PRML LAB 9

Shalin Jain (B21CS070)

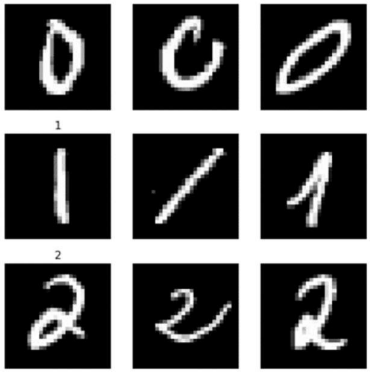
Problem 1

Part A

In this part we downloaded the dataset, found the mean and the standard deviation of the respective train test sets and then applied the given augmentations in question in the transformed and then again downloaded the dataset.

Part B

In this part we plotted some of the images of the handwritten digits from the dataset and also made the dataset loader for all training, validation as well the testing dataset



Part C

In this part we created a perceptron using the pyTroch module having three layers, the input layer, the hidden layer and the output layer. The number of nodes in input layer is 784, in hidden layers is 256 and in the output layer is 10.

The number of trainable parameters obtained is 269,322.

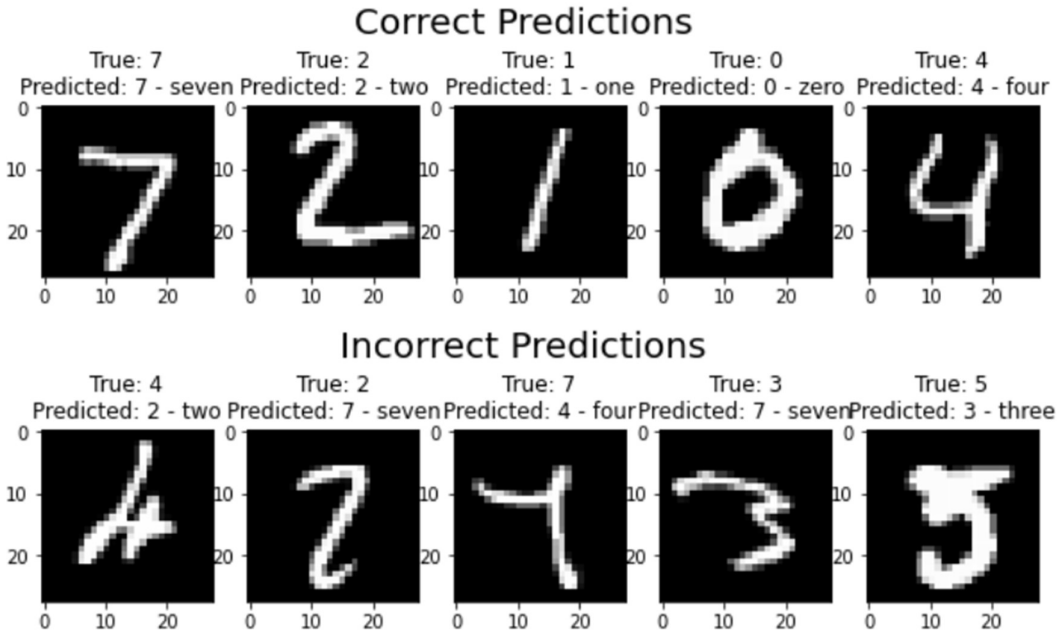
Part D

In this part we train the perceptron for the 5 epochs calculated the training loss, training accuracy as well the validation loss and the validation accuracy for all the epochs.

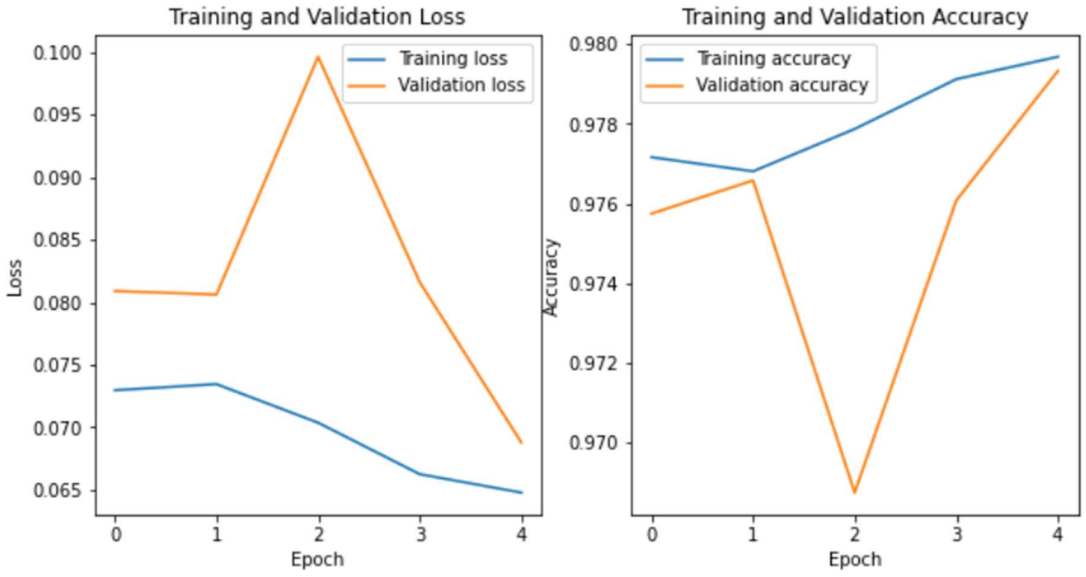
```
Epoch 1/5, Train Loss: 0.0730, Train Acc: 0.9772, Val Loss: 0.0809, Val Acc: 0.9758
Epoch 2/5, Train Loss: 0.0734, Train Acc: 0.9768, Val Loss: 0.0806, Val Acc: 0.9766
Epoch 3/5, Train Loss: 0.0704, Train Acc: 0.9779, Val Loss: 0.0996, Val Acc: 0.9688
Epoch 4/5, Train Loss: 0.0662, Train Acc: 0.9791, Val Loss: 0.0816, Val Acc: 0.9761
Epoch 5/5, Train Loss: 0.0648, Train Acc: 0.9797, Val Loss: 0.0688, Val Acc: 0.9793
Best Val Acc: 0.9793
```

Part E

In this part we plotted the some of the correct as well as the incorrect predictions as shown below:



After that we calculated loss versus epoch and accuracy versus epoch for both the training and the validation set as shown below



From the graph we can infer that loss decrease with increase in number of iteration i.e. epochs which corresponds to increase in accuracy on both training and validation set which shows the correctness of the model.

Problem 2

Part 1

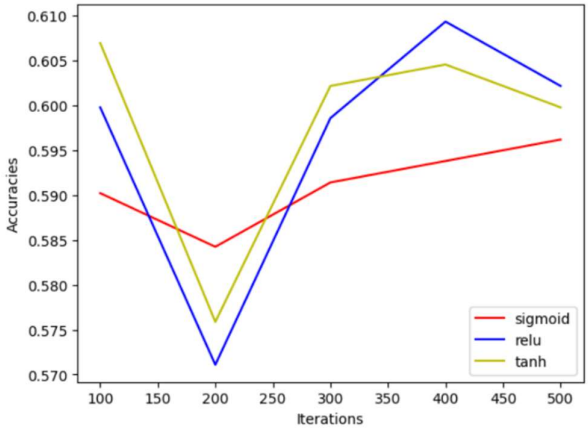
In this part we loaded the abalone dataset and pre-processed it. Since we can see that some of the classes are such that they have only one sample in dataset. This shows that we cannot use basic train test split method for splitting the dataset. Hence, we used a special technique known as stratified split for splitting. In this method we forcibly pushed the low number class samples into both training and testing sets. In this part we have also reduced the number of classes from 29 to 4.

Part 2

In this part we made a multiclass perceptron with 2 hidden layers, different weight initialization options as well as different activation function options. Since it was multiclass problem, we used SoftMax function as an activation function for the final layer. This implementation includes forward propagation of input i.e. first we find the output on the input layer used it to calculate the output on second and so on. We used backpropagation of error i.e. we calculated the error on the last layer, then before the last layer and so on. We the made stochastic gradient descent function i.e. the fit function that divides the input dataset into batches, the batch size is taken as input by the user, and perform forward as well as backward propagation alternatively till the epoch limit is reached. It also consist of a predict function which predicts the class of the input datapoints. It also contains function to load as well as save weights.

Part 3

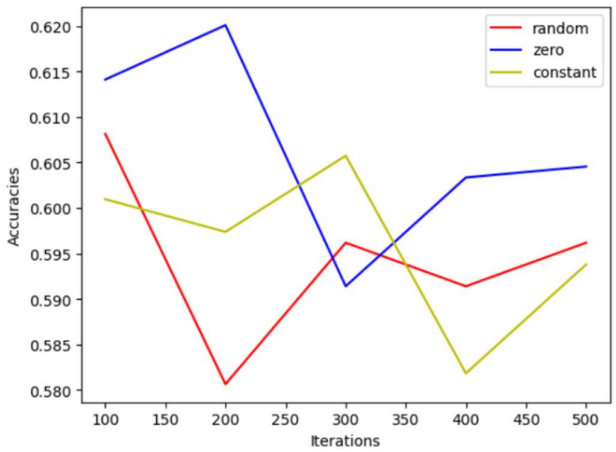
In this part we vary the activation function and saw the variation in accuracy as shown below: -



We can see that the accuracy increases slightly by with iteration but on changing the activation function the accuracy increases by almost 1 to 5%. This is because some activation functions are good for binary classification while some are good for multiclass classification. We can see that sigmoid function is more stable.

Part 4

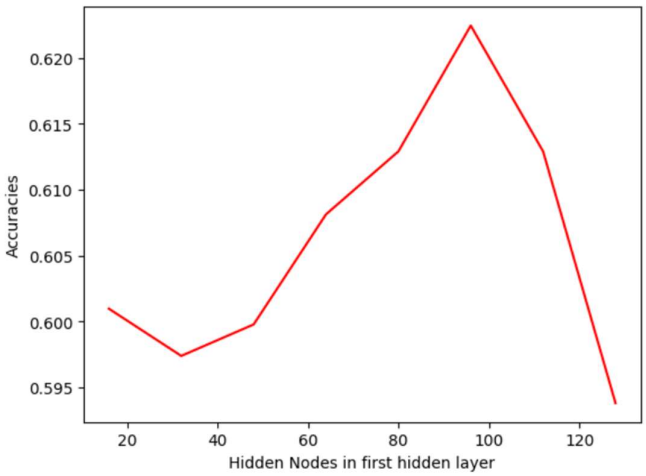
In this part we varied the weight type and observe the accuracy as shown below: -



We can see that the variation in accuracy on varying the weight initialization type. We can see that we get better accuracies if the weights initialization is zero.

Part 5

In this part we vary the number of nodes in the first as well as the second layer and obtained the following graph



We can see that on increasing the number of nodes to a very high number increases the complexity of the model which lead to increment in the accuracy.

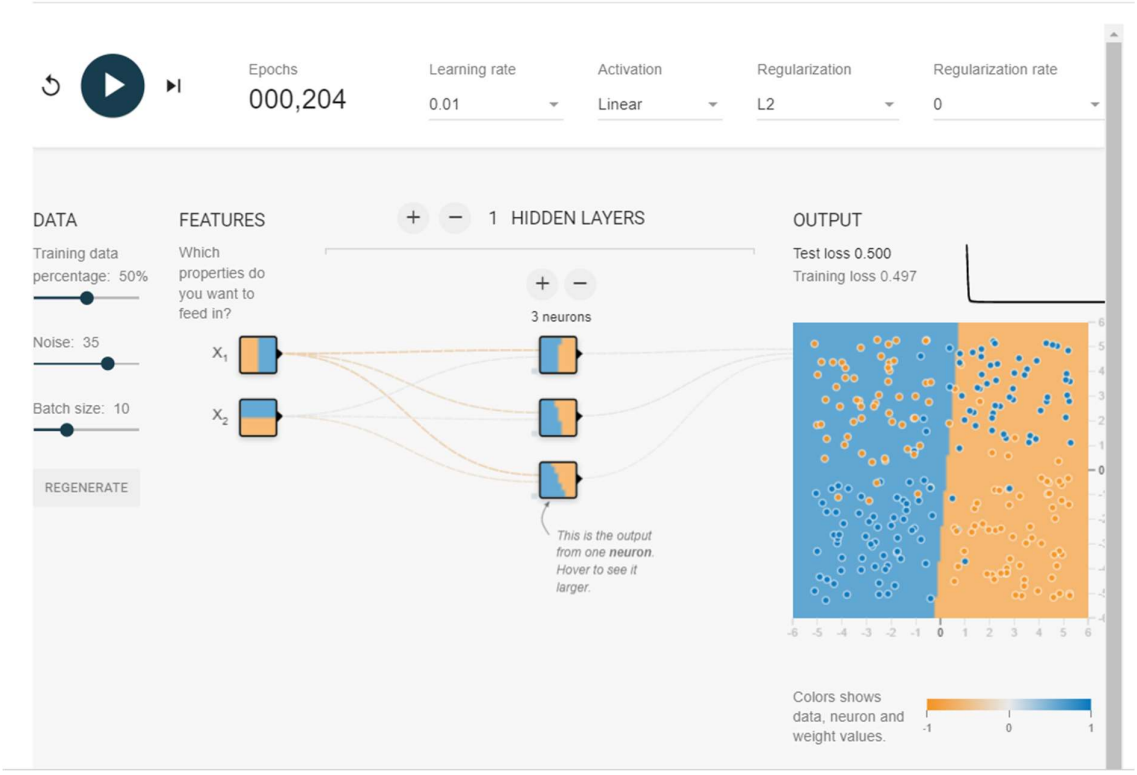
The loaded weights can be seen in Colab notebook itself.

Problem 3

Playground

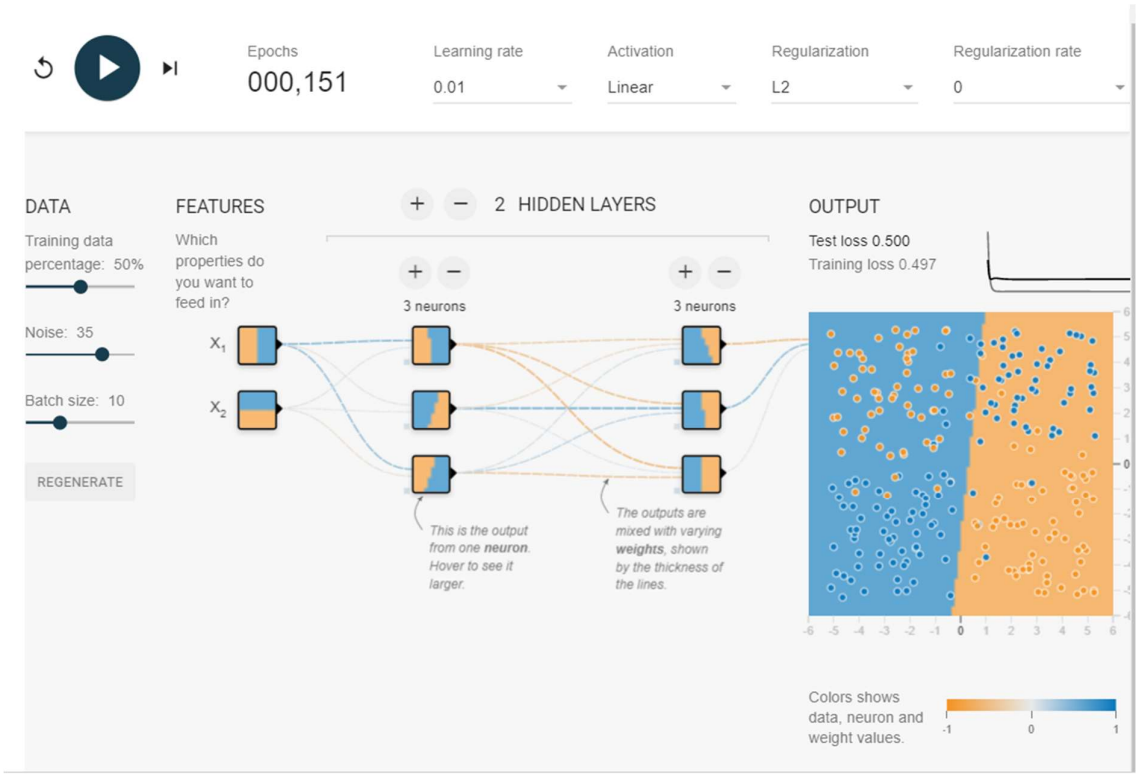
A) Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model? For example, try the following architecture:

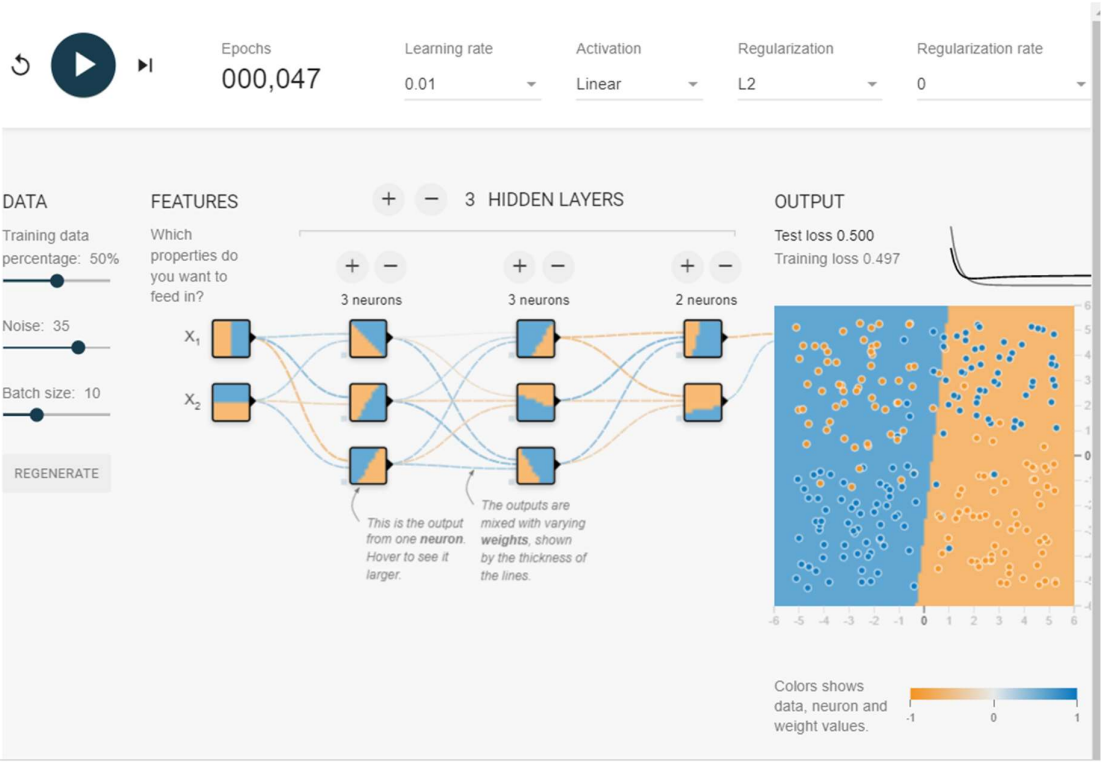
- First hidden layer with 3 neurons.
- Second hidden layer with 3 neurons.
- Third hidden layer with 2 neurons.



As we increase the number of neurons and the number of layers the model get converges in lesser number of epochs but it takes a little more time than the lesser number of the neurons. This is also depicted by the screenshots attached below.

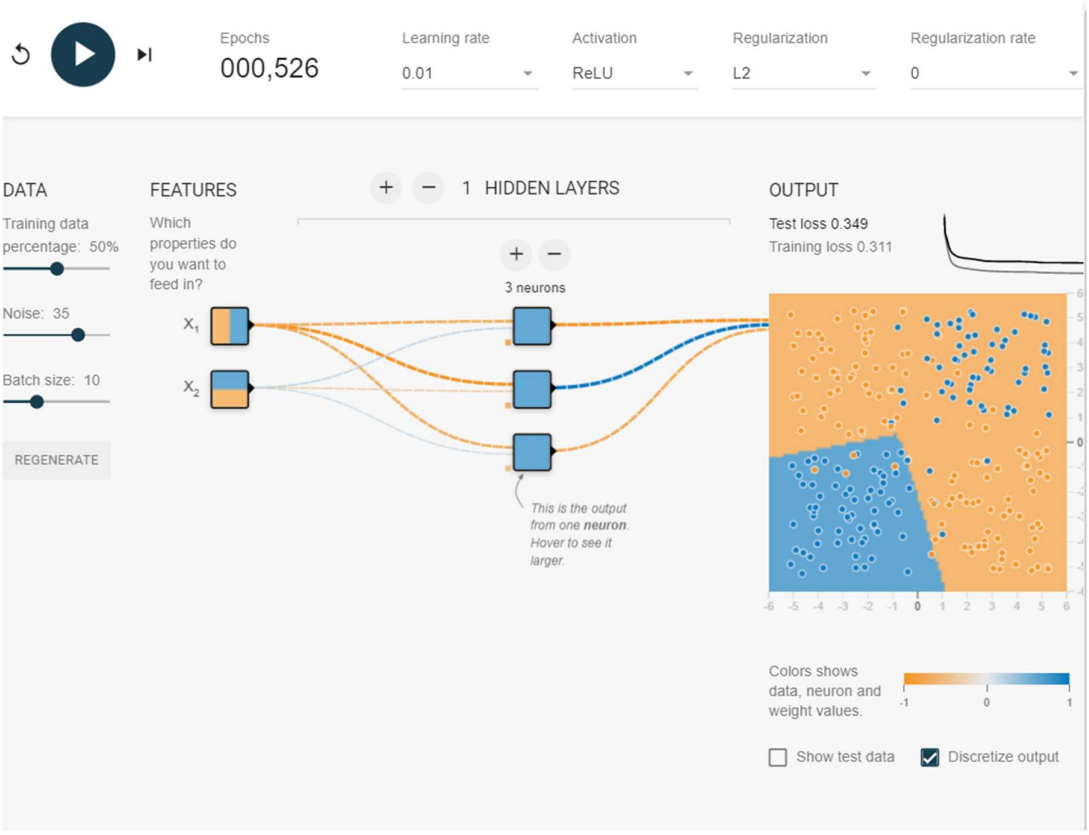
(Answers appear just below the exercise.)

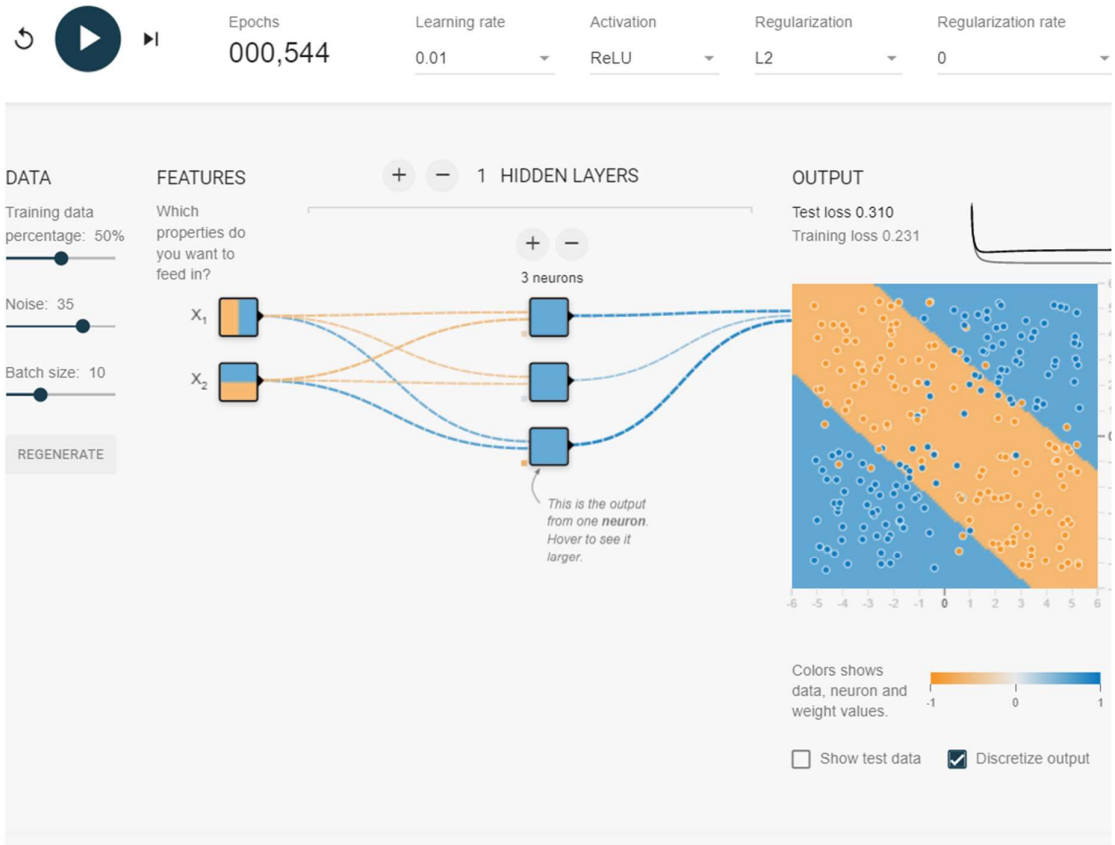


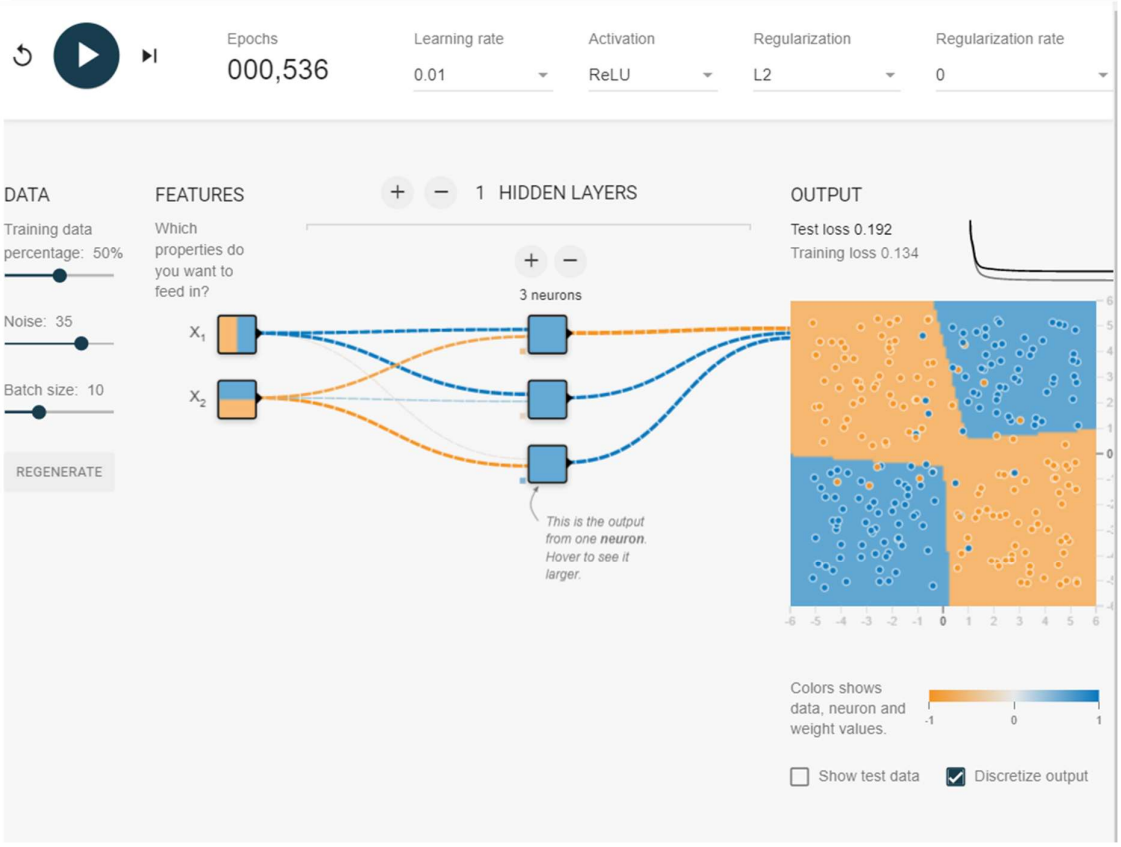


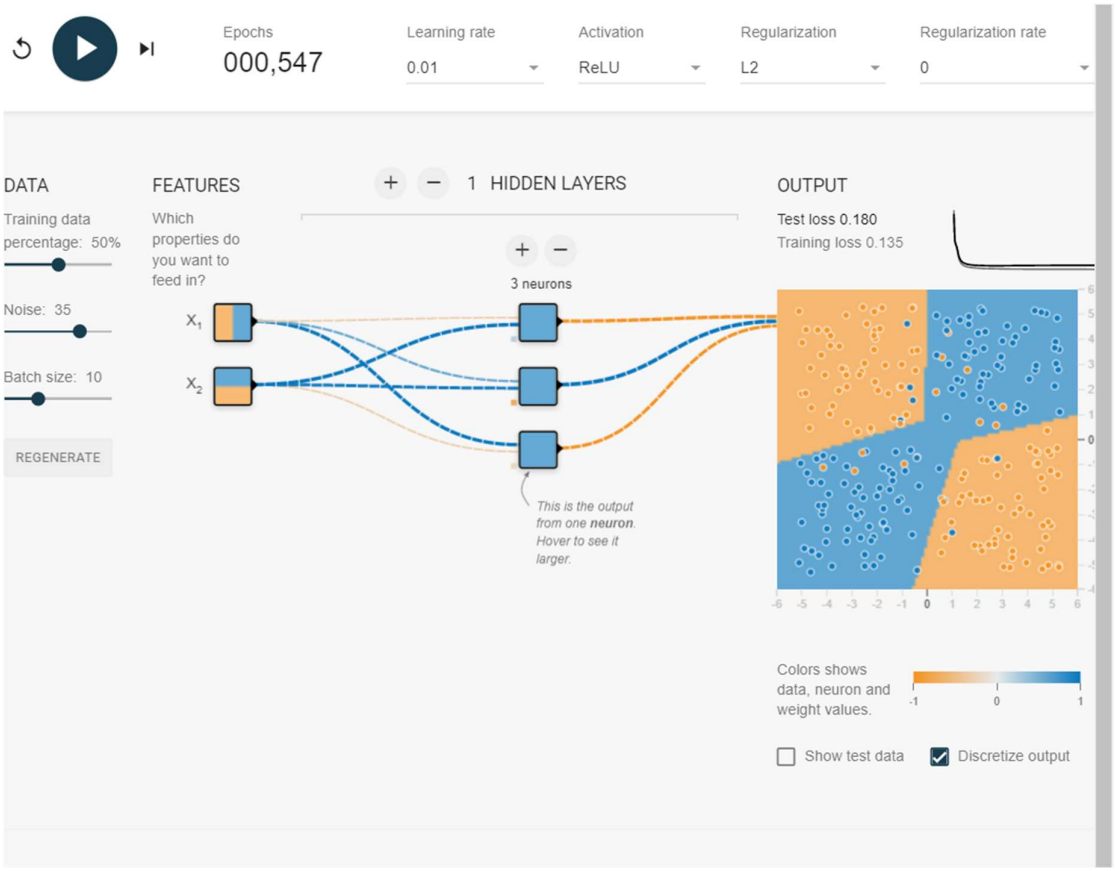
B) Run the model as given four or five times. Before each trial, hit the Reset the network button to get a new random initialization. (The Reset the network button is the circular reset arrow just to the left of the Play button.) Let each trial run for at least 500 steps to ensure convergence. What shape does each model output converge to? What does this say about the role of initialization in non-convex optimization? Try making the model slightly more complex by adding a layer and a couple of extra nodes. Repeat the trials from Task 1. Does this add any additional stability to the results?

In this part we can see that every boundary is different from the other one and on adding one more layer to the network the accuracy might not increase to a large extent.

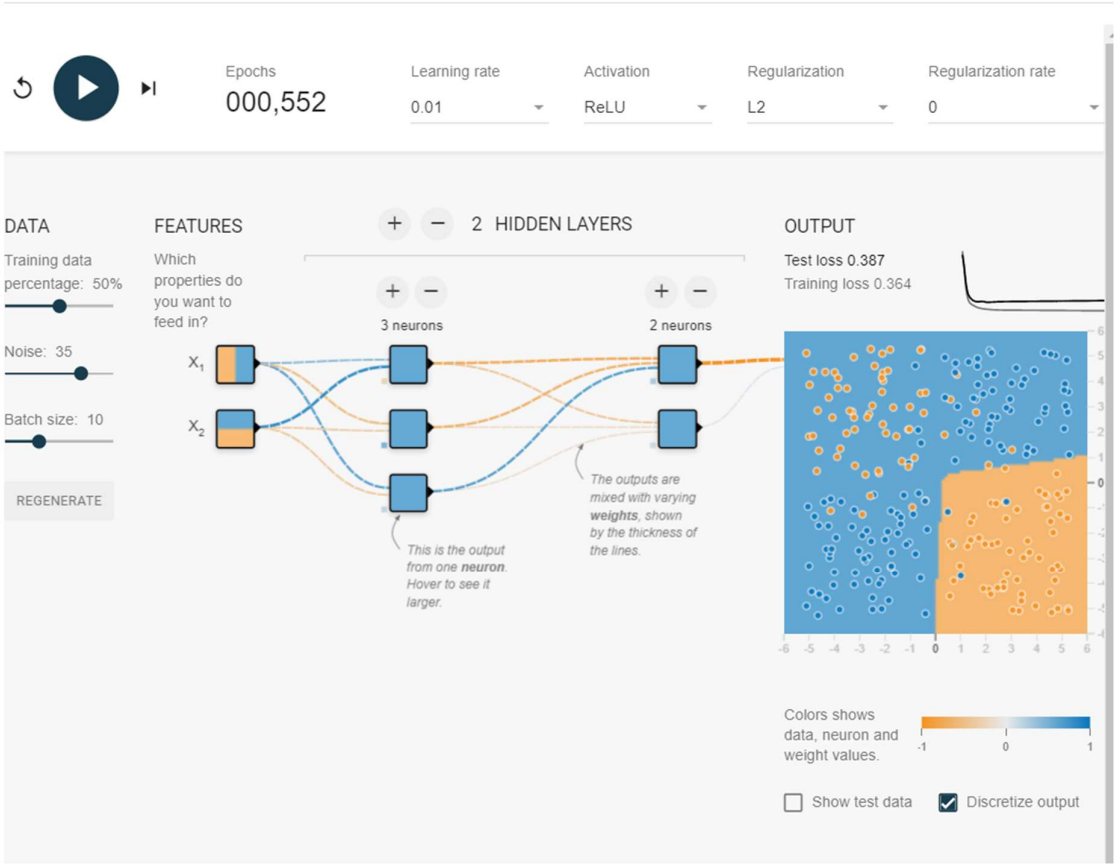


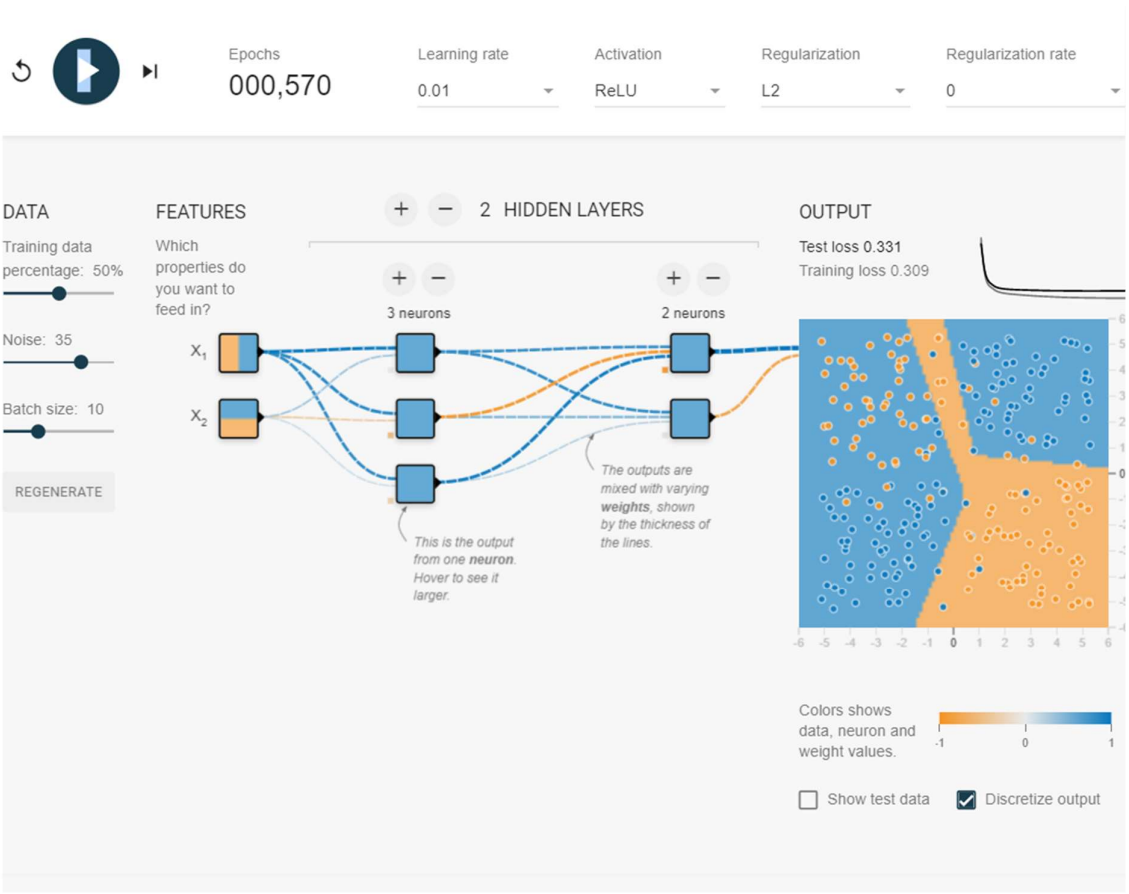

Epochs
000,544
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0
DATA
Training data percentage: 50%
Noise: 35
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYERS
3 neurons
X1
X2
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.310
Training loss 0.231
Colors shows data, neuron and weight values.
Show test data
Discretize output

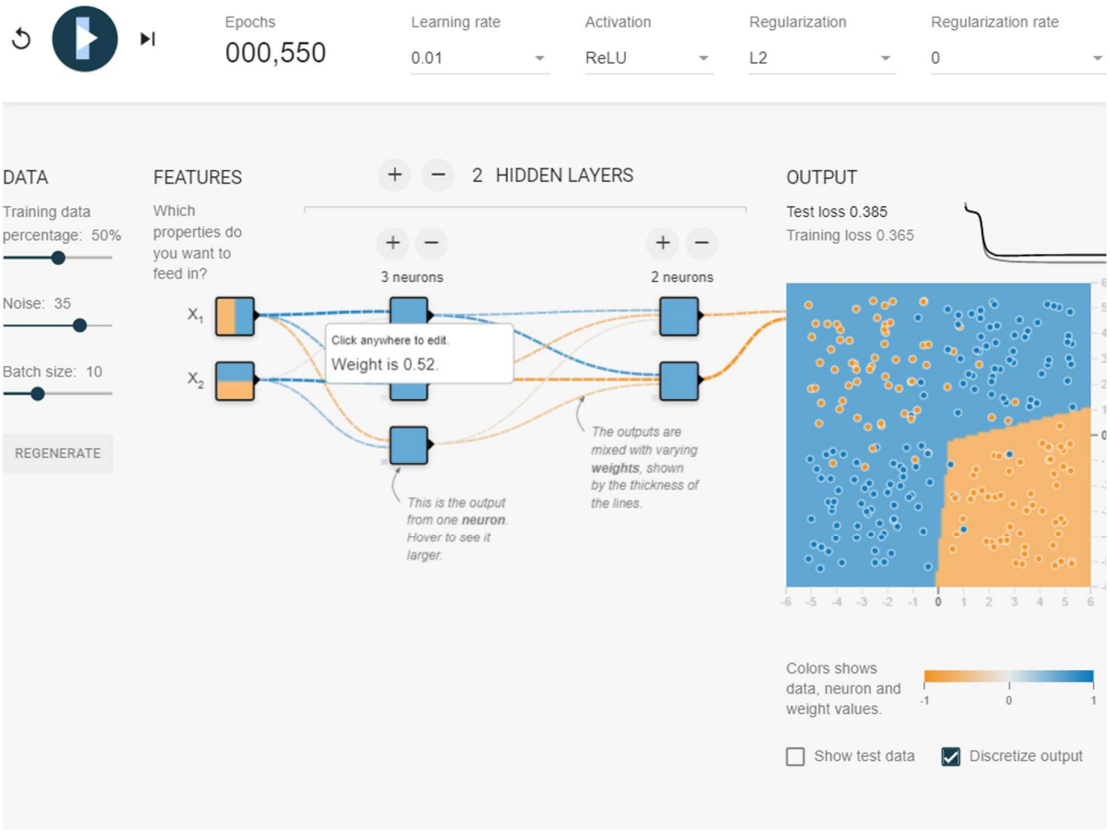

Epochs
000,536
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0
DATA
Training data percentage: 50%
Noise: 35
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYERS
3 neurons
X1
X2
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.192
Training loss 0.134
Colors shows data, neuron and weight values.
Show test data
Discretize output

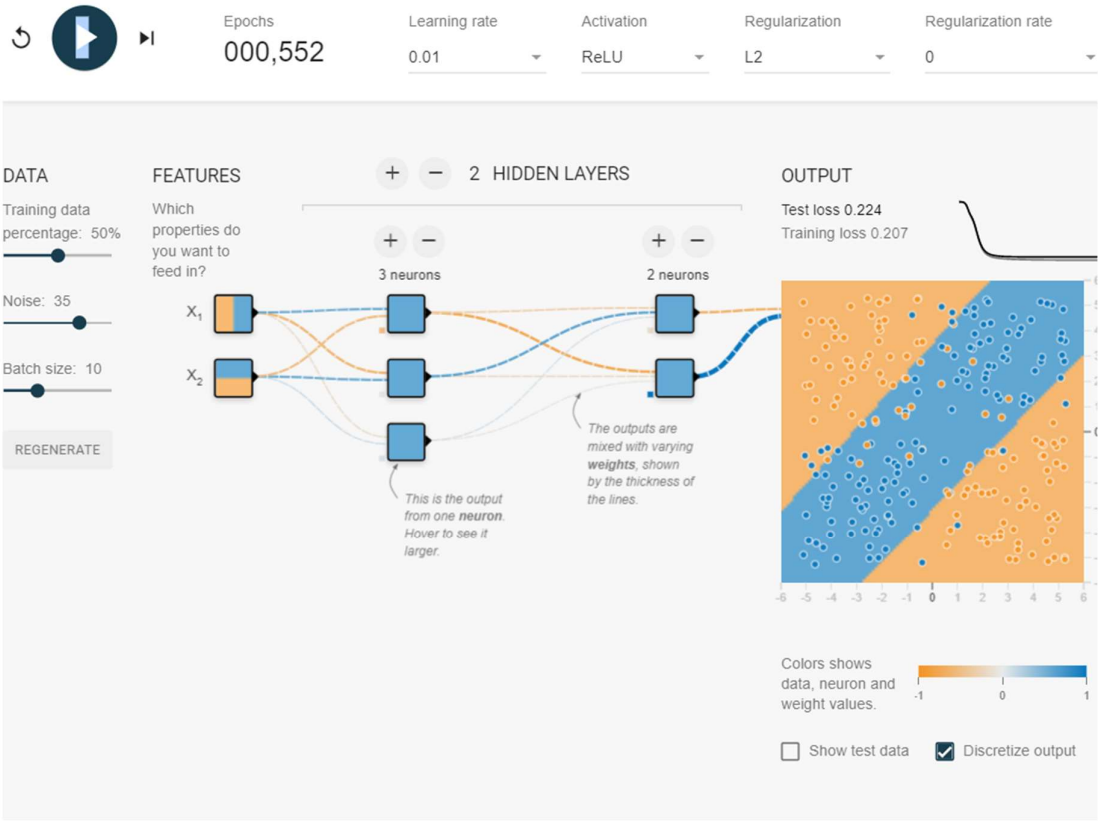


More layers and nodes

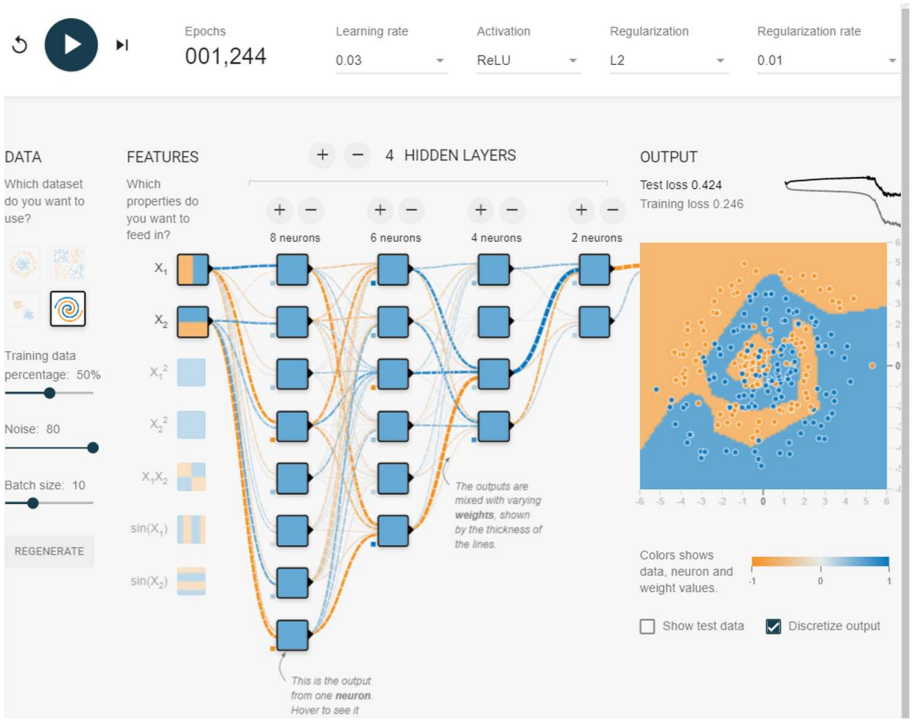


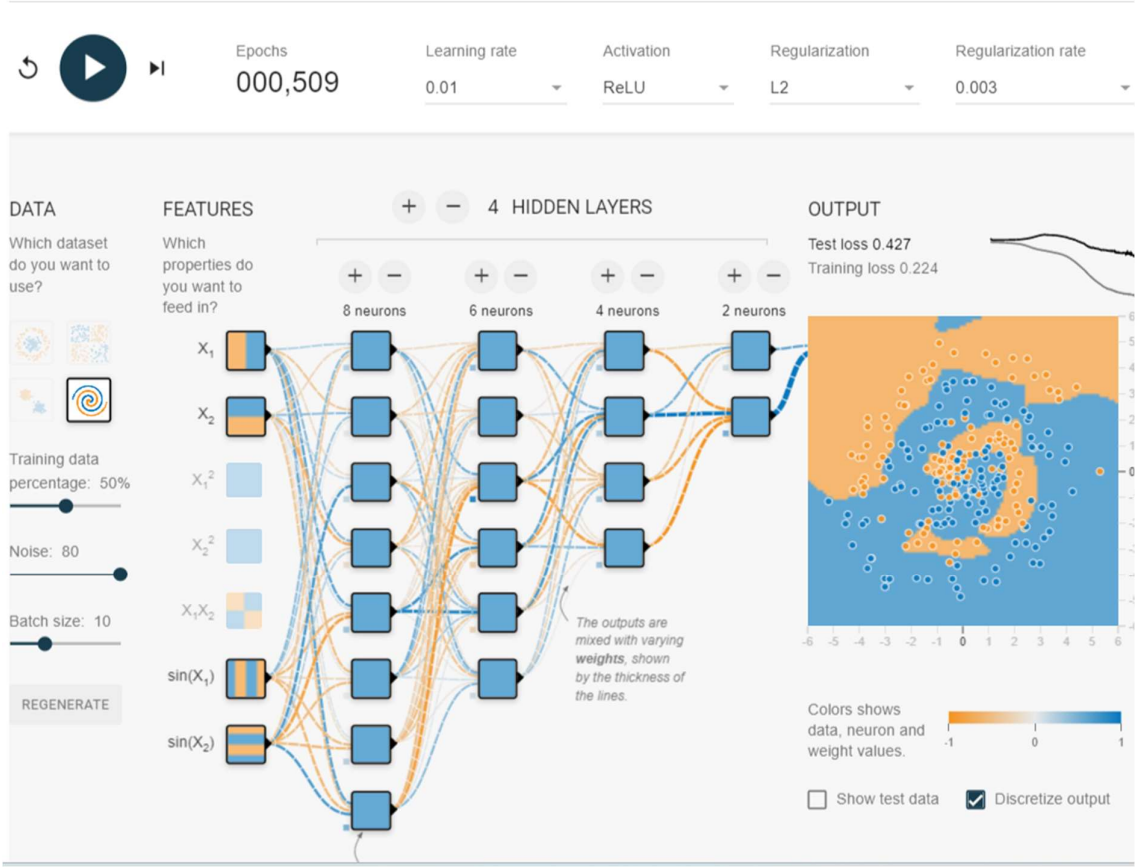

Epochs
000,570
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0
DATA
Training data percentage: 50%
Noise: 35
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
3 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.331
Training loss 0.309
Colors shows data, neuron and weight values.
Show test data
Discretize output


Epochs
000,550
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0
DATA
Training data percentage: 50%
Noise: 35
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
3 neurons
2 neurons
Click anywhere to edit.
Weight is 0.52.
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.385
Training loss 0.365
Colors shows data, neuron and weight values.
Show test data
Discretize output



C) Train the best model you can, using just X1 and X2. Feel free to add or remove layers and neurons, change learning settings like learning rate, regularization rate, and batch size. What is the best test loss you can get? How smooth is the model output surface? Even with Neural Nets, some amount of feature engineering is often needed to achieve best performance. Try adding in additional cross product features or other transformations like sin(X1) and sin(X2). Do you get a better model? Is the model output surface any smoother?

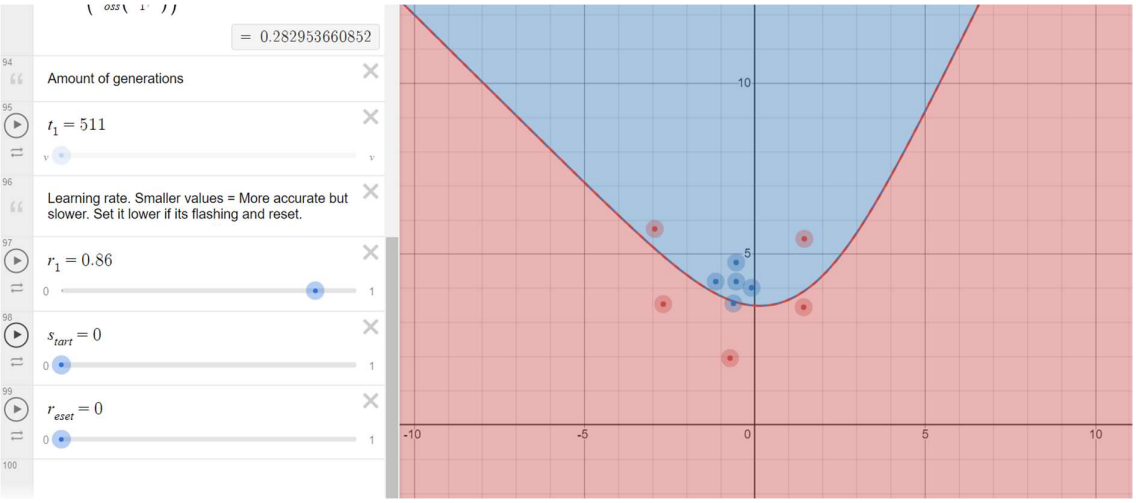




On normally using two features is accurate but take a large amount of time but if we introduce a complex feature function then the time take for getting the boundary is very less.

Neural Nets

Try tweaking the learning rate and report your observations regarding the stability of training by observing the decision boundary.



We see that at learning rate of 0.86 and epochs > 500 the almost perfectly shows the boundaries.