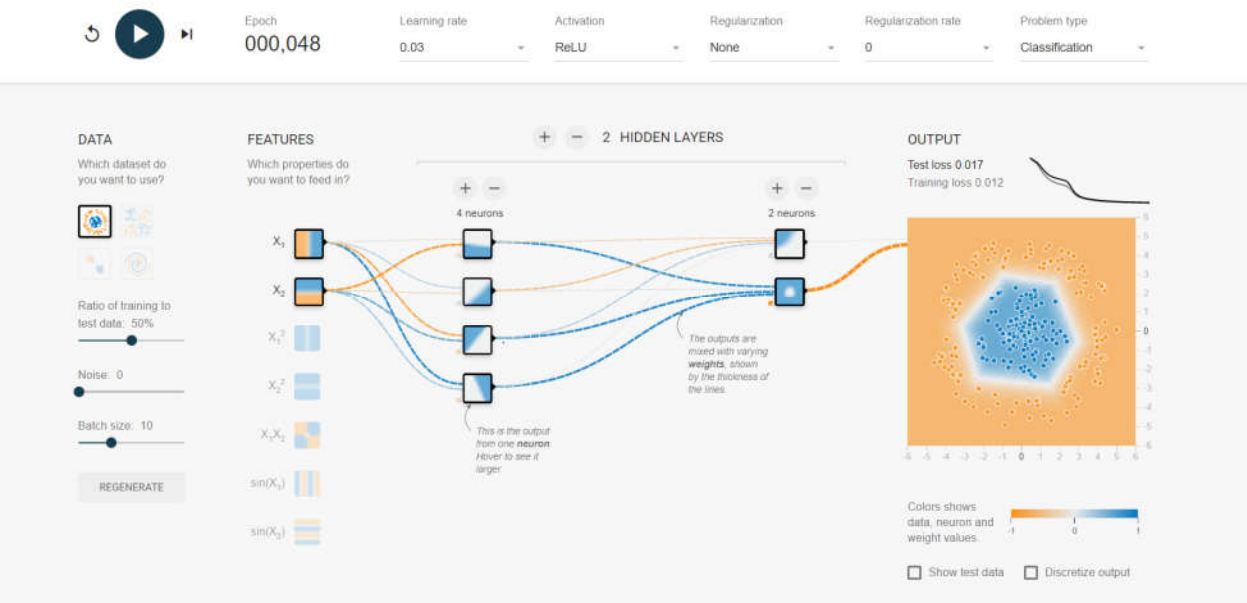
Lab 5 - Exploring Multilayer Perceptrons

Exercise 1, Chapter 10

1a. It took around 100 epochs to get to a good solution for the classification problem. As we can see from the picture below, the four neurons in the first layer learned simple patterns which are pretty much linear, and the two neurons in the second layer learned to combine the simple patterns from the first layer to create more complex patterns.

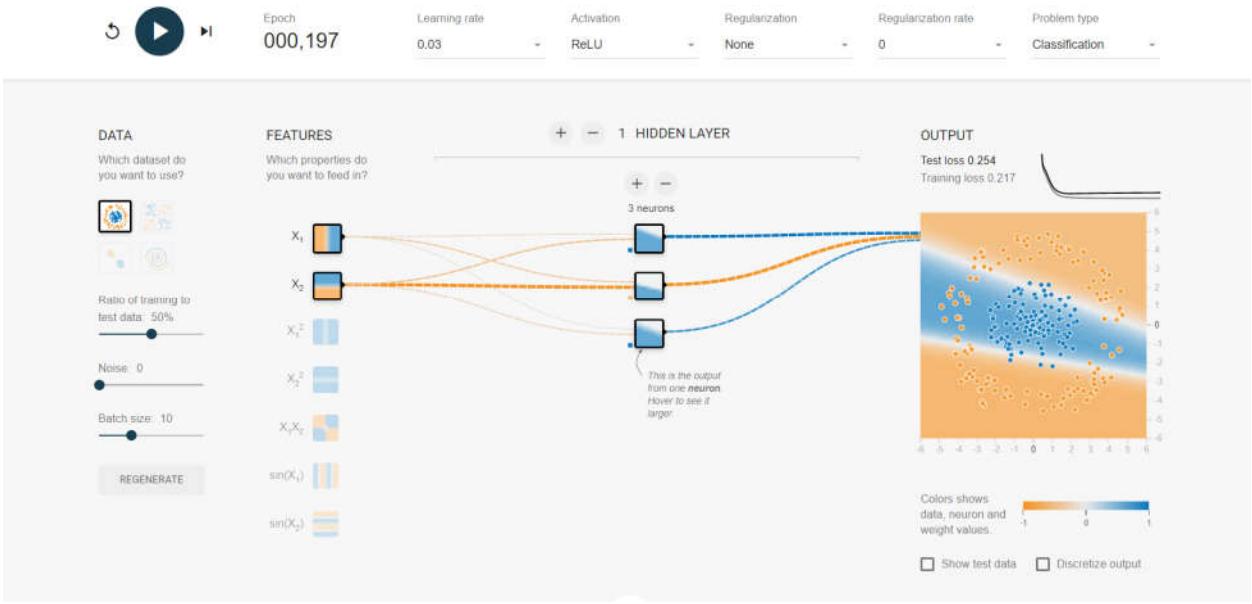


1b. After replacing the tanh function with ReLU, it only took about 45 to 50 epochs to reach a good solution for the classification problem which is twice as fast compared to the previous part. Clearly, the boundaries are linear (in this case, a hexagon) owning to the nature of ReLU.

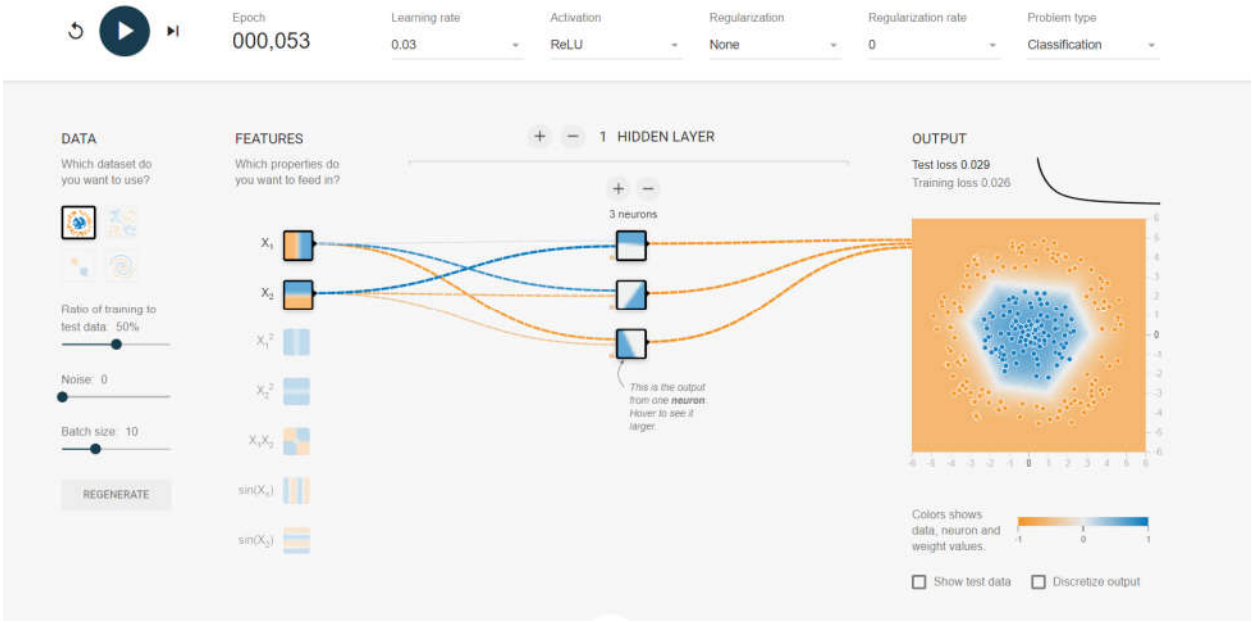


1c. Modify the network architecture to have just one hidden layer with three neurons and train for 6 times as listed below:

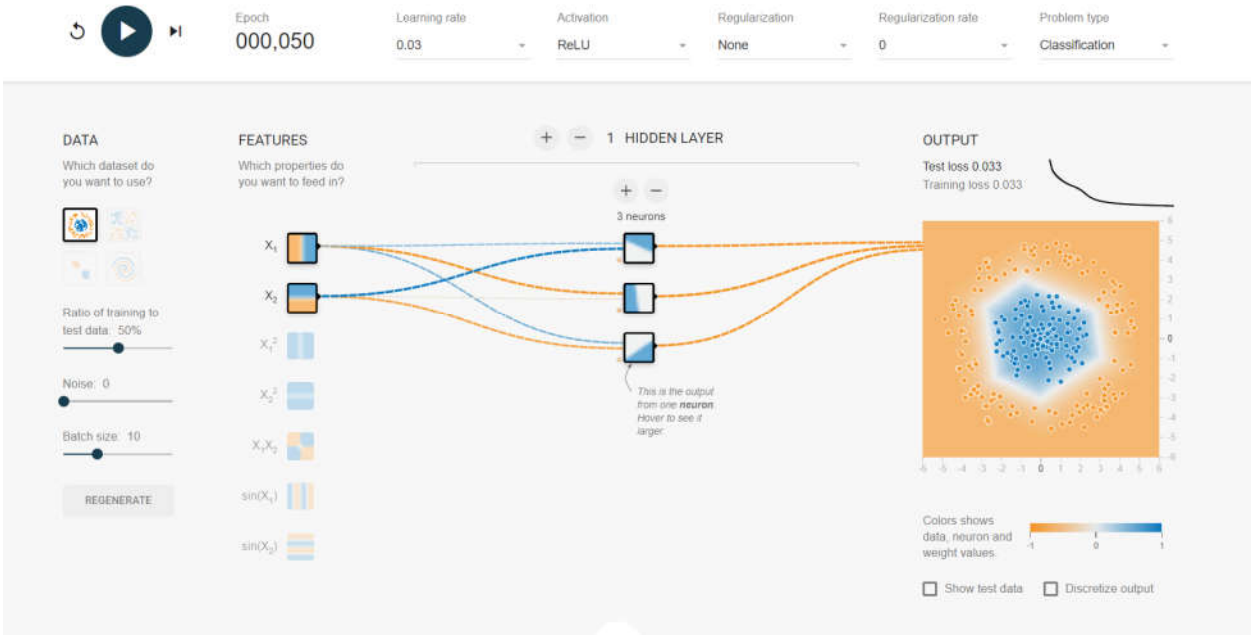
- Train no.1:



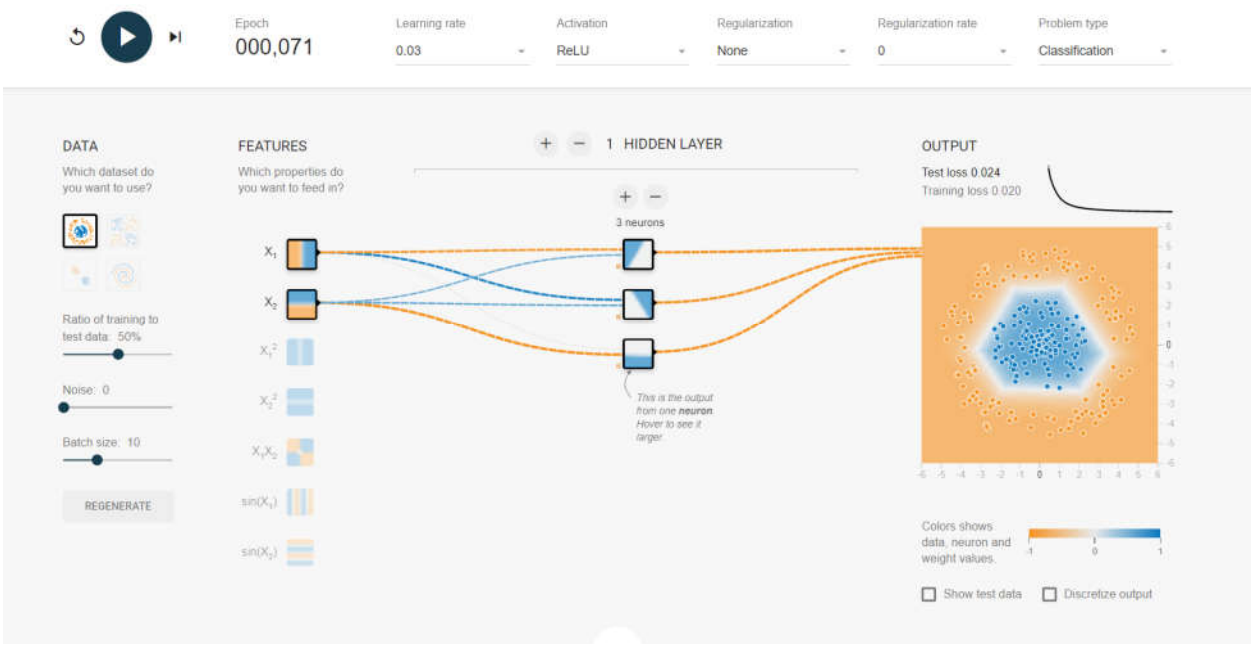
- Train no.2:



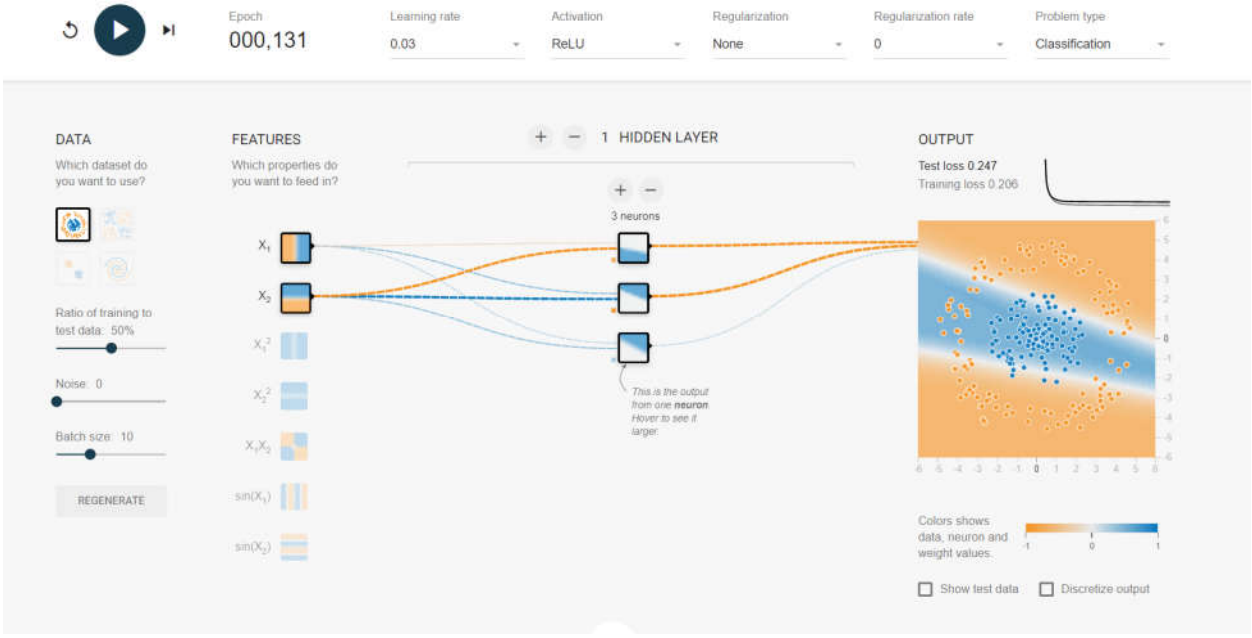
- Train no.3:



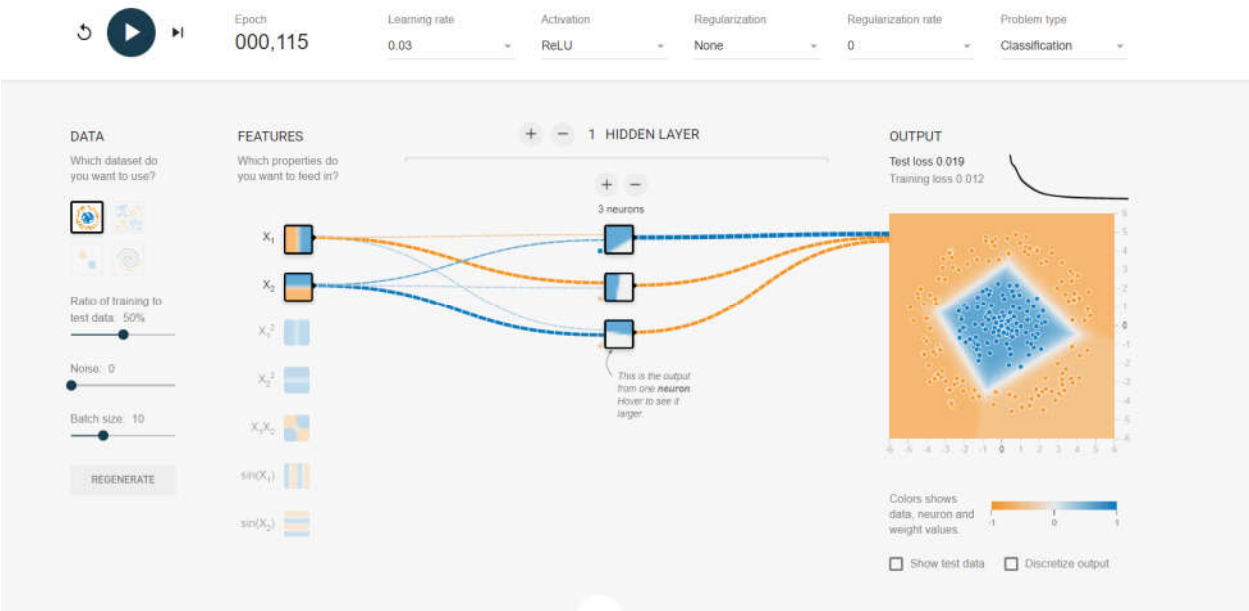
- Train no.4:



- Train no.5:



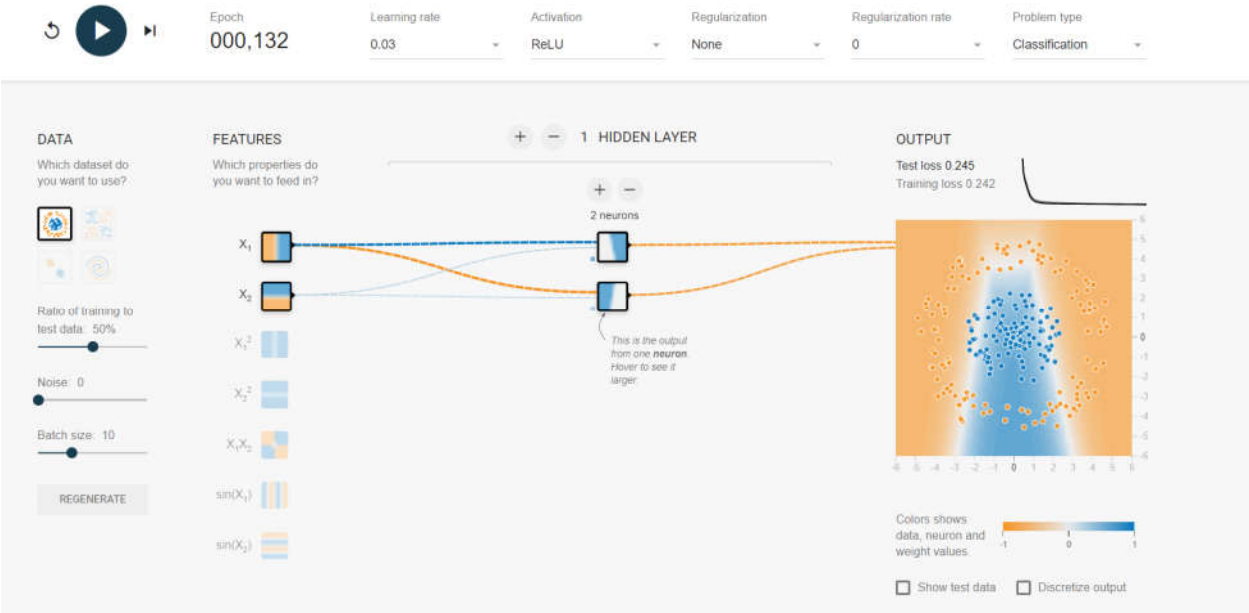
- Train no.6:



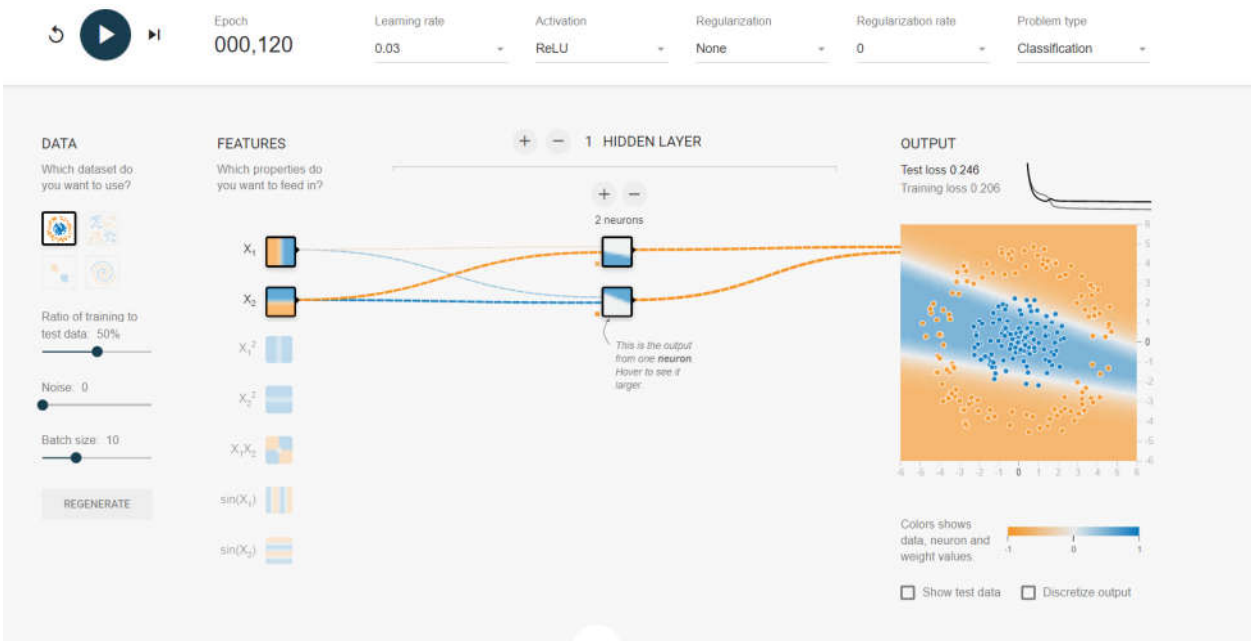
- **Observation:** the first and the fifth training attempt got stuck in a local minimum which has proven the fact that this model might get stuck at some points. Also, the last training attempt took longer to reach a good solution compared to the second, third, and fourth attempt showing that the training times vary.

1d. Remove one neuron to keep just 2 and train the model 3 times as shown below:

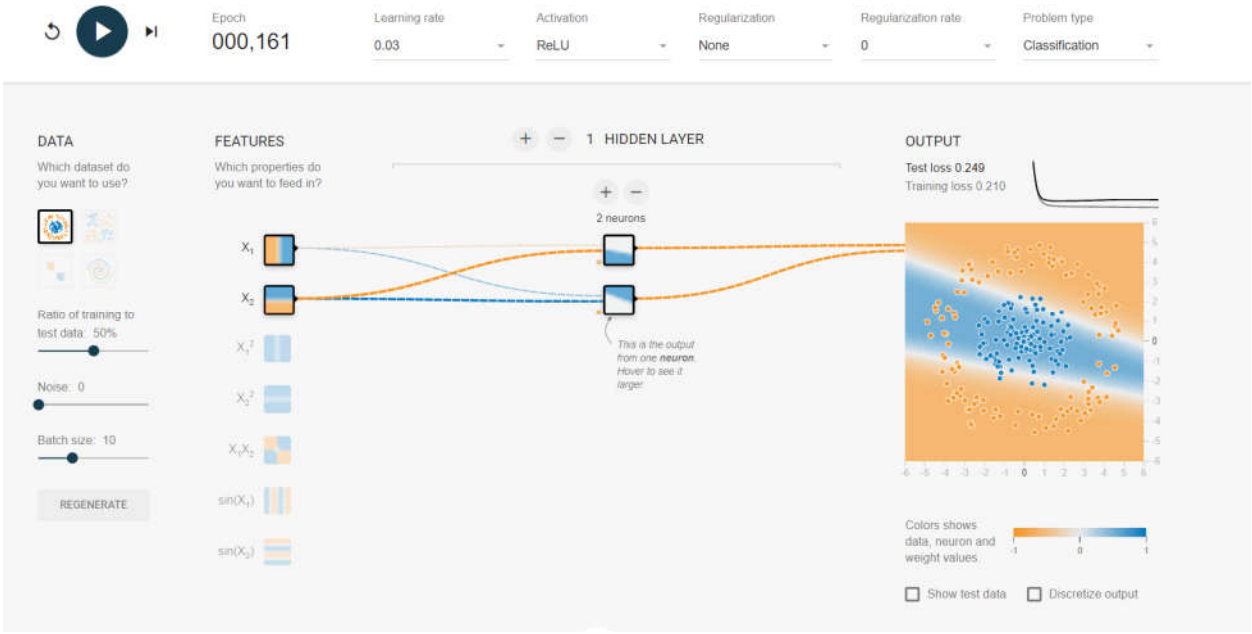
- Train no.1:



- Train no.2:



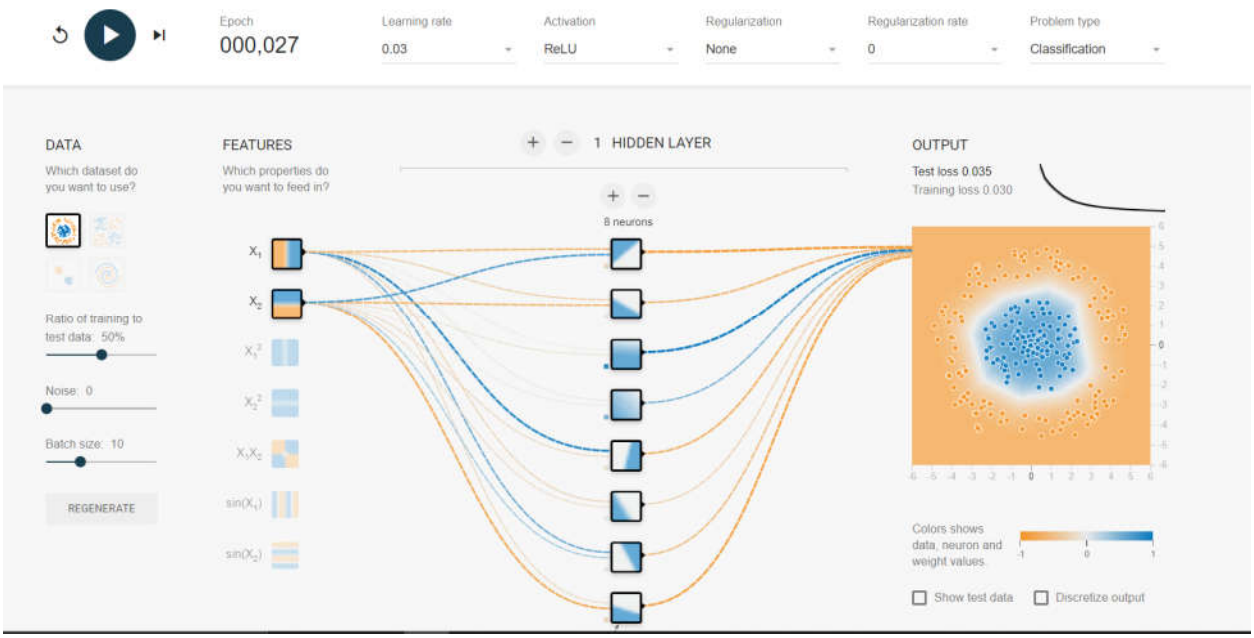
- Train no.3:



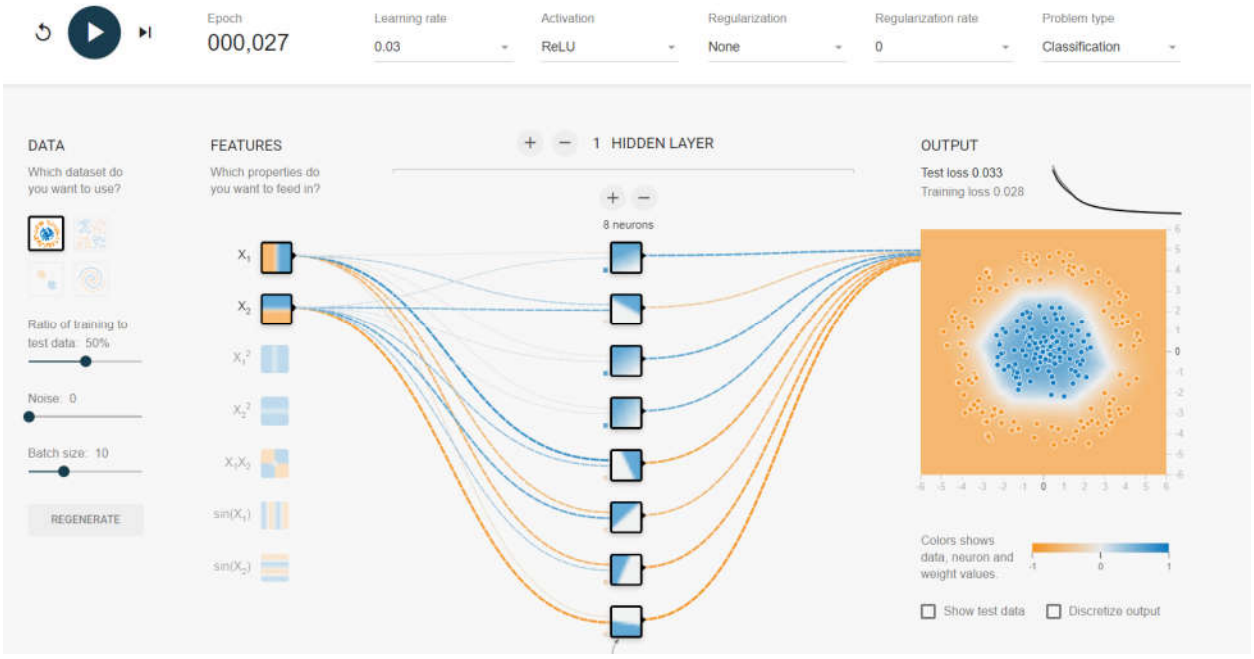
- **Observation:** as expected, the neural network is now incapable of finding a good solution, even if it is trained multiple times. Specifically, once the network has a very small number of neurons, i.e. too few parameters for training, it will underfit the training set and thus, fail to produce a good model that generalizes well on the data.

1e. Set the number of neurons to 8 and train the network 4 times as shown below:

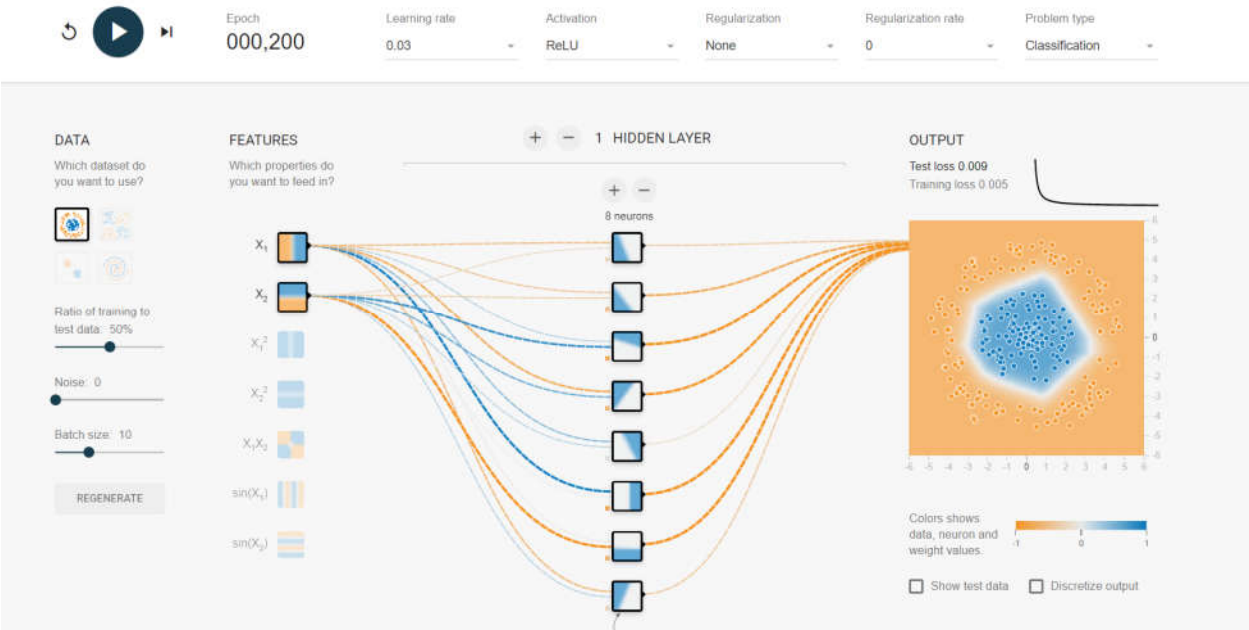
- Train no.1:



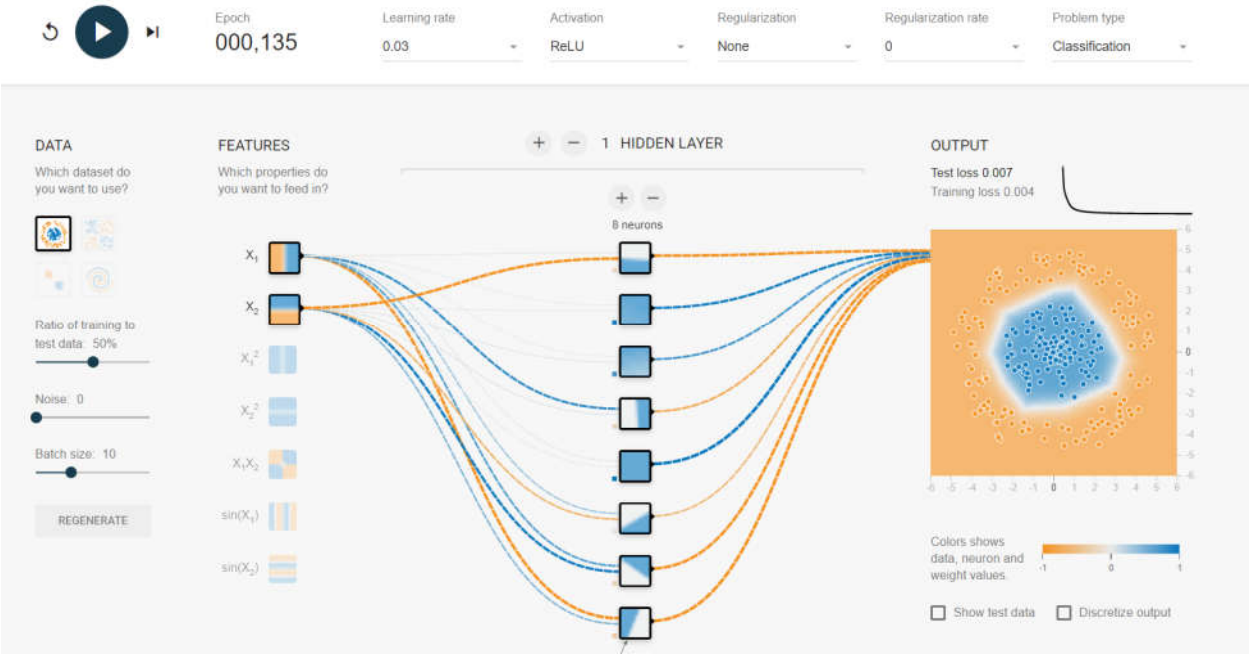
- Train no.2:



- Train no.3:



- Train no.4:



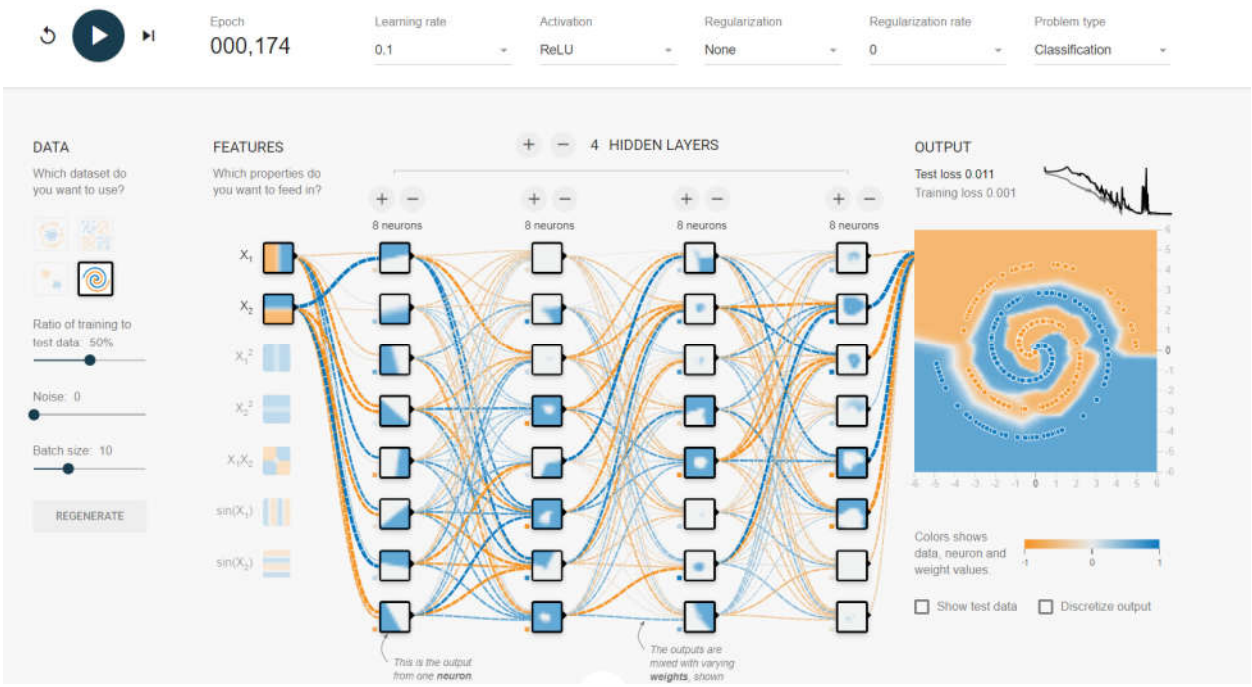
- **Observation:** as expected, it did not take long for the neural network to reach a good solution (about 27 epochs) as shown in the first two training attempts and it did not get stuck either (I let the last two attempts run a little bit longer just to make sure.

1f. Change the dataset to be the spiral (bottom right dataset under “DATA”). Change the network architecture to have 4 hidden layers with 8 neurons each.

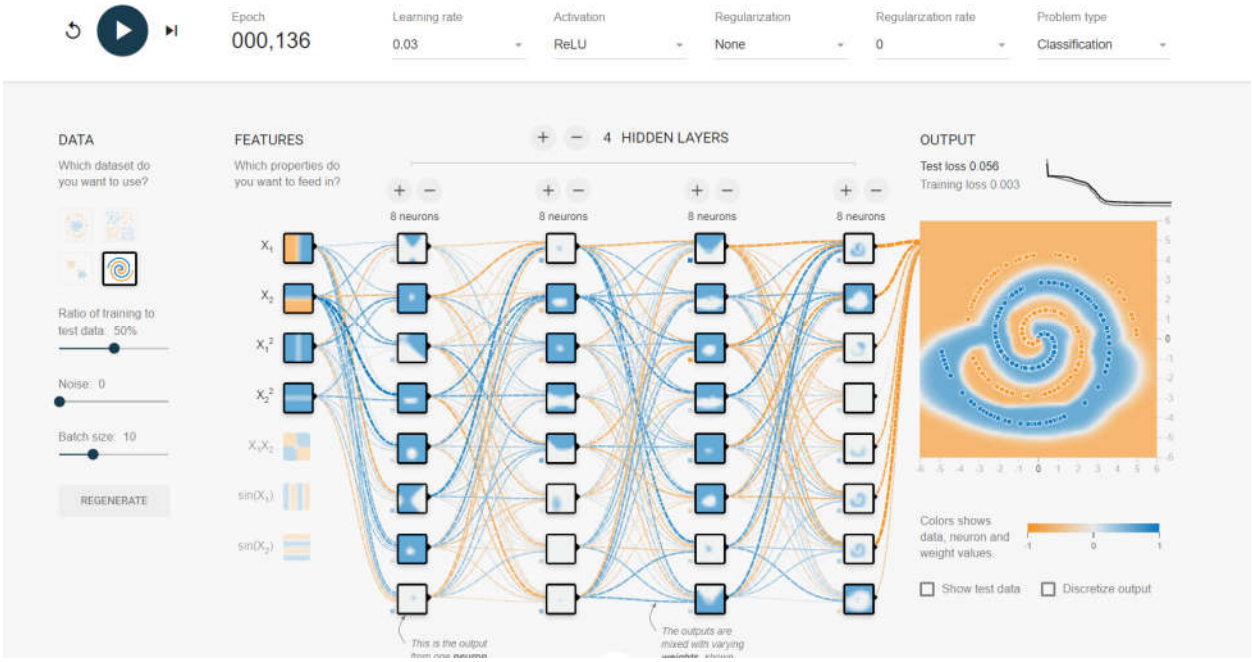
- **Observation:** as expected, it took longer for the neural network to reach a good solution compared to the other trainings that I did previously. Also, looking at the output graph, we can see that it fluctuates tremendously showing that it got stuck on the plateau for a period of time before actually settling down.

1g. I modified the setup of 1f in a few different ways and compared the results with that of 1f (i.e. I used 1f as the baseline).

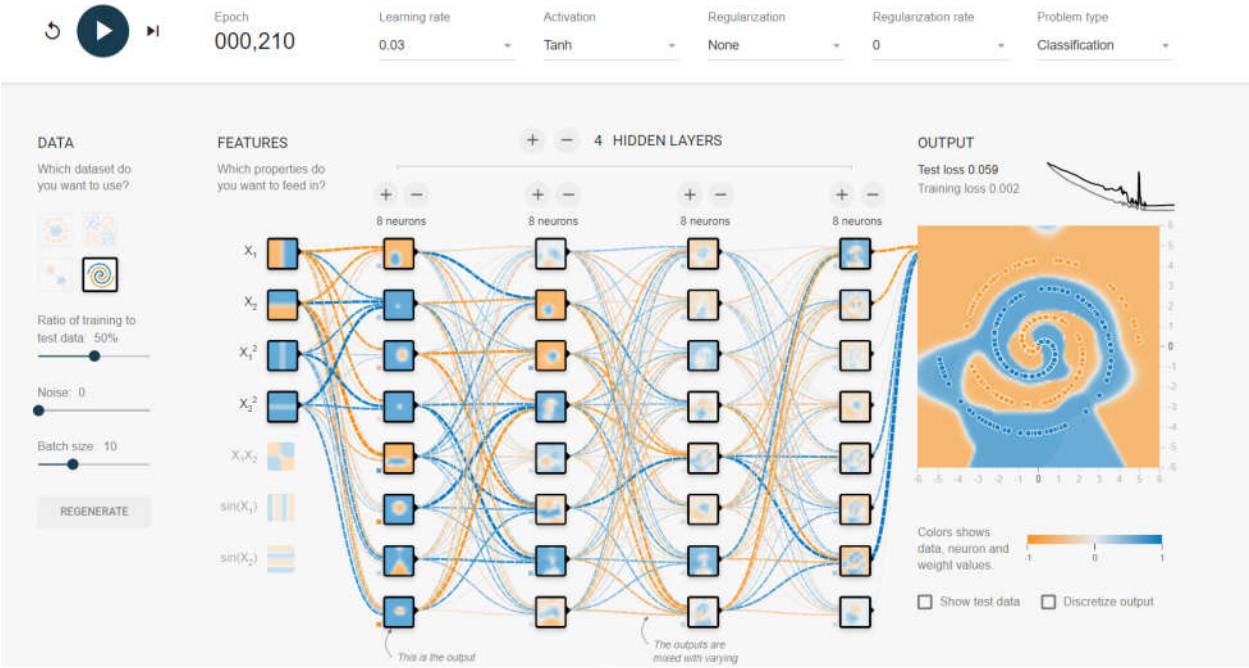
- Train no.1: same as 1f but change the learning rate to 0.1 instead of 0.03 in default. The neural network took less time to finish than part g and it also got stuck on the plateau for a while (around epoch 130th to 145th) - see the below image for details. I also tried learning rate of 0.3 and the neural network could not reach a good solution which means that for this particular set up, the learning rate should not exceed 0.1, not sure if 0.2 works or not but I bet it won't.



- Train no.2: I kept the learning rate as default, 0.03, but this time I added more features and as a result, it took even less time for the neural network to reach a good solution, compared to both 1f and the previous training attempt. Interestingly, there was no fluctuation in the output graph meaning that the network did not stuck in a plateau for this particular training, but when I ran this setup multiple times, the output graph had light fluctuation. I also tried using all features, but it did not seem to boost the speed a lot.



- Train no.3: same setup as the previous attempt but change the activation function to tanh. Apparently, it did not perform as well as the ReLU but it still got a pretty good solution.



- Train no.4: same setup as train no.2 but the learning rate is now 0.1. It was a bit faster to reach a good solution compared to train no.2 but taking a closer look at the output graph, we can see that it fluctuates significantly which gives me a thought that this learning rate is a bit too high for this model, a bit lower, like 0.07 or 0.08 might be better perhaps?

- Train no.5: same as no.4 but reduce the learning rate to 0.01. The model did manage to reach a good solution but as expected, it took way longer. I also tried learning rate of 0.003 and it was too low which caused the fact that the model could not converge to a good solution.