Using the link to the app TensorFlow Playground, classify the spiral datasets and email me a screenshot of your outcomes. Experiment and comment on the role of the optimal number and type of inputs, the size of the train/test split, learning rates and the setting of other hyperparameters with regard to convergence/performance.

Using the default settings as a baseline (inputs: x1 and x2, Hidden Layer 1: 4 neurons, Hidden Layer 2: 2 neurons, learning rate: .03, activation: tanh, regularization: none, 50/50 train/test split) the model performs poorly. The model converged at around 1000 epochs with a training loss of .356 and a test loss of .407. As we can see with the decision boundaries, they are basic and do not pick up the patterns of the data well. The model seems too simple.

A screenshot of a computer

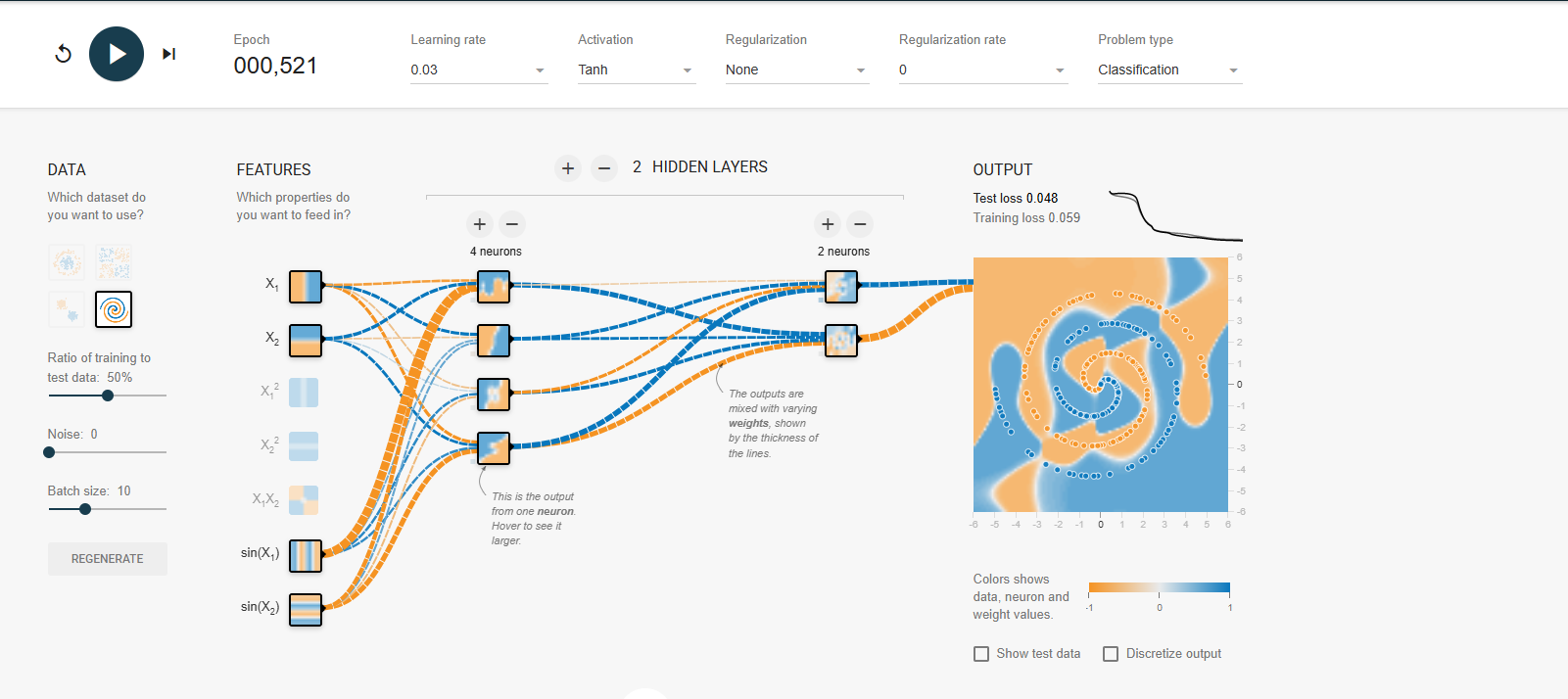
AI-generated content may be incorrect.

The next model I tried, I included all the inputs to see if there was any improvement on the model, and there was! The first observation I had was that the model is converging much faster than with only two inputs (about 300 epochs) and finished with a train loss of .095 and a testing loss of .149. This shows that the inputs can change the model output a lot.

A screenshot of a computer

AI-generated content may be incorrect.

From looking at the image, we can see that x1, x2, sin(x1), sin(x2) carry the thickest weight lines into the hidden layer 1. Next, I will play around with the hidden layers using those 4 inputs, as they seem most relevant for this data distribution. First, I rant the baseline model using those four inputs, and our model drastically improved! We have a training loss of .059 and a testing loss of .048 after about 500 epochs! Looking at the output image, we can see this model is performing okay, and the decision boundary shapes are becoming closer and closer to the true spiral shape.



This shows just how important picking the correct inputs is. More is not always better, as we saw, the model with all 7 inputs performs badly. This can introduce too much noise and cause overfitting, resulting in poor testing performance.

The first thing to try with the hidden layers is to add more neurons. I increased the neuron count in the first hidden layer from 4 to 6 and improved our results even further. In 500 epochs, we achieved a training loss of .011 and a testing loss of .021. These are really good results for the distribution of data we have (a decent rule of thumb is a loss under .1 is a good enough model to deploy in most circumstances). From the picture we can see the extra neurons caused the decision boundaries to be even more accurate (but also still shows many similarities to our previous model).

A screenshot of a computer

AI-generated content may be incorrect.

To illustrate the importance of picking the correct inputs again, I ran the model with the original 2 inputs, but with 6 neurons in the hidden layer and achieved a training loss of .327 and testing loss of .447. A big decline in performance.

Adding an additional hidden layer with 4 neurons and we achieve almost a perfect classification model. After 500 epochs the model achieves a training loss of .003 and a testing loss of .004! It will be hard to improve the model much more than this. The decision boundaries on the output are not perfect (as in they are not a perfect spiral), but do a good job of classifying both sets of our data.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Adding an additional hidden layer with 3 neurons did not help the model much. After 500 epochs the train loss is .017 and the testing loss was .007.

Decreasing the learning rate to .01 resulted in similar results, with the only major difference in being the number of epochs it takes to converge. Increasing it anywhere above .03 (.1 or above) causes chaos in the model. It converges too fast with a very inconsistent loss metric, being around .102 for training and .251 for testing. It is not recommended to change the learning rate of .03 for this data.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Another important aspect of NNs is their activation function. All the above models were ran with tanh activation. Another popular activation is the ReLU. Running our best model with ReLU we get a training loss of .049 and a testing loss of .146, which is not as good as our tanh models, but still acceptable.

A screenshot of a computer

AI-generated content may be incorrect.

The default on tensorflow playground is 50% train test split. Traditional ML recommends 20% testing data to 80% training data. Running a few trails, the model converges faster than the others (in about 300 epochs) and achieves a training loss of .004 and a testing loss of .004. The decision boundary is loosely spiral.

A screenshot of a computer

AI-generated content may be incorrect.

Flipping it to 20% train and 80% test, the model takes much longer to converge and has poor performance with a training loss of .001 but a testing loss of .235. This seems to be overfitting the training data as there is not as much of it.

A screenshot of a computer

AI-generated content may be incorrect.

Lastly, changing the batch size does not have a huge impact on this model. However, decreasing it below 5 causes a large dip in performance. Increasing the batch size leads to similar results, with a slightly longer convergence.

All the parameters in a NN are important, it is hard to single out one being much more important than the others. Having the correct data inputs for the model is paramount to reduce noise and irrelevant data in our model. We saw this with step 1, when choosing the correct inputs drastically improved out model and we saw that more does not mean better. The hidden layers are important, and something that will need to be played around with for all models. 1 hidden layer usually is not going to be enough for complex data. The neurons should also decrease in number for each additional hidden layer. It is important to reiterate again that more does not always mean better. When we added a 4th hidden layer, the model performed poorly and overfit some of our data. In more complex models, using random neuron dropout during training can reduce overfitting. The activation function will differ based on the problem at hand. Generally speaking, having more training data than testing data is standard practice in data science. A 80% training and 20% testing split is what is generally recommended and this split performed much better than the 50/50 split in the default settings. Batch size can be a problem if it is too small (converges too fast and overfits the data) and large batch sizes can cause long convergence times and not enough weight adjusting. Every problem with NNs will be different and the more models you create the better you will get at identifying which parameters need to be adjusted in different settings.