Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

Learning and generalisation in feed-forward networks — from perceptron learning to backprop

Yu Hu, Omar E. Contreras Z., Boyu Li

Jan 27, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to design and apply networks in classification, function approximation and generalisation tasks
- to understand single and multi-layer networks by implementing them from scratch
- to observe the limitations and how the parameters affect the performance

2 Methods

For the first part of the assignment, everything was implemented from scratch in python and for the second part of the assignment, matlab was used.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron *(ca.1 page)*

In Fig 1, the difference between Delta learning and perceptron learning is shown, where the boundary for Delta rule is in the middle of the two cluster and the perceptron barely classifies correctly each class.

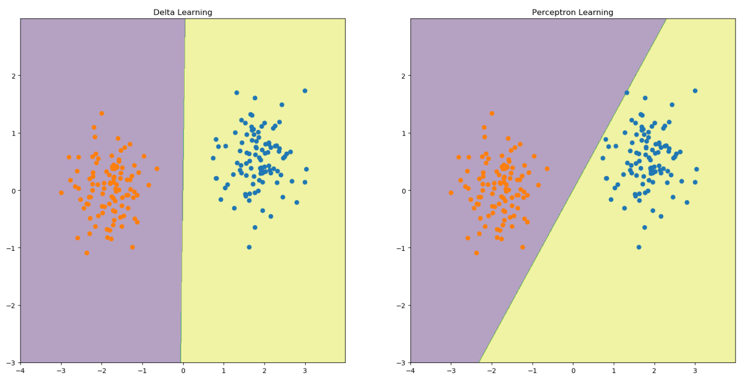


Figure 1: Plot of delta learning(left) and Perceptron learning(right), for 55 epochs, with batch learning

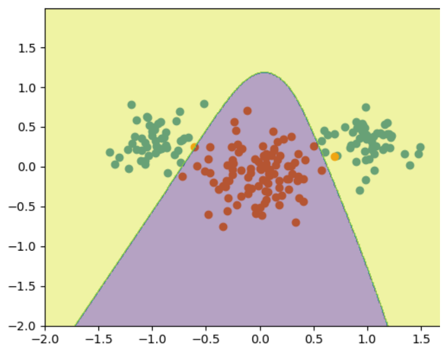
Sequential learning converges faster than batch learning for both, perceptron and delta learning, however, sequential learning takes more time per epoch.

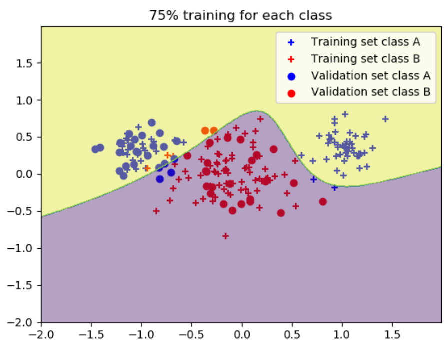
Without bias, the network is unable to correctly classify clusters with a boundary that doesn't pass through the origin, so is data dependent the network.

3.2 Classification and regression with a two-layer perceptron *(ca.2 pages)*

3.2.1 Classification of linearly non-separable data

The number of hidden nodes affects the performance of the network. For the classes shown in Figure 2a, the minimum needed was 2.


(a) 4 nodes in hidden layer for a non linearly separable data



(b) Data set was divided in training and validation set, 25 percent of each class was used for testing

Figure 2b shows how splitting the data doesn't affect the network if the data used for training correctly represents the clusters for each of the classes.

3.2.2 The encoder problem

The network always converge and map inputs to themselves. As it shown in the figure below, output has the same result with target. The internal code compresses the 8-dimensional data into 3-dimension, then the output layer is like a decoder that map the binary number to 8-dimensional data.The weight matrix of the first layer is like a truth table. When the size of hidden layer is two, the network can also converge, that's because we added bias into the hidden layer, actually the size of the network is three. The autoencoder can be used as data compressor.

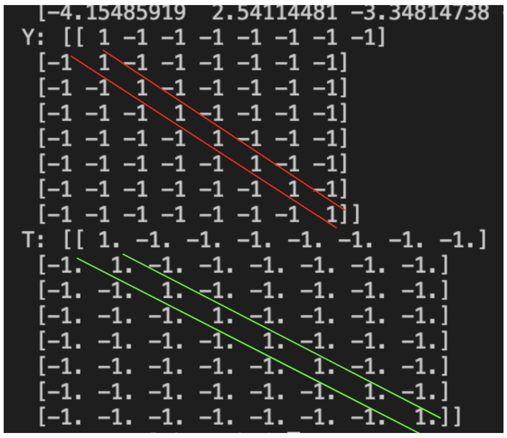

```
  [-4.15485919  2.54114481 -3.34814738
Y: [[ 1 -1 -1 -1 -1 -1 -1 -1]
  [-1  1 -1 -1 -1 -1 -1 -1]
  [-1 -1  1 -1 -1 -1 -1 -1]
  [-1 -1 -1  1 -1 -1 -1 -1]
  [-1 -1 -1 -1  1 -1 -1 -1]
  [-1 -1 -1 -1 -1  1 -1 -1]
  [-1 -1 -1 -1 -1 -1  1 -1]
  [-1 -1 -1 -1 -1 -1 -1  1]]
T: [[ 1. -1. -1. -1. -1. -1. -1. -1.]
  [-1.  1. -1. -1. -1. -1. -1. -1.]
  [-1. -1.  1. -1. -1. -1. -1. -1.]
  [-1. -1. -1.  1. -1. -1. -1. -1.]
  [-1. -1. -1. -1.  1. -1. -1. -1.]
  [-1. -1. -1. -1. -1.  1. -1. -1.]
  [-1. -1. -1. -1. -1. -1.  1. -1.]
  [-1. -1. -1. -1. -1. -1. -1.  1.]]
```

Figure 3: output and target

3.2.3 Function approximation

For the function approximation, the best model was the one with 25 hidden nodes, when the model had 1 or 2, it was uncapable of approximating the function, from 3 it starts to predict similarly, however, the smaller MSE was obtained using 25 hidden units(prediction shown in Figure 4).

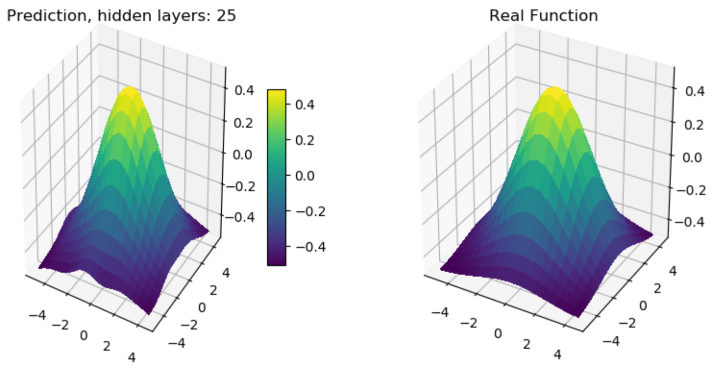


Figure 4: Best model with smaller MSE using 25 hidden units

Varying the number of training samples from .80 to .20 of all dataset, the result was graph in Figure 5.

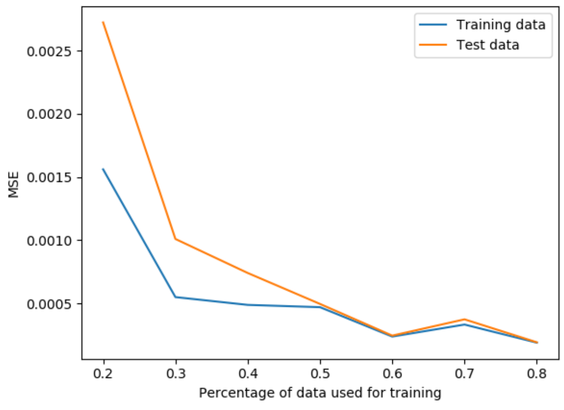


Figure 5: Training and test set error plotted against the percentage of training data used for the best model (25 hidden units).