CS7015
Deep Learning
Assignment 2B

Divya K Raman
EE15B085

7th October 2018

1 Introduction

In this assignment, we implement the backpropagation algorithm, stochastic gradient descent, activation functions and their gradients in python using numpy. We are given two datasets: the first is a spiral with 2 classes and the second is a spiral with 3 classes. We construct neural networks with 2 and 3 fully connected layers, experiment on them with the activation functions: linear, ReLU, tanh, sigmoid. We experiment with two loss functions: Softmax (cross entropy) loss and SVM (max margin) loss. We further experiment with different learning rates and different random initializations. We present a study of all this below.

2 Details

We use stochastic gradient descent for the optimisation. The batch size is set to be equal to the training dataset size. For each case, the model architecture, experimental results, train and test error, train error plots, decision boundary plots are given. Parameter tuning is also done on the learning rate, random initialisation of weights and number of epochs to get better results.

Gradients for backpropagation are computed as derived in assignment 2A.

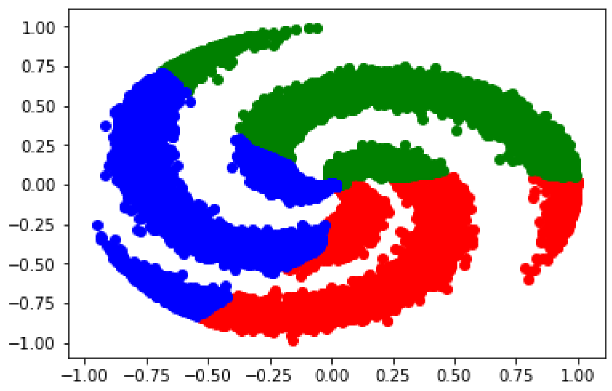
3 Softmax (Cross entropy) loss

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

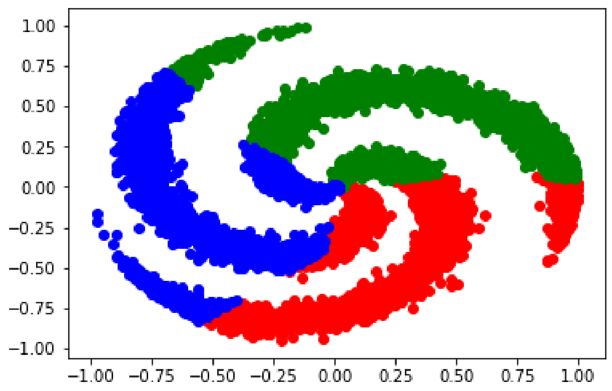
3.1 Linear followed by ReLU

ipython notebook: Softmax.ipynb
Linear followed by ReLU at each layer
Softmax followed by cross entropy loss in the output layer

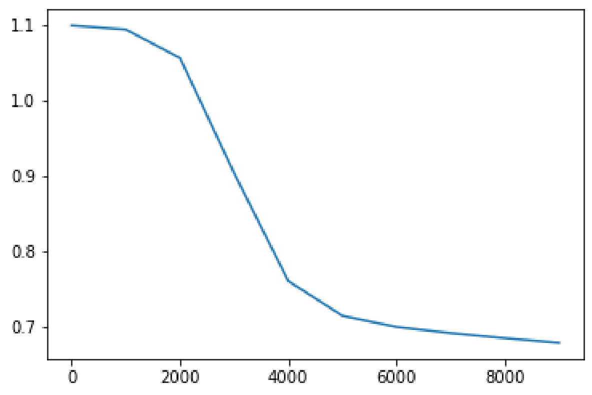
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.01 epochs = 10000
Train accuracy: 0.59



Test accuracy: 0.58



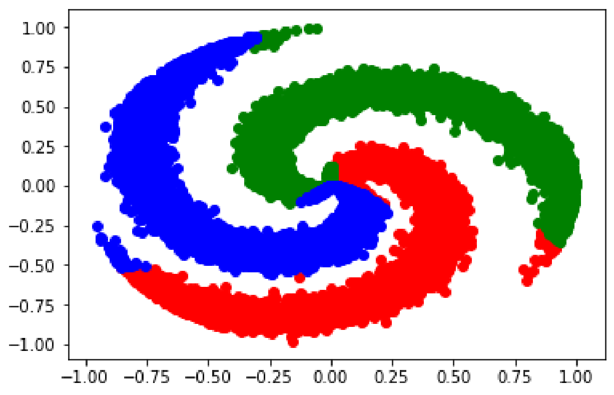
Train error plot:



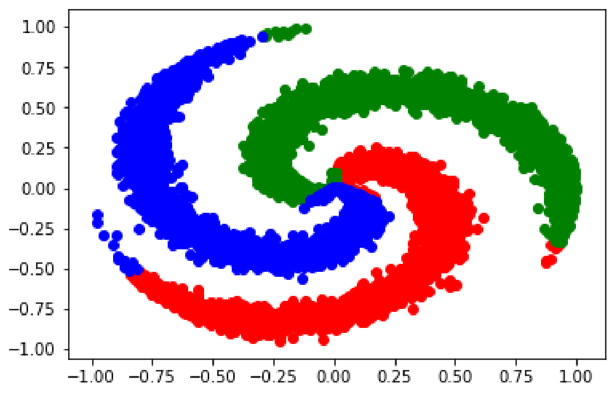
2. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

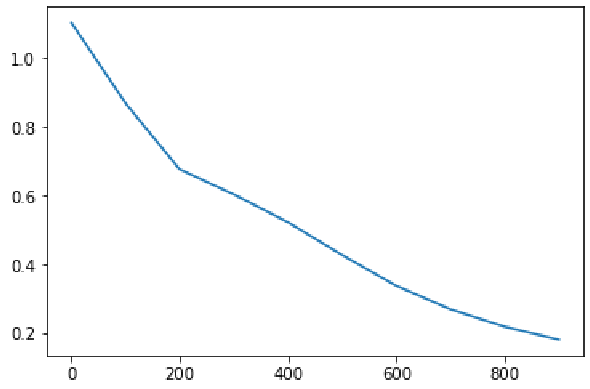
Train accuracy: 0.96



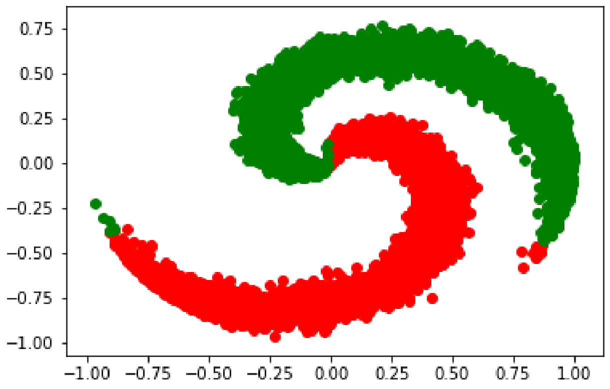
Test accuracy: 0.96



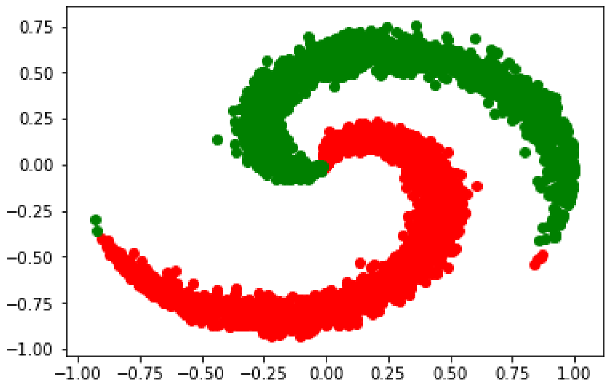
Train error plot:



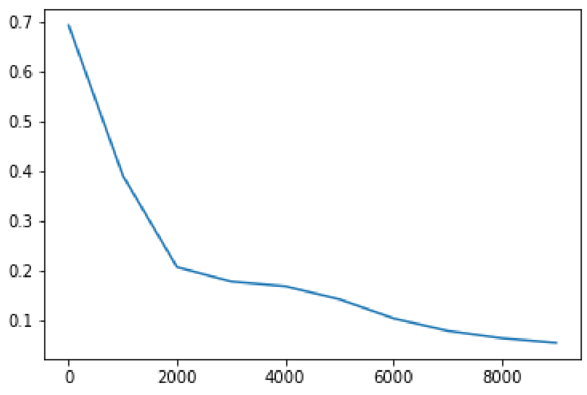
3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.1 epochs = 10000
Train accuracy: 0.99



Test accuracy: 0.99



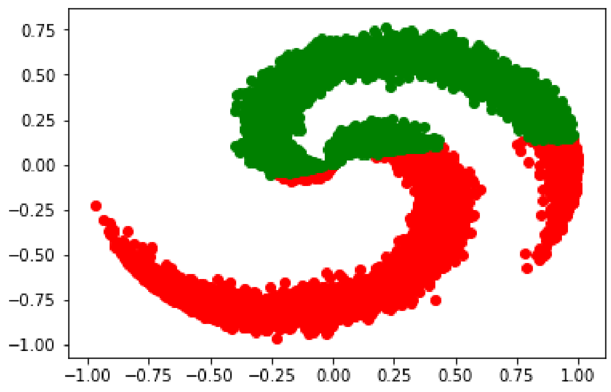
Train error plot:



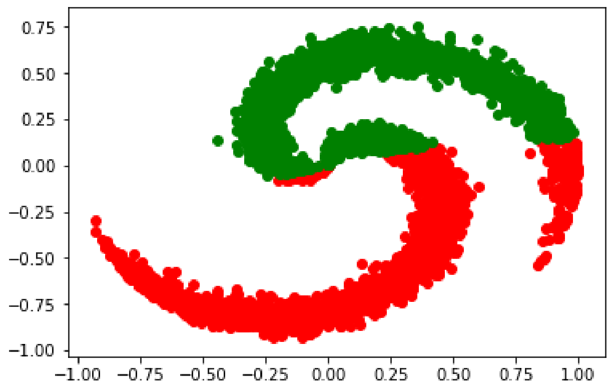
4. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.01 epochs = 1000

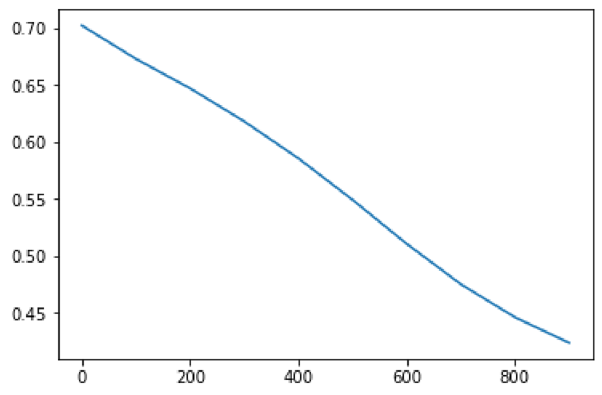
Train accuracy: 0.74



Test accuracy: 0.75



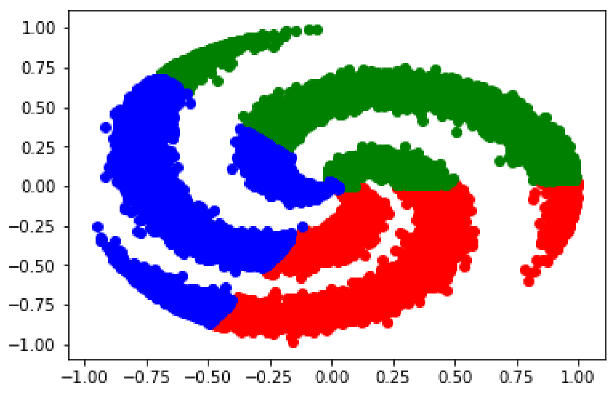
Train error plot:



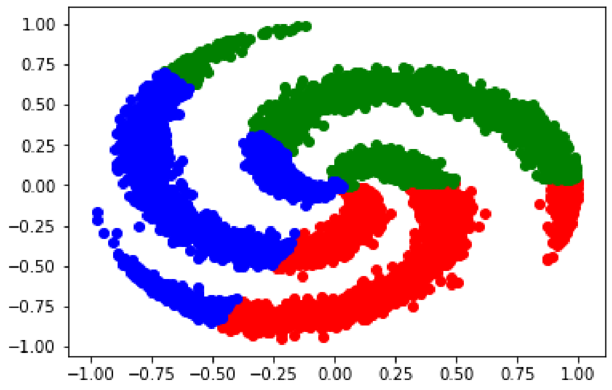
3.2 Sigmoid

ipython notebook: Softmax$_{S}igmoid.ipynb$
$Sigmoidateachlayer$
$Softmaxfollowedbycrossentropylossintheoutputlayer$

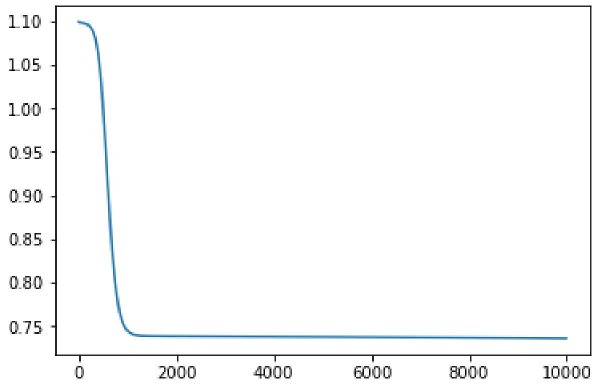
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.2 epochs = 10000
Train accuracy: 0.53



Test accuracy: 0.53

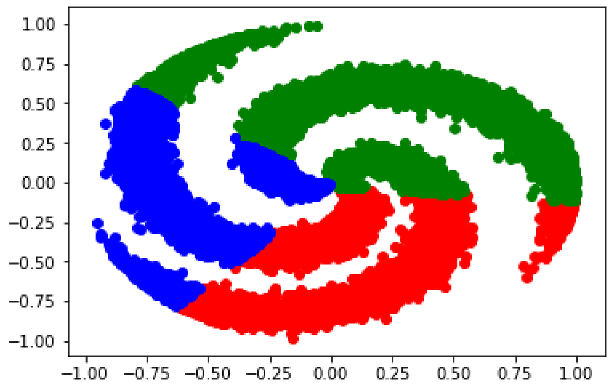


Train error plot:

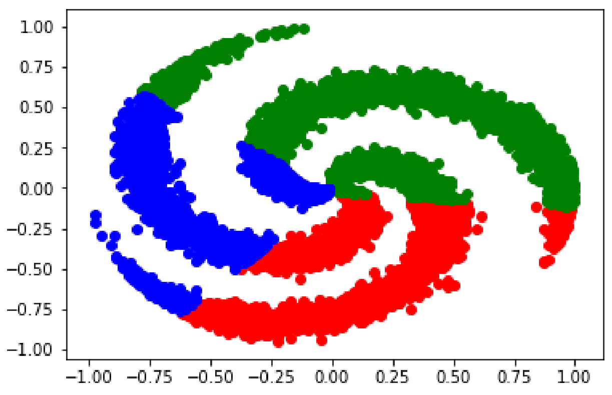


2. 3 fully connected layers, 3 classes
hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

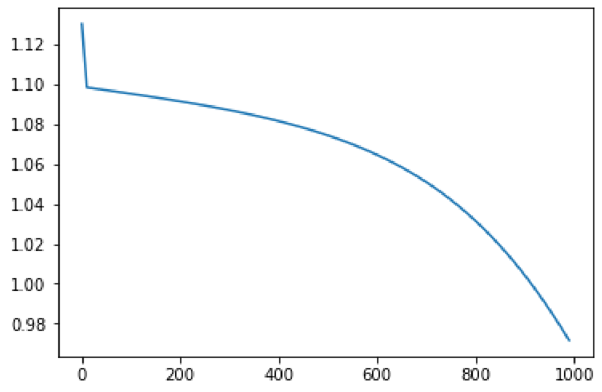
Train accuracy: 0.52



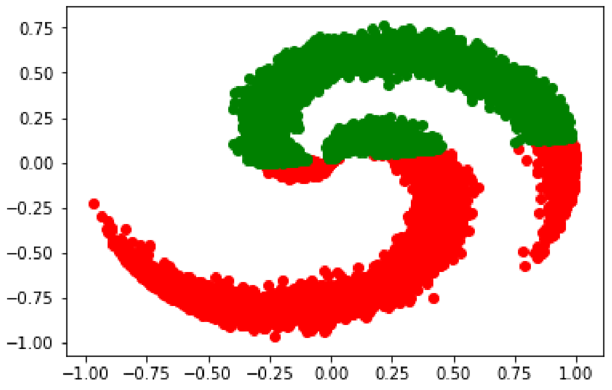
Test accuracy: 0.52



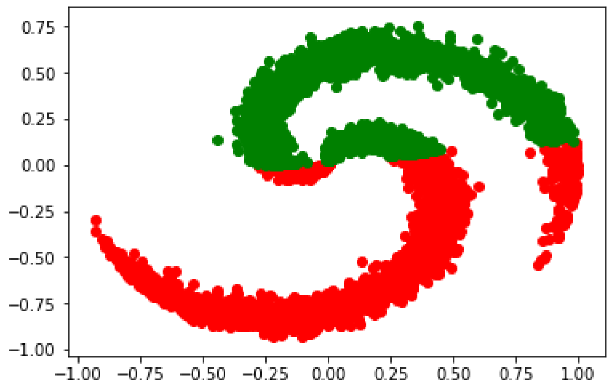
Train error plot:



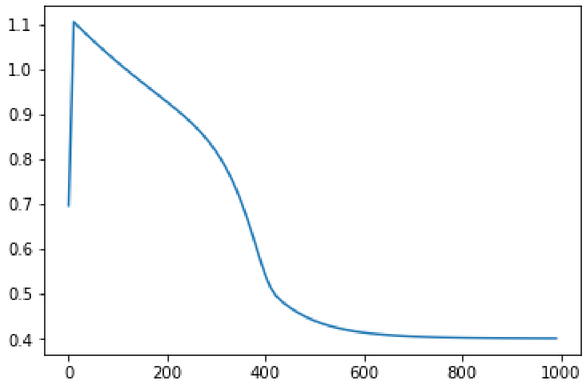
3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.2 epochs = 1000
Train accuracy: 0.71



Test accuracy: 0.71



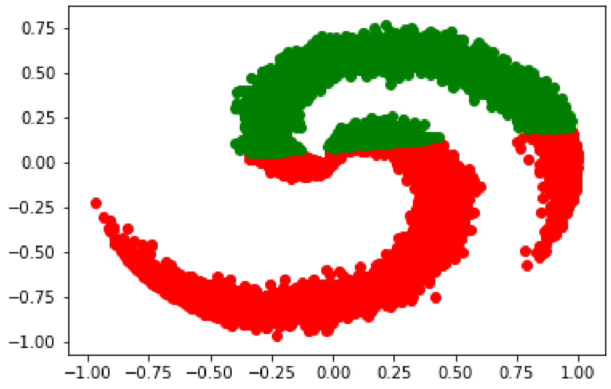
Train error plot:



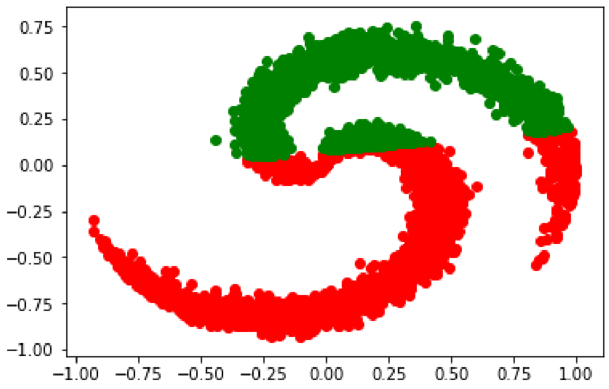
4. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 1 epochs = 1000

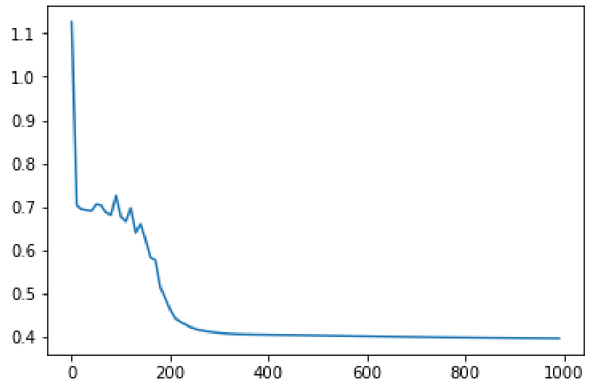
Train accuracy: 0.74



Test accuracy: 0.75



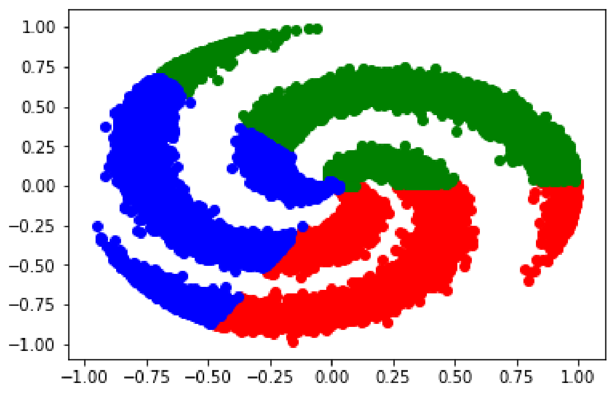
Train error plot:



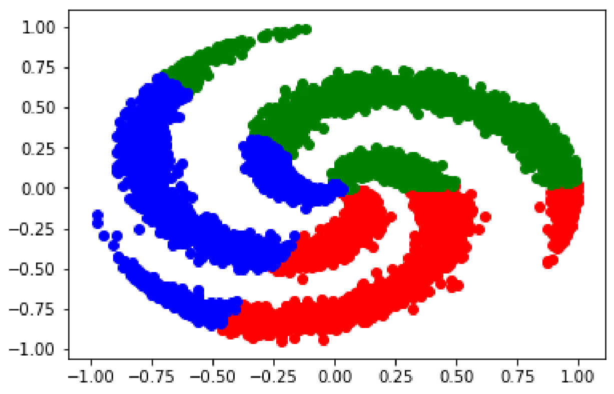
3.3 Tanh

ipython notebook: Softmax$_Tanh.ipynb$
$Tanh at each layer$
$Softmax followed by cross entropy loss in the output layer$

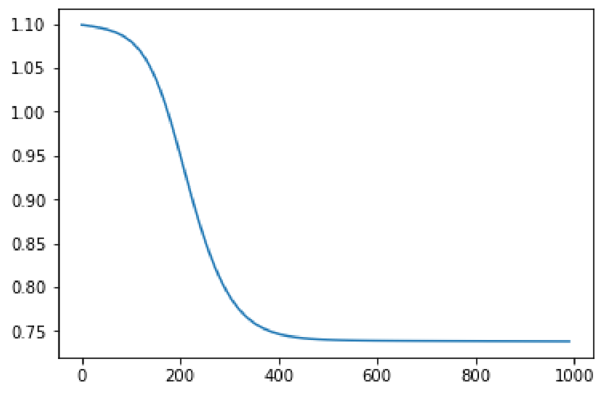
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.1 epochs = 1000
Train accuracy: 0.53



Test accuracy: 0.53



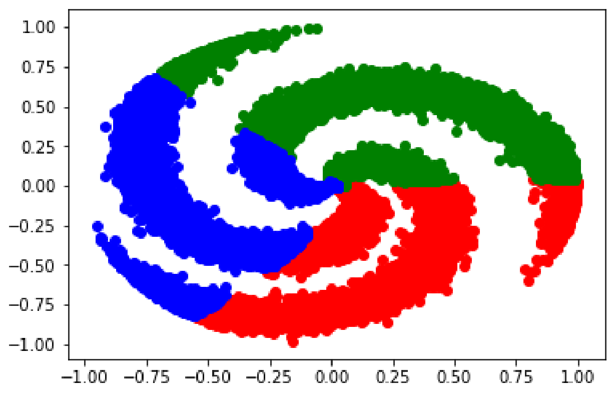
Train error plot:



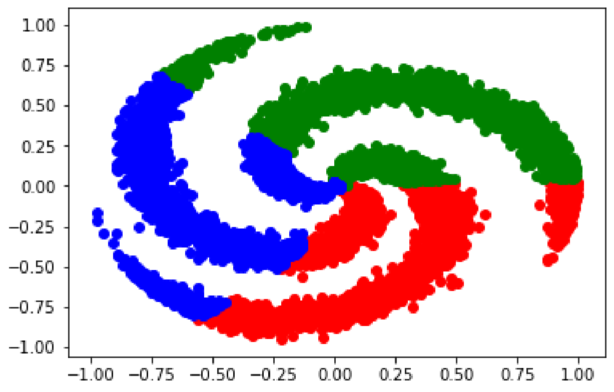
2. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

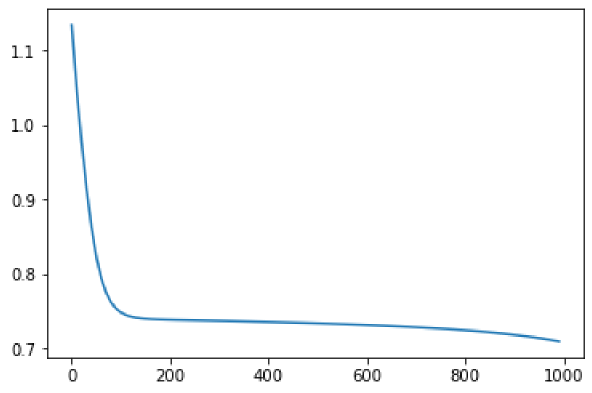
Train accuracy: 0.55



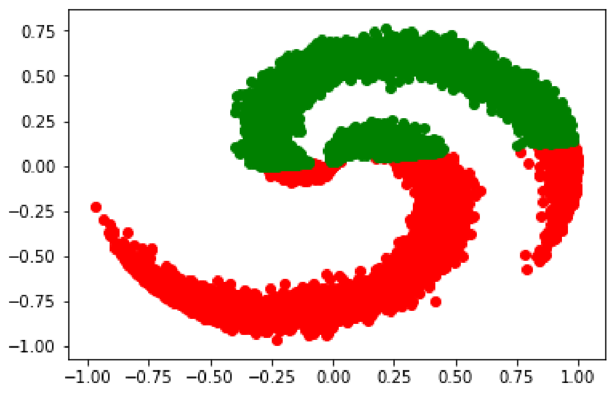
Test accuracy: 0.55

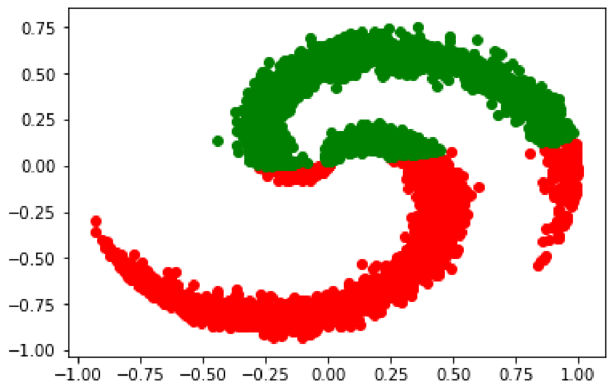


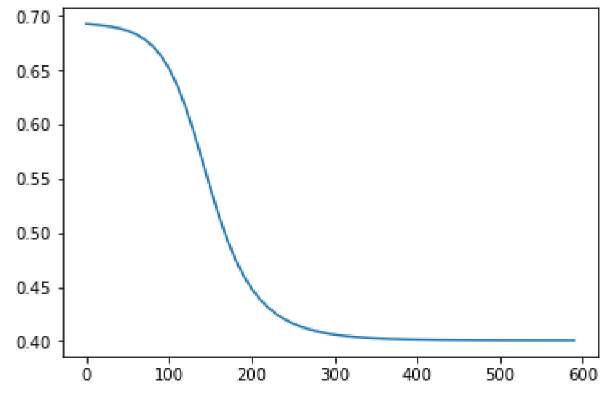
Train error plot:



3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.1 epochs = 600
Train accuracy: 0.71

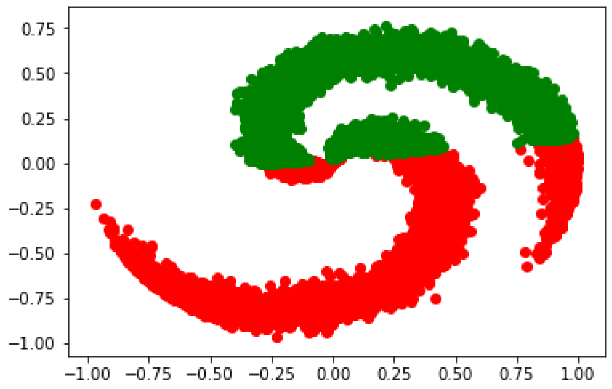

Test accuracy: 0.71


Train error plot:

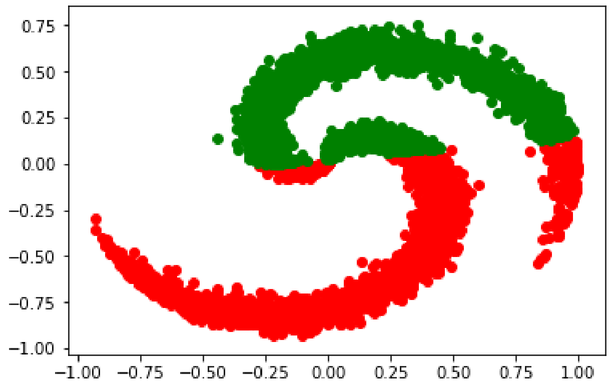

4. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 600

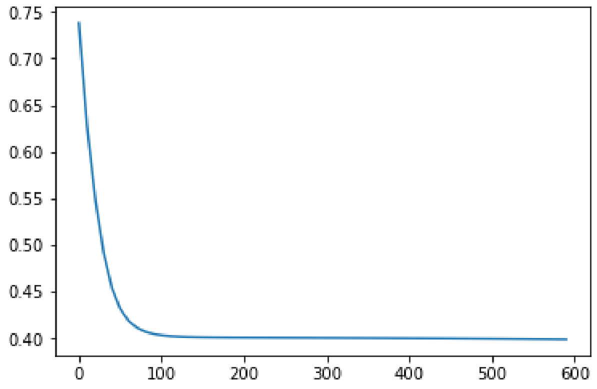
Train accuracy: 0.71



Test accuracy: 0.72



Train error plot:



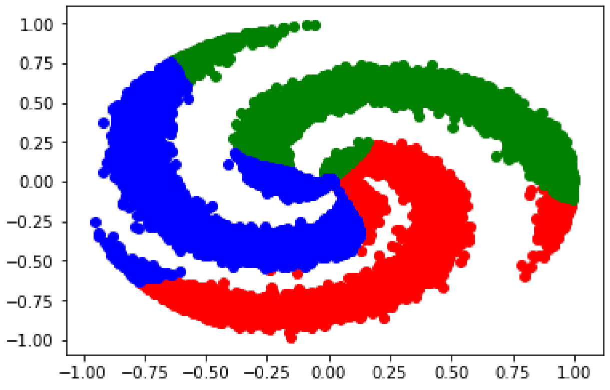
4 SVM (Max Margin) loss

Hinge loss is a loss function used for maximum margin classification

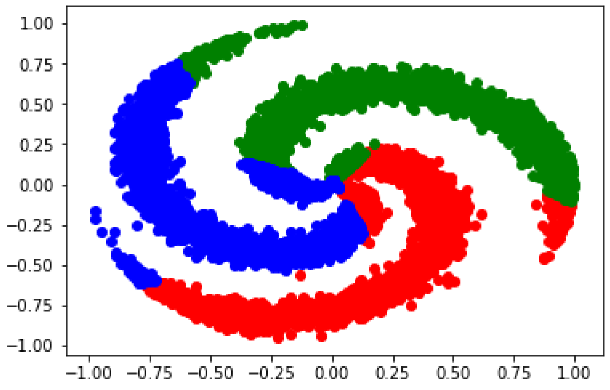
4.1 Linear followed by ReLU

ipython notebook: HingeLoss.ipynb
Linear followed by ReLU at each layer
Hinge loss in the output layer

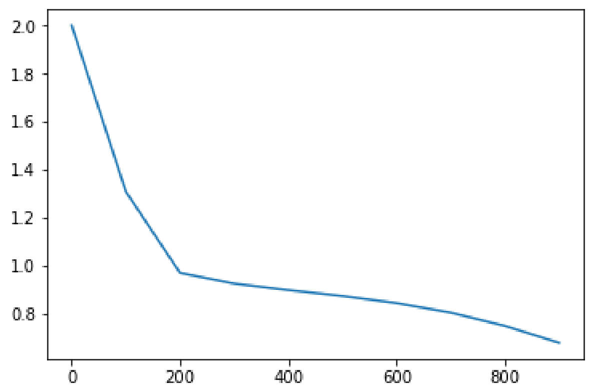
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.1 epochs = 1000
Train accuracy: 0.75



Test accuracy: 0.75



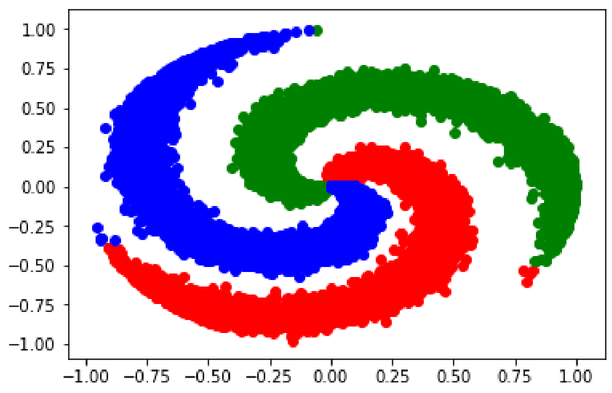
Train error plot:



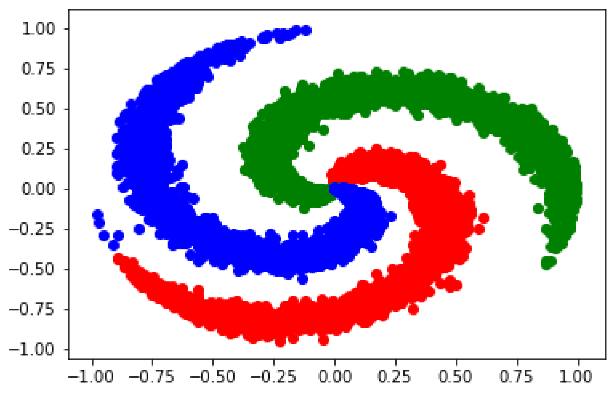
2. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

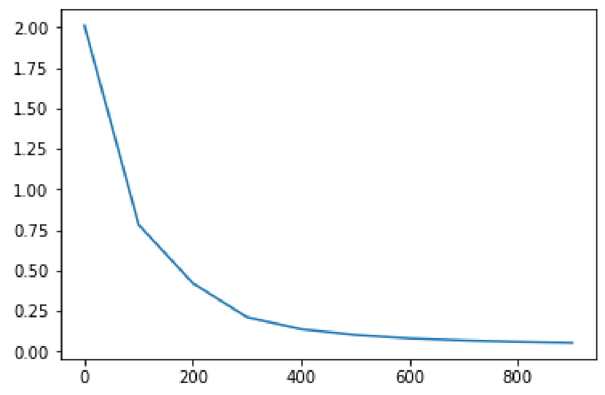
Train accuracy: 1



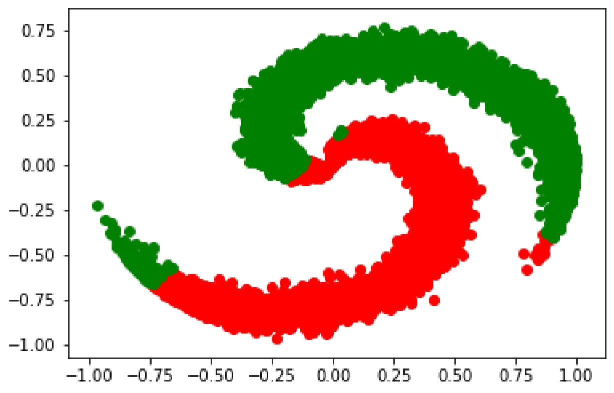
Test accuracy: 1



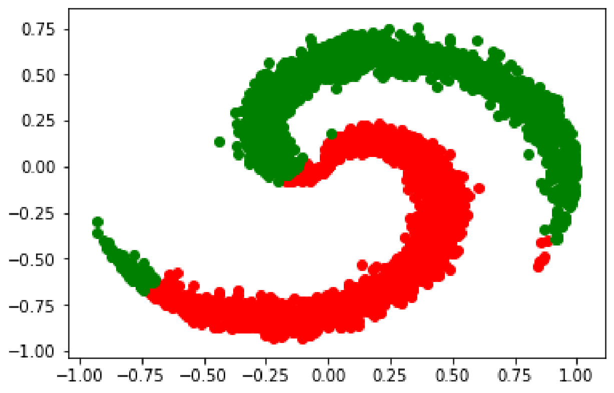
Train error plot:



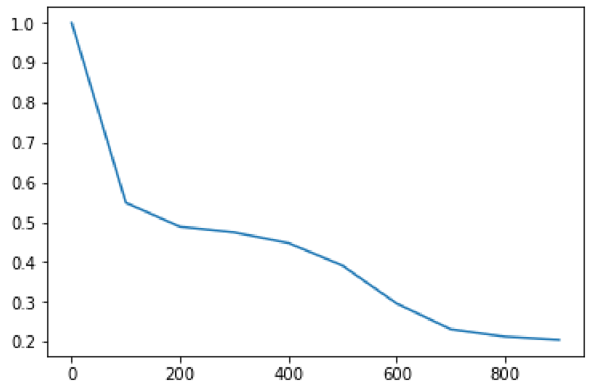
3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.1 epochs = 1000
Train accuracy: 0.92



Test accuracy: 0.91



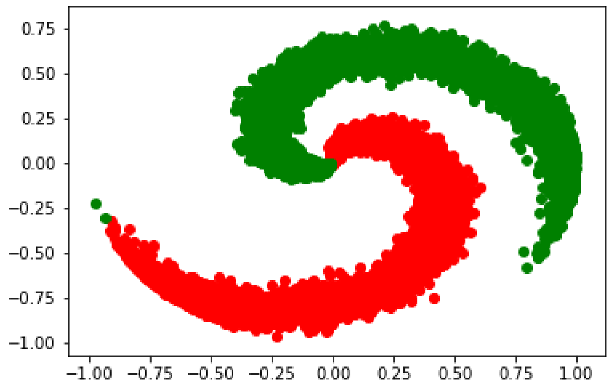
Train error plot:



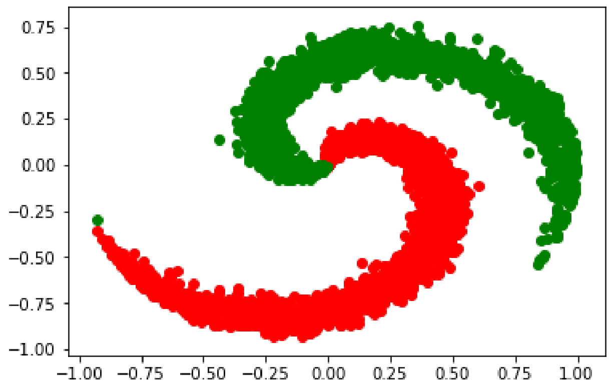
4. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

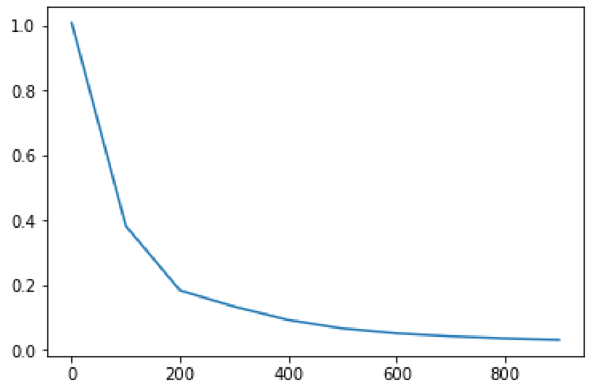
Train accuracy: 0.92



Test accuracy: 0.91



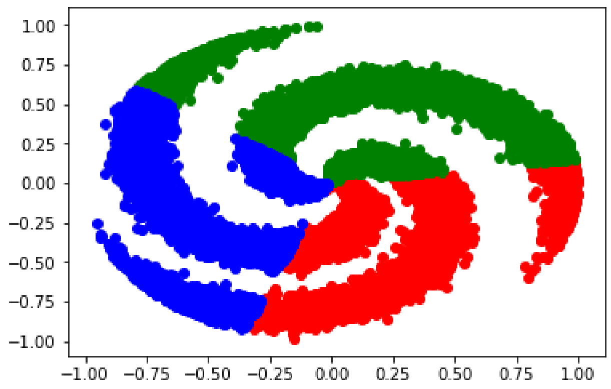
Train error plot:



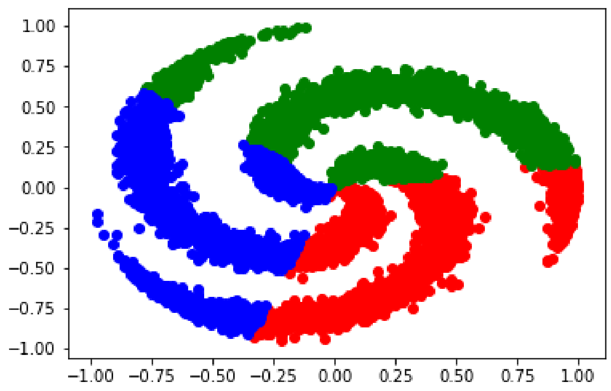
4.2 Sigmoid

ipython notebook: HingeLoss$_Sigmoid.ipynb$
$Sigmoidateachlayer$
$Hingelossintheoutputlayer$

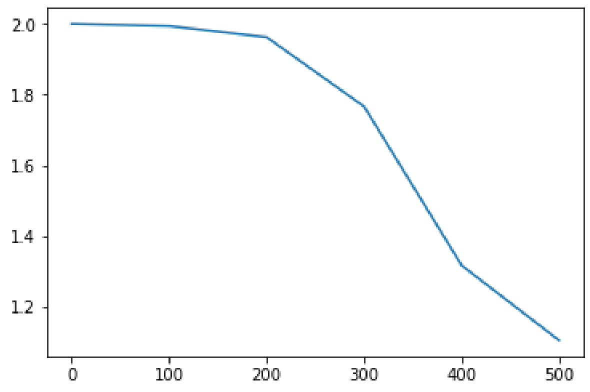
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.1 epochs = 600
Train accuracy: 0.53



Test accuracy: 0.52



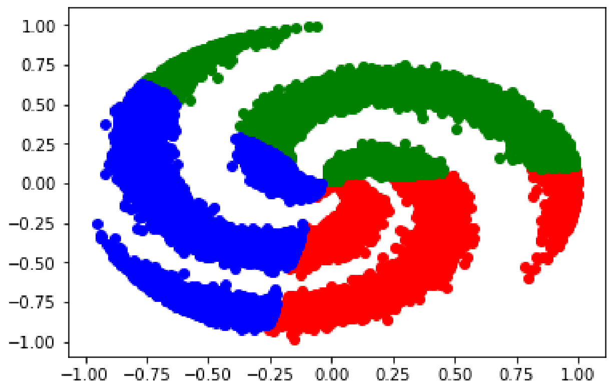
Train error plot:



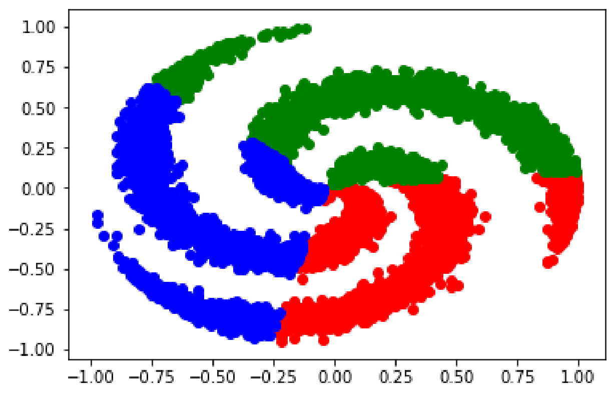
2. 3 fully connected layers, 3 classes

hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 600

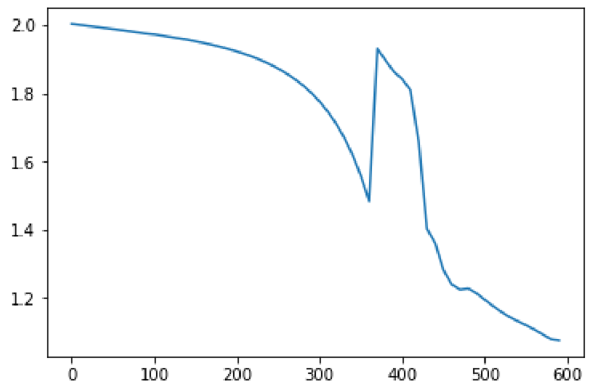
Train accuracy: 0.53



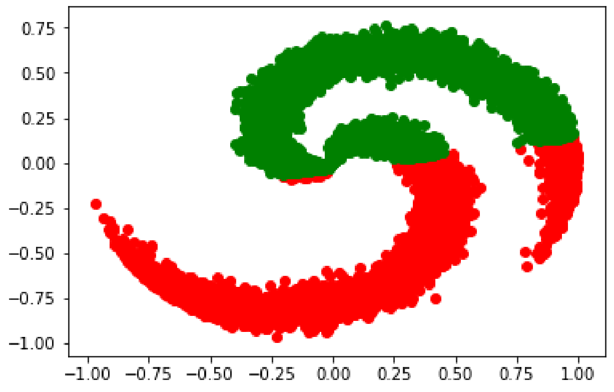
Test accuracy: 0.53



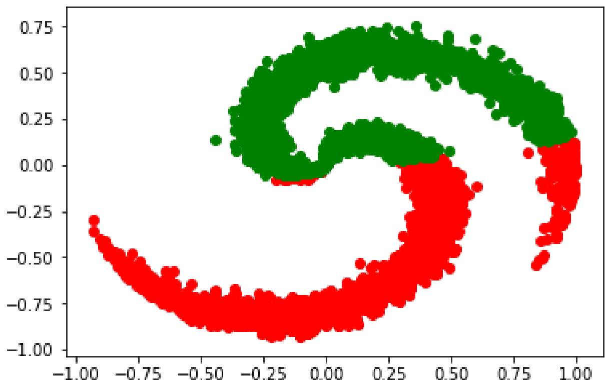
Train error plot:



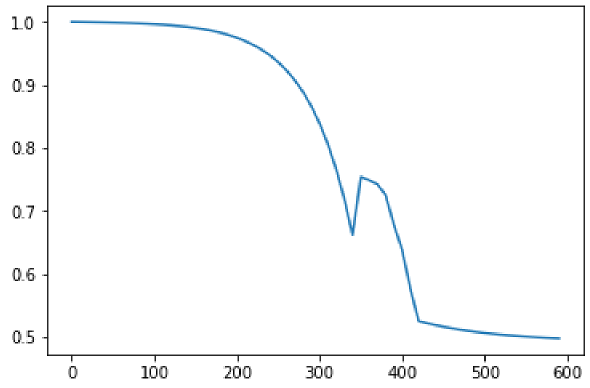
3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.1 epochs = 600
Train accuracy: 0.77



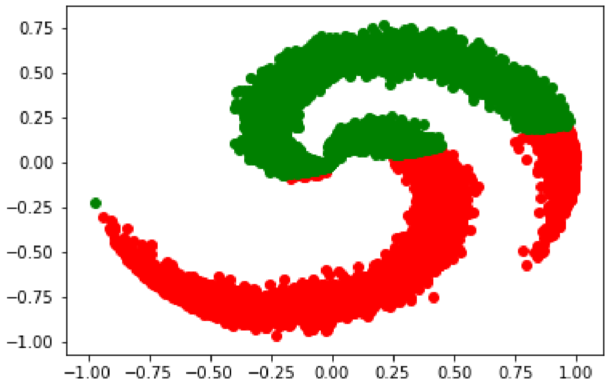
Test accuracy: 0.77



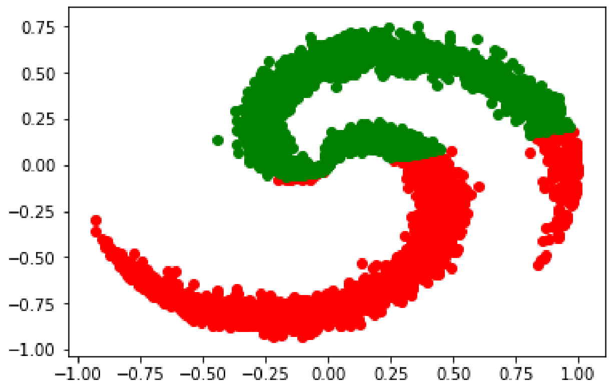
Train error plot:



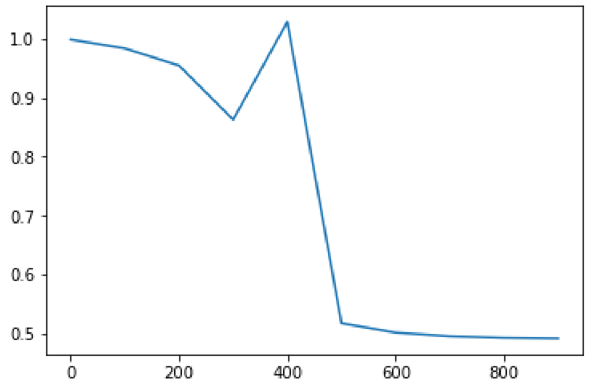
4. 3 fully connected layers, 3 classes
hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000
Train accuracy: 0.77



Test accuracy: 0.78



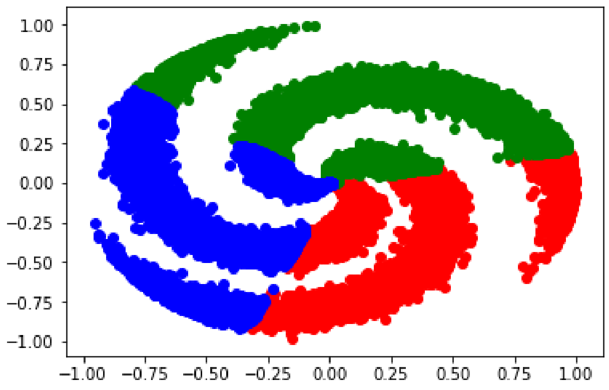
Train error plot:



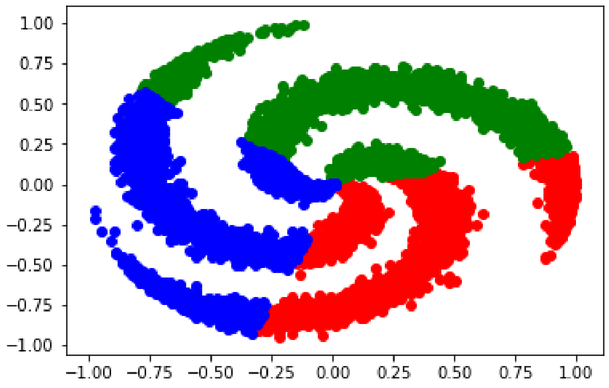
4.3 Tanh

ipython notebook: HingeLoss$_Tanh.ipynb$
$Tanhateachlayer$
$Hingelossintheoutputlayer$

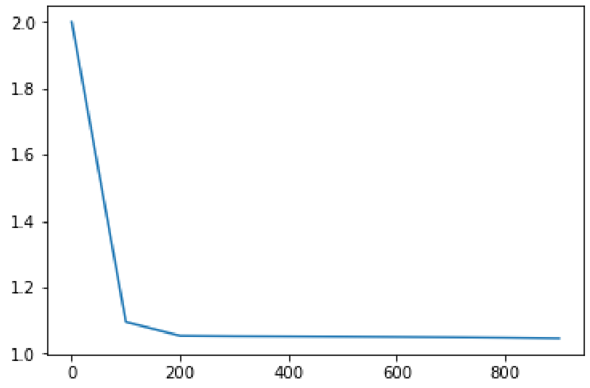
1. 2 fully connected layers, 3 classes
hidden layer size = 100 step size = 0.1 epochs = 1000
Train accuracy: 0.53



Test accuracy: 0.53

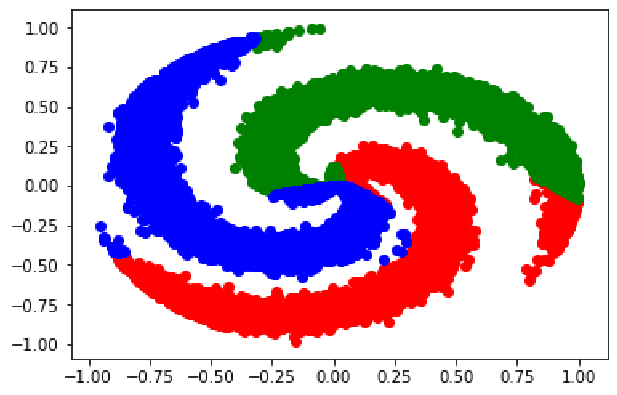


Train error plot:

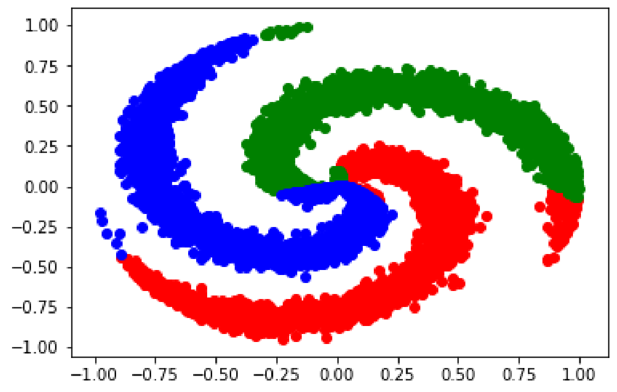


2. 3 fully connected layers, 3 classes
hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 1000

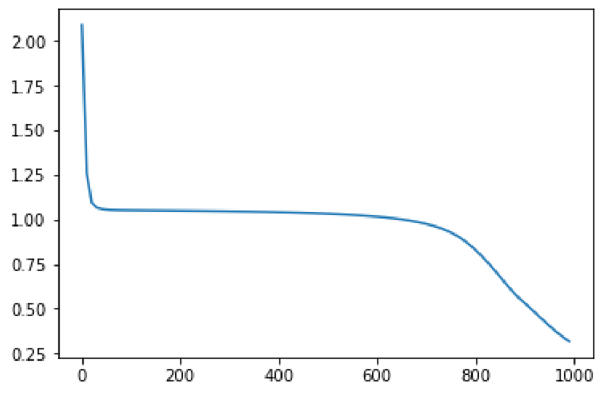
Train accuracy: 0.91



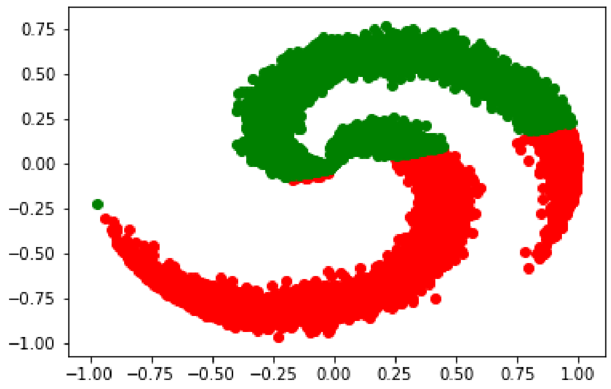
Test accuracy: 0.91



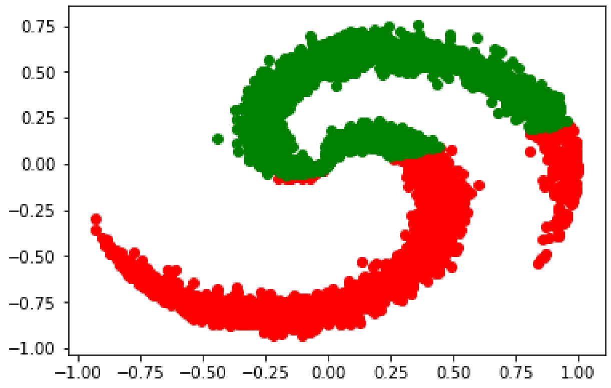
Train error plot:



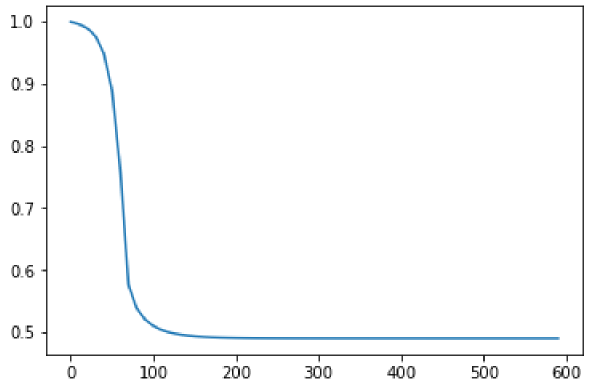
3. 2 fully connected layers, 2 classes
hidden layer size = 100 step size = 0.1 epochs = 600
Train accuracy: 0.77



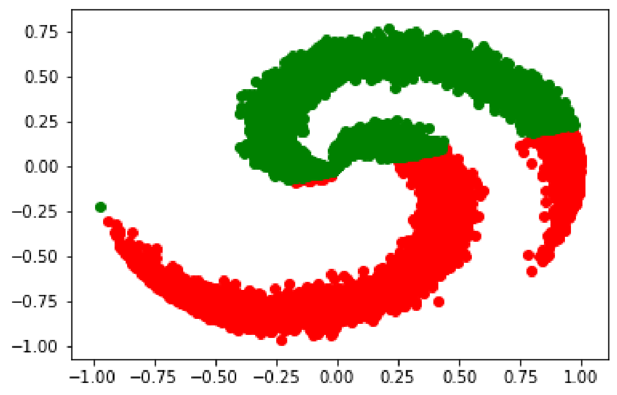
Test accuracy: 0.77



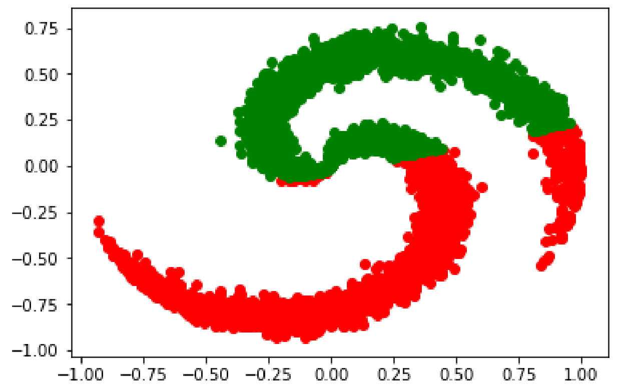
Train error plot:



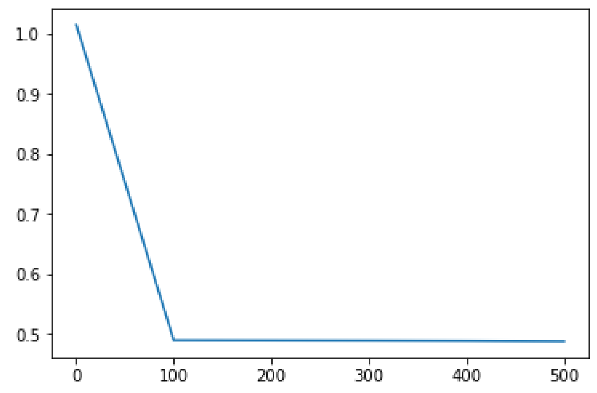
4. 3 fully connected layers, 3 classes
hidden layer 1 size = 100 hidden layer 2 size = 100 step size = 0.1 epochs = 600
Train accuracy: 0.77



Test accuracy: 0.77



Train error plot:



5 Observations and Conclusion

1. Hinge loss performs better than cross entropy loss. This is probably because the hinge loss aims at maximising the distance between each point and its corresponding class boundary.

2. Linear + ReLU performs the best, followed by Tanh. Sigmoid performs the worst. This is because of the saturating gradients problem. We also don't apply batch normalisation anywhere in our architectures.

3. 2 fully connected layers performs better than 3 fully connected layers for the 2 classes dataset. This is because 3 fc leads to overfitting. 3 fc performs better than 2 fc for the 3 classes dataset. This is because to classify 3 classes, a more complex function is needed.

4. Initialisation of weights and learning rate performs a huge rule in the fitting of the model and the final results.

5. Increasing the number of iterations can lead to better convergence and hence better results.

6 References

1. http://cs231n.github.io/neural-networks-case-study/
2. https://twice22.github.io/hingeloss/
3. https://deepnotes.io/softmax-crossentropy
4. https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c