Problem Set 1

CMSC 828L

Eleftheria Briakou

September 25, 2018

1 Description of Python implementation

In this section I briefly describe the steps followed to implement the back propagation algorithm. Specifically, I followed a more general implementation of a fully connected neural network for all the three first tasks of the problem set. The main functions of the submitted code are included in the above pseudocode.

Algorithm 1: My implementation

```
Data: eta, iterations
Result: trained neural network, results on test data
procedure DATA GENERATION
    return train data, test data
end procedure
initialize neural network;
procedure TRAIN(train data)
    for i ← 1 to iterations do
        for x in train do
            procedure FEED DORWARD(x)
                return alpha, z
            end procedure

            procedure BACK PROPAGATE(alpha,z)
                return delta
            end procedure

            procedure GRADIENT(alpha, delta)
                return weights derivatives, biases derivatives
            end procedure
        update the parameters of neural network;
end procedure
procedure APPLY(test data)
    return predictions
end procedure
procedure ANALYSIS(predictions)
    return statistical results
end procedure
```

2 Simple network

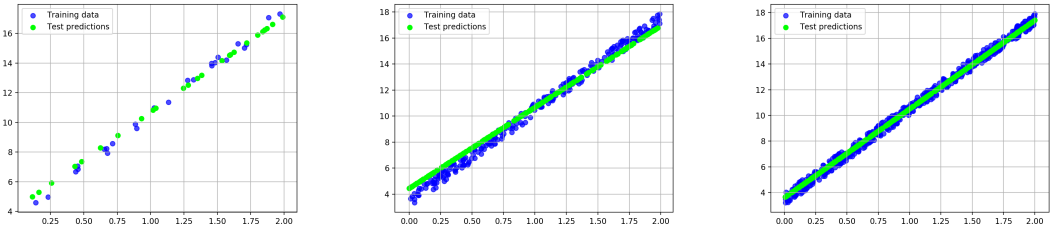
Data generation

In order to generate input data for the linear regression task, I started by picking N random samples from a uniform distribution over [0, 2). After that, each random sample was passed through the linear function $y = 7x + 3 + \xi$, where ξ is a Gaussian random variable. Furthermore, I followed the same procedure to create random data of higher dimensionality, where the expected output for a sample $x_1, x_2, ..., x_n$ was equal to $y = 7(x_1 + x_2 + ... + x_n) + 3 + \xi$. Finally, half of the created data were used for training the model, while the rest of them were used as a held-out dataset over which the performance of the model was evaluated.

Basic parameters to tune

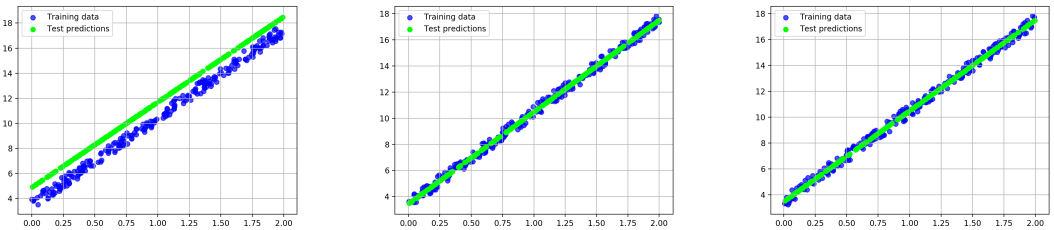
- N: number of training and test data used (0.5 split).
- i: number of iterations.
- η: step size.

2.1 1D input data



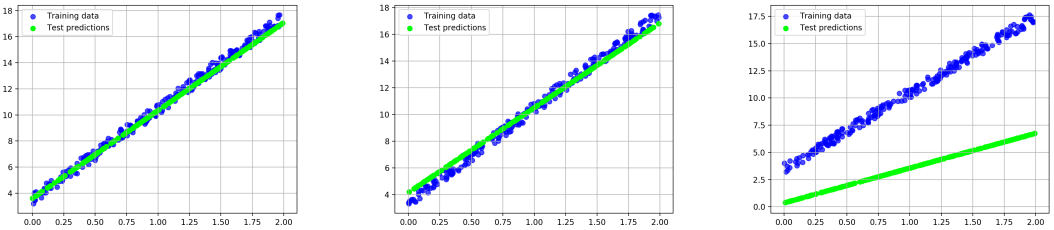
(a) $N = 50$ (b) $N = 500$ (c) $N = 1000$

Figure 1: Different size of data ($i = 10$, $\eta = 0.5$)



(a) $i = 1$ (b) $i = 100$ (c) $i = 500$

Figure 2: Different number of iterations ($N = 500$, $\eta = 0.5$)



(a) $\eta = 0.8$ (b) $\eta = 0.1$ (c) $\eta = 0.01$

Figure 3: Different step size ($N = 500$, $i = 20$)

2.2 ND input data

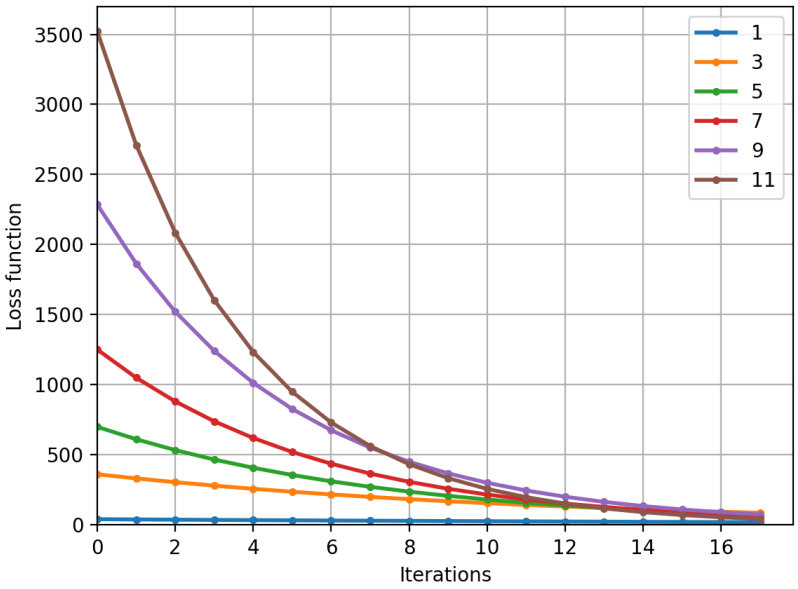


Figure 4: Performance comparison of simple neural network architecture on linear regression task for different dimensionality of input vectors ($\eta = 0.01$, $i = 20$, $N = 500$).

2.3 Hyperparameters

The choice of hyperparameters was easy for this task due to its simplicity (linear and convex problem). The neural network is guaranteed to converge to a good solution for this problem even when few data are presented to it for small number of iterations (Figures 1,2) given that the step size is sufficiently large. Moreover, the speed of convergence is associated with the step size that we use as we can see in Figure 3; for a small step size the neural network does not manage to converge in few iterations.

3 Shallow network

Data generation

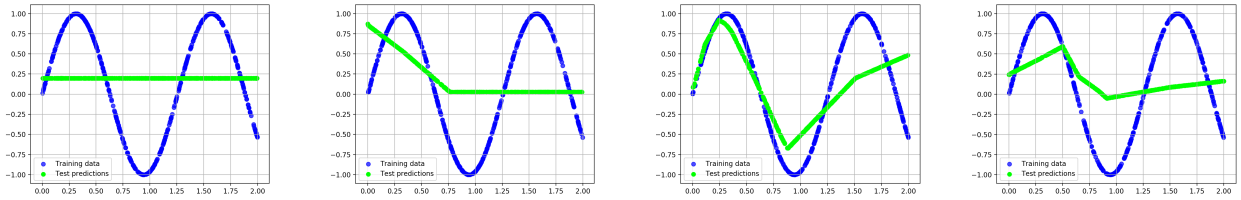
In order to generate input data for the non-linear regression task, I started by picking N random samples from a uniform distribution over [0, 2). After that, each random sample was passed through the linear function $sin(5x)$. Furthermore, I followed the same procedure to create random data of higher dimensionality, where the expected output for a sample $x_1, x_2, ..., x_n$ was equal to $y = sin(5x_1 + 5x_2 + ... + 5x_n)$. Again, half of the created data were used for training the model, while the rest of them were used as a held-out dataset over which the performance of the model was evaluated.

System configuration

- $N = 1000$[1] (500 for train, 500 for test)
- $i \in [10, 5000]$
- $\eta \in [0.5, 0.01]$
- $k_1 \in [5, 100]$ (number of neurons in the unique hidden layer)

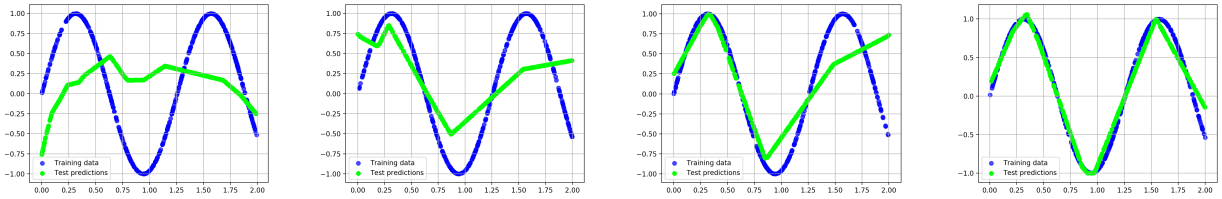
[1] I keep the data size sufficientyl large and stable for the above experiments.

3.1 1D input data



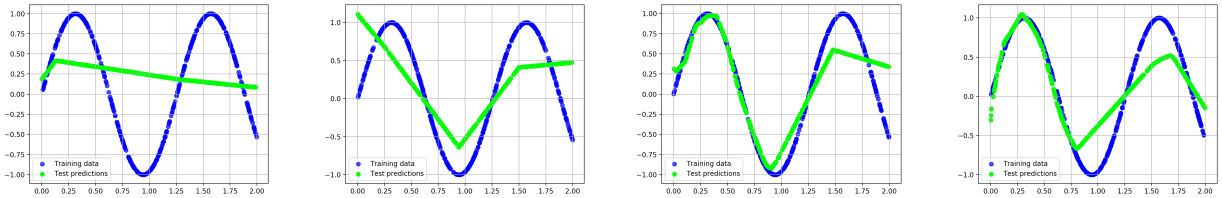
(a) $\eta = 0.5$ (b) $\eta = 0.1$ (c) $\eta = 0.01$ (d) $\eta = 0.001$

Figure 5: Different step size ($N = 1000$, $i = 1000$, $k_1 = 50$)



(a) $i = 100$ (b) $i = 2000$ (c) $i = 5000$ (d) $i = 10000$

Figure 6: Different number of iterations ($N = 1000$, $\eta = 0.01$, $k_1 = 50$)



(a) $k_1 = 5$ (b) $k_1 = 20$ (c) $k_1 = 100$ (d) $k_1 = 200$

Figure 7: Different number of nodes on hidden layers ($N = 1000$, $\eta = 0.01$, $i = 1000$)

3.2 ND input data

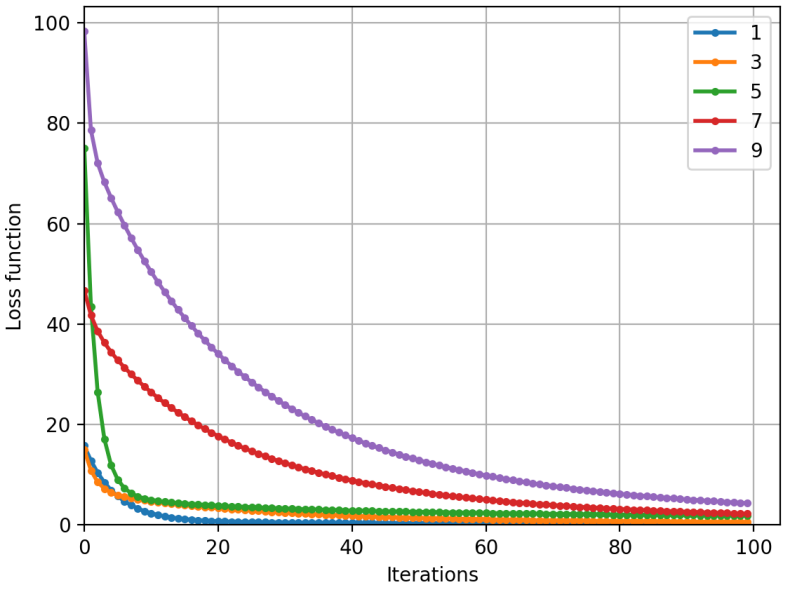


Figure 8: Performance comparison of shallow neural network architecture on non-linear regression task for different dimensionality of input vectors ($\eta = 0.001$, $i = 100$, $N = 1000$, $k_1 = 50$).

To check the effectiveness of the shallow neural network algorithm on problems for which data dimensionality is greater than 1 I contacted the above experiment. Specifically, I chose different dimensions and tried to solve the problem via iterating for a few epochs and keeping the hyperparameters for all the experiments the same. Figure 8 shows the results of this experiment. It can be noticed that the loss function decreases through time for all the problems; although, it corresponds to very high values for problems of higher dimensions for the first few iterations. Furthermore the loss function converges to bigger values for higher dimensionalities.

3.3 Hyperparameters

Generally the problem was more difficult than that of the previous step due to its non-linear nature. Although, finding the appropriate set of hyperparameters for fitting the sine wave in 1-dimensional data did not require a lot of time since it converged to an adequate solution after little experimentation (meaning that I didn't have to go through a grid search for choosing them). However, the higher dimensional problem was more difficult to fit since the loss function needed more time to converge to a sufficiently low value.

4 Deep neural network

At this stage I implemented a deep neural network with an arbitrary number of hidden layers and an arbitrary number of nodes in each hidden layer. I experimented with various values for the hyperparameters ending up the with the following general configurations.

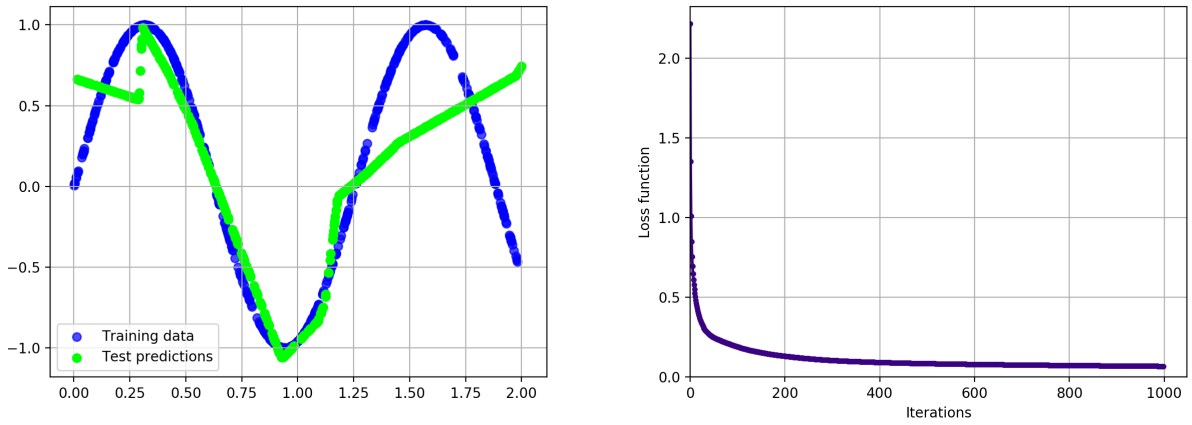
System configuration

- $N = 1000$ (500 for train, 500 for test)
- $i \geq 1000$
- $\eta = 0.01$
- $\vec{k} = [k_1, k_2, ..., k_n]$, where $n \in [10, 20]$ (number of neurons in the each of the hidden layers)

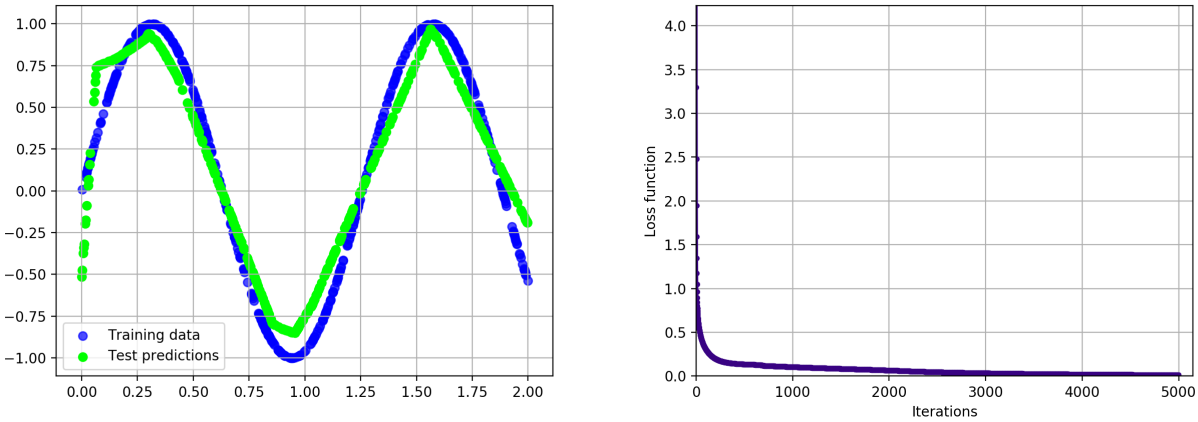
  Generally I chose the number of nodes per hidden layer so that the number of the parameters that the model has to learn will be comparable to the number of parameters that correspond to the shallow network case.

4.1 1D input data

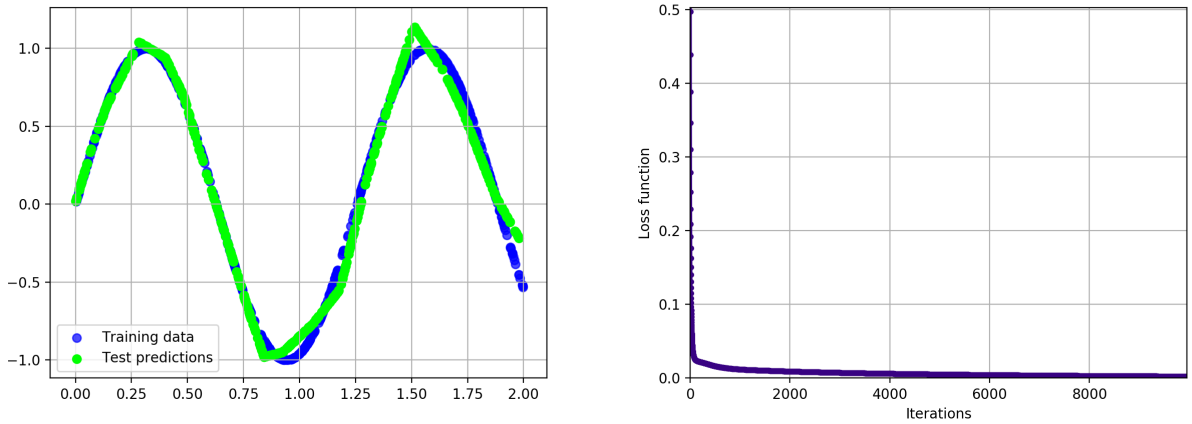
4.1.1 3 hidden layers



(a) $i = 1000$



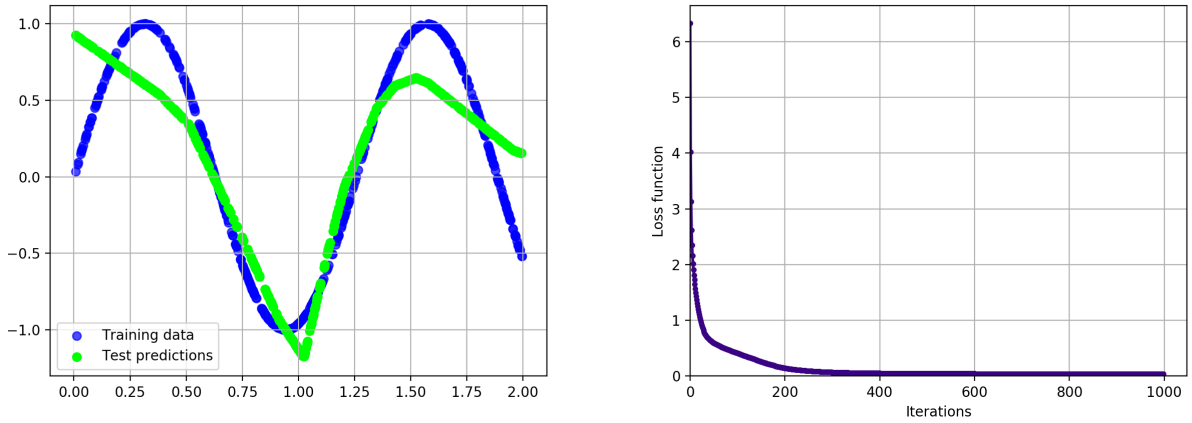
(b) $i = 5000$



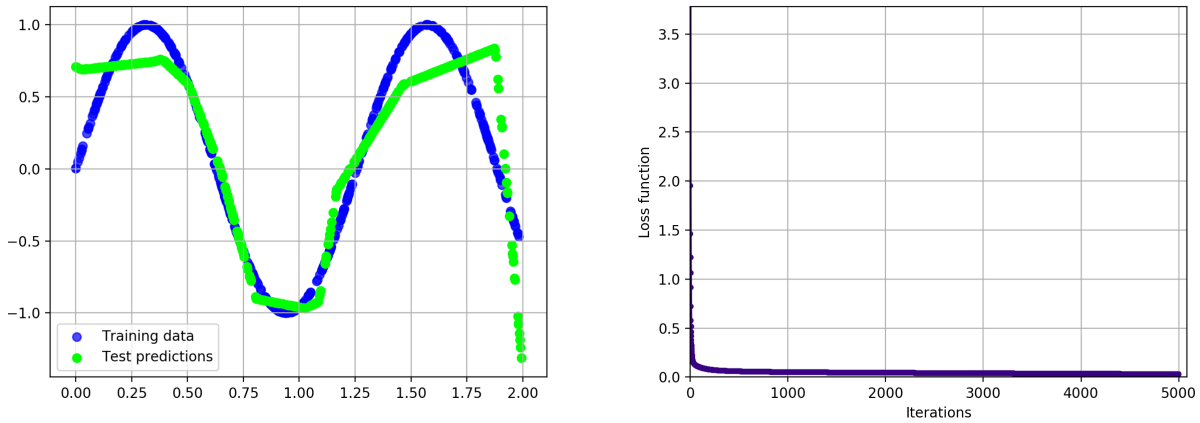
(c) $i = 10000$

Figure 9: Different number of iterations ($N = 1000$, $\eta = 0.001$, $\vec{k} = [20, 10, 20]$)

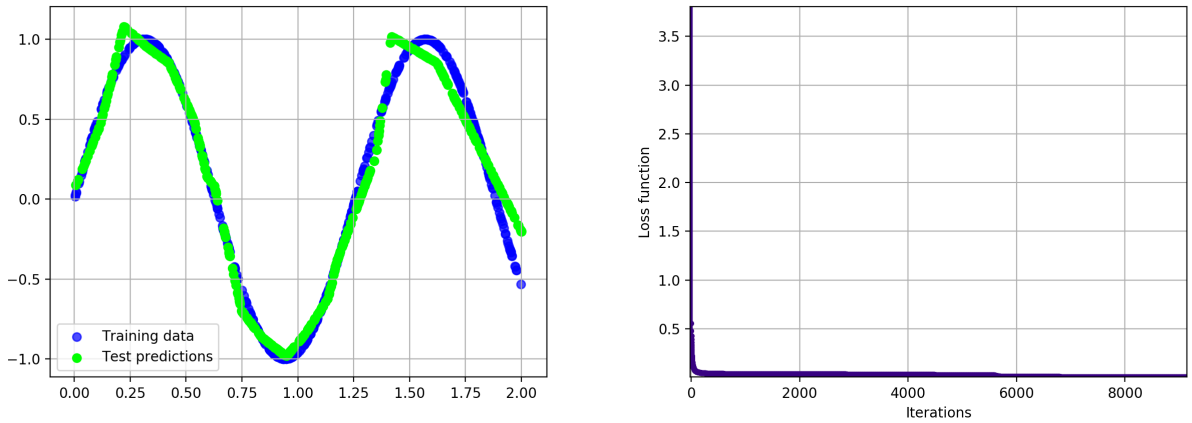
4.1.2 5 hidden layers



(a) $i = 1000$



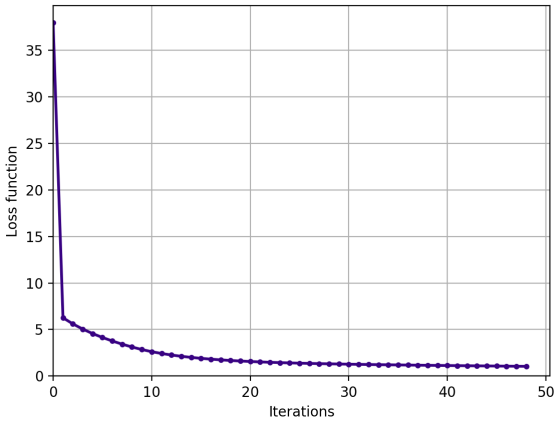
(b) $i = 5000$



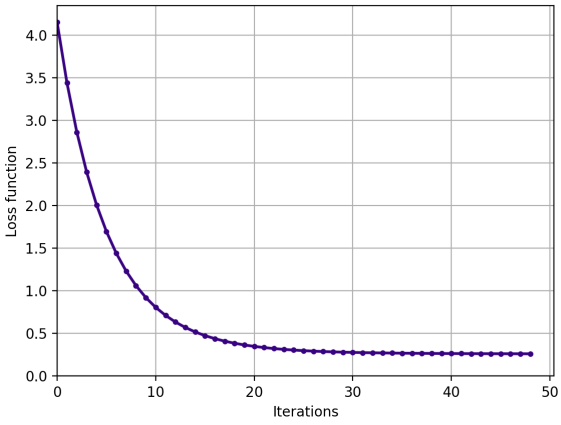
(c) $i = 10000$

Figure 10: Different number of iterations ($N = 1000$, $\eta = 0.001$, $\vec{k} = [10, 10, 5, 10, 10]$)

4.2 2D input data



(a) $\vec{k} = (20, 10, 20)$



(b) $\vec{k} = (10, 10, 5, 10, 10)$

Figure 11: Loss function for 2D data and different deep configurations ($N = 1000$, $\eta = 0.001$, $i = 50$)

4.3 Hyperparameters

As the width of the neural network increases more parameters need to be tuned in order to achieve the desirable results for a specific task. Furthermore, the extra hidden layers introduce non-linearities that increase the complexity of the output functions and are sometimes really hard for a human to interpret. For these reasons, the tuning of a deeper network is harder when compared to the tuning of a shallow network.

4.4 Depth VS Convergence

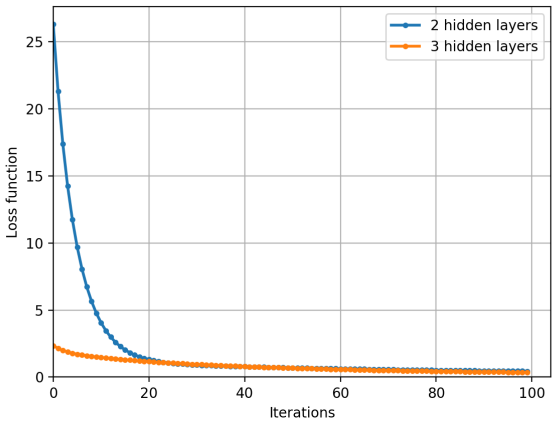


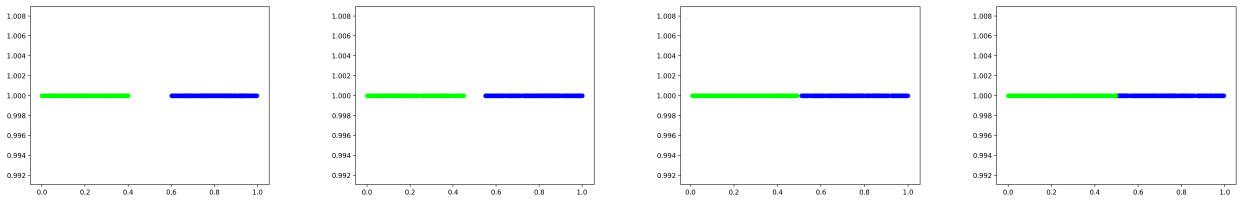
Figure 12: Loss for 2D data for deep architectures with different widths ($N = 1000$, $\eta = 0.001$, $i = 100$)

In Figure 12, I compare the loss function for a neural network with 2 hidden layers with $\vec{k} = [10, 10]$ and a neural network with 3 hidden layers with $\vec{k} = [7, 7, 7]$. I chose those number of nodes per layer so that both networks will have to 'learn' almost the same number of parameters and the comparison could be fair. Based on the results, the deeper neural network converges faster.

5 Classification

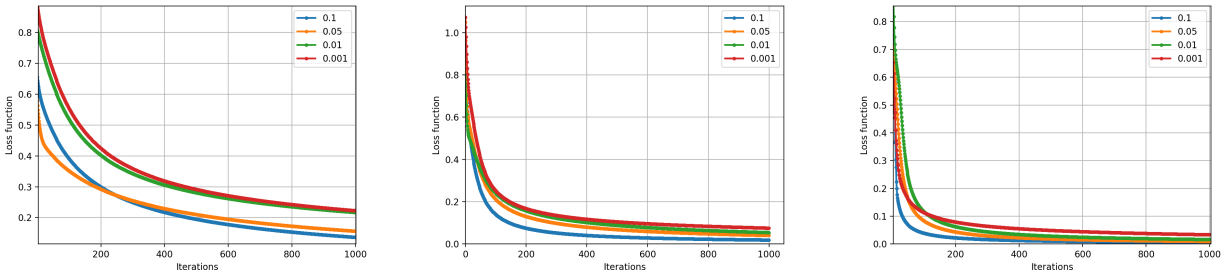
5.1 1D input data

5.1.1 Linear Separable



(a) $margin = 0.1$ (b) $margin = 0.05$ (c) $margin = 0.01$ (d) $margin = 0.001$

Figure 13: 1D linear separable data with different margins ($N = 500$)

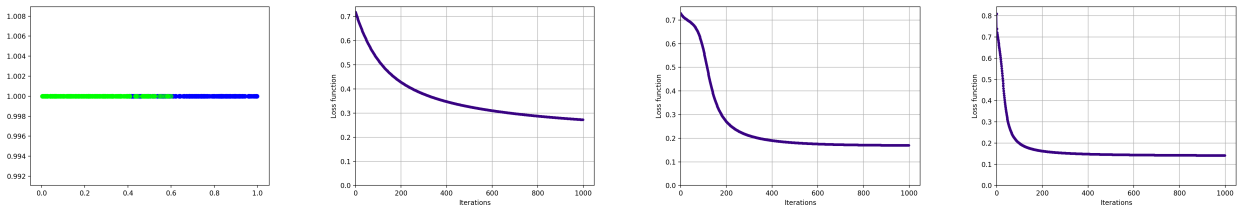


(a) Simple (b) Shallow (c) Deep

Figure 14: Performance comparison between different neural network architectures applied to a binary classification problem for linear 1D data separated by a margin.

Figure 17 depicts the loss functions for three different neural network architectures applied to the problems presented in Figure 16. The simple neural network consists of only an input and one output layer, the shallow neural network consists of one hidden layer with 2 nodes, while the deep one has two hidden layers with 2 and 3 nodes correspondingly. Generally, one can observe that the speed with which the networks converge to good solutions depends on the margin. Specifically, the biggest margin as well as the addition of hidden layers lead to faster convergences.

5.1.2 Non-linear separable



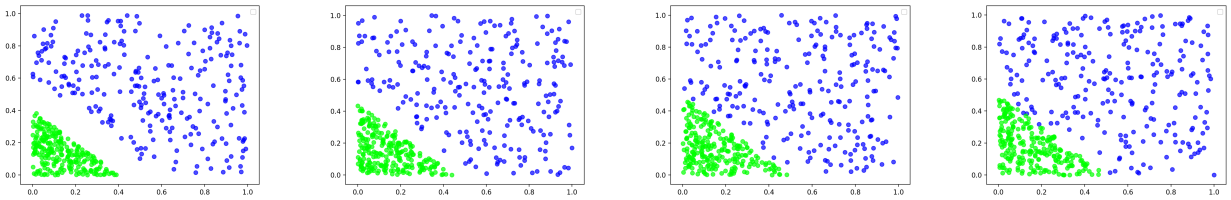
(a) Data (b) Simple (c) Shallow (d) Deep

Figure 15: 1D non-linear separable data and performance of different neural network architectures.

For the non-linear separable case I observe that the loss function is bigger than that of the linear separable case as expected. Also the increase in the number of hidden layers improves the performance as depicted in 18.

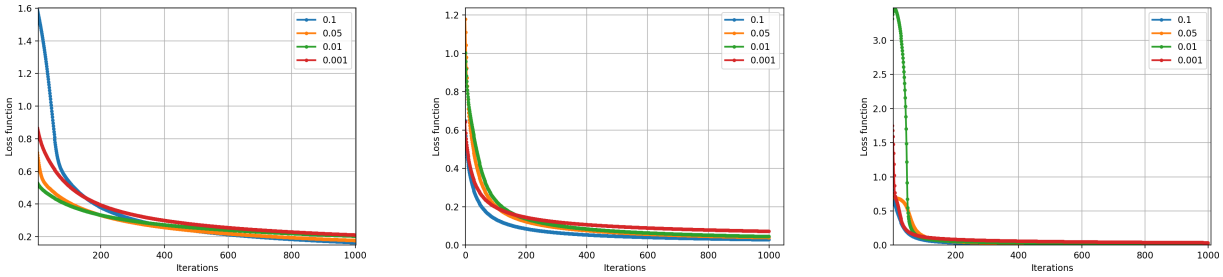
5.2 2D input data

5.2.1 Linear Separable



(a) $margin = 0.1$ (b) $margin = 0.05$ (c) $margin = 0.01$ (d) $margin = 0.001$

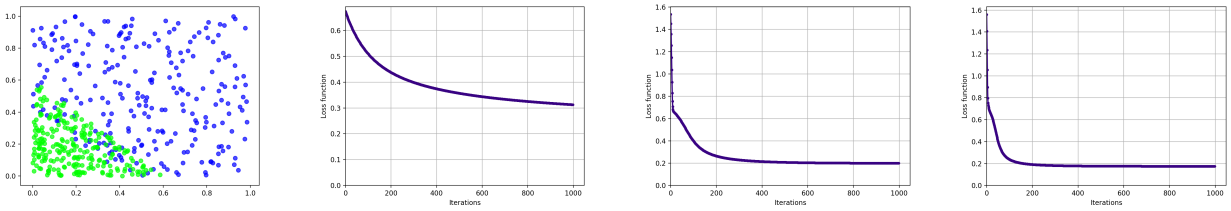
Figure 16: 2D linear separable data with different margins ($N = 500$)



(a) Simple (b) Shallow (c) Deep

Figure 17: Performance comparison between different neural network architectures applied to a binary classification problem for linear 2D data separated by a margin.

5.2.2 Non-Linear Separable



(a) Data (b) Simple (c) Shallow (d) Deep

Figure 18: 2D non-linear separable data with different margins ($N = 500$)