Project 2 Mini deep-learning framework (EE-559)

Abdeljalil Moussa (Sciper 250755), Wenuka Gunarathna (Sciper 309398), Florian Genilloud (Sciper 271121)

May 22, 2020

Introduction

For this project, we have implemented a Sequential Neural Network Object which can be used to create a multi-layer, fully-connected (Dense) neural network with non-linearity layers Tanh or Rectified-linear (Relu).

Training and Testing Data

Training and testing data were generated using the function *"generate_ disc_ set"*. Two datasets were generated as following with the two definitions given in two versions of the mini-project document. Then the data were normalized. Each dataset is comprised of 1000 training samples and 1000 test samples. We will refer to the following definitions for the datasets hereafter for this report.

- **Dataset 1**: data corresponding to the disk centred at (0.5,0.5) - Figure 1
- **Dataset 2**: data corresponding to the disk centred at (0,0) - Figure 2

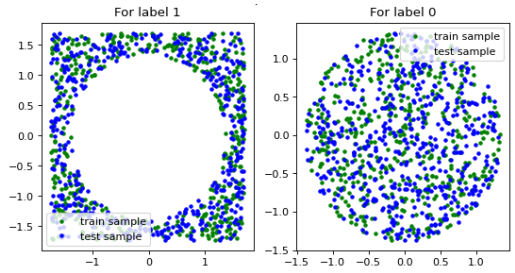

Figure 1: Label 1 and 0 for Disk centred at (0.5,0.5)

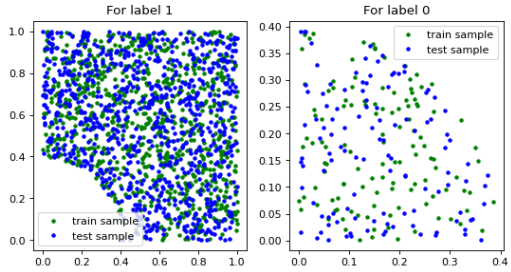

Figure 2: Label 1 and 0 for Disk centred at (0,0)

Neural Network Modules

Basic Neural Network Modules were developed to implement a Feed-Forward Neural Network without using the torch *"set_ grad_ enabled"* and only using the *"Tensor, matmul, zeros_ like, sum, argmax and abs"* function objects from torch module. Following is a list of all the modules used to implement the feed-forward neural network.

Linear Module

NNLinear class was defined to imitate the fully-connected layer of the feed-forward network. It can be initialized by only giving the input and output sizes. It has separate methods for the forward pass, backward pass, to output parameters and to set the learning rate to update its weights.

Tanh Module

NNTanh module was developed to use as a non-linear function to use in the feed-forward network. It has methods to facilitate the forward pass, backward pass and also a method to pass its parameters which is nothing but a Null array.

Relu Module

NNRelu module was developed also as a non-linear function to facilitate building a feed-forward network. It also has methods to execute the forward pass, backward pass and a parameter method as stated in the Tanh Module.

Sequential Module

The sequential module was designed to add any number of the above-mentioned modules to create a sequential model. The sequential module should be initialized with the list of modules that are needed to design the feed-forward network of your choice. Further, the learning rate can be dynamically (at initialization or in between any epoch) set to the sequential module. Mean Squared Error (MSE) or Mean Absolute Error (MAE) can be used as the loss function.

With an input of correct compatible size, the Sequential module can do the forward pass and return the output. Then with the intended output it will calculate the loss and do the backward pass to update the weights according to the Stochastic Gradient Descent (SGD) algorithm.

The sequential module also comes with an in-build method *"train_network"* which allows the sequential module to train for a given number of epochs using batch updates. By setting the *"early_stopping"* flag True, we can command the sequential module to stop training if the model is not improved in terms of loss over the last number of epochs. This number of epochs is set via a parameter named *"look_back_count_early_stopping"* in the code. In such a scenario, the sequential module will return the feed-forward network which achieved the lowest loss for an epoch. The training will also get terminated if the loss gets to zero.

Results

We have used two neural network models to generate results, namely :

- Relu model: two input units, two output units, three hidden layers of 25 units each with a Rectified Linear (Relu) non-linear function in-between
- Tanh model: two input units, two output units, three hidden layers of 25 units each with a Tanh non-linear function in-between

Model	**MSE: Dataset 1**	**MAE: Dataset 2**	**MSE:Dataset 2**	**MAE: Dataset 2**
Relu Model	1.84%	1.73%	0.28%	0.26%
Tanh Model	1.57%	1.89%	0.26%	0.30%

Table 1: error percentage for each model for each loss for each dataset

In generating results we have averaged the results over 20 samples for each model for each dataset mentioned above and the error rates for each such model are mentioned in Table 1.

When comparing the results we can clearly see the results are much better for the dataset 2 that the dataset 1. This is mainly due to the limited decision boundary length exist in dataset 2 as seen in Figure 3 and 4. However, apart from that, the results were almost identical for each dataset. For the dataset 1, the error rate was in-between 1.5% - 1.9% while for dataset 2 the error was in-between 0.26% - 0.30%. Figure 3 and 4 shows a sample output for the Relu model with MSE loss with prediction 1 labelled in green dots and prediction 0 is labelled in blue dots. The red-cross marks denote the incorrectly classified points.

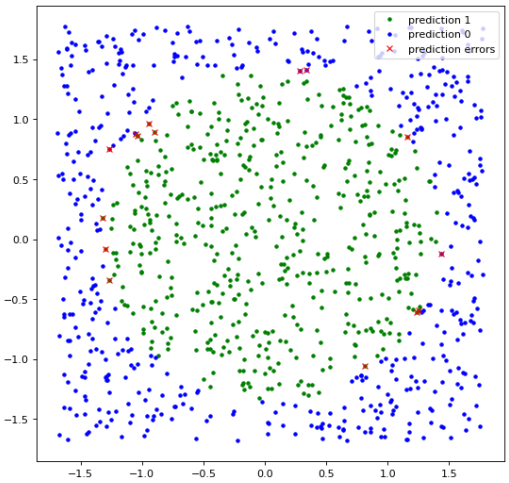

Figure 3: prediction for the dataset 1

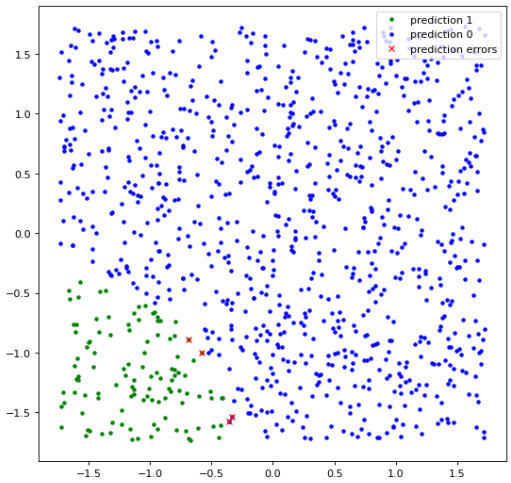

Figure 4: prediction for the dataset 2

Conclusion

In conclusion, the models using the manually implemented neural network modules are predicting the data as expected. The errors occurred at the edges of the decision boundary could be eliminated if we can use more data to train the model, giving the model a chance to understand the decision boundary better. However, this lower error rate was only possible because of the computer-generated data input giving a noise-less decision boundary. The losses MSE and MAE both have shown the equal ability in the classification as well as both the non-linear layers, namely Tanh and Relu. The early stopping criteria has also helped the error rate drop drastically by reducing the oscillation around the minimum loss.