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

Ravi Bir—ravib@kth.se
Bharat Sharma—bsharma@kth.se
Qiao Ren—qiaor@kth.se

January 31, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement a one-layer and two-layer perceptron for classification, function approximation and generalisation tasks
- to investigate their convergence, limitations, and properties that affect their performance, including learning rate and number of hidden nodes. Investigations performed on linearly separable and non-separable data
- to investigate multi-layer perceptron networks for chaotic time-series prediction

2 Methods

Part 1 - Python with Jupyter Notebooks. Libraries for Numpy , MatplotLib, Mpl toolkit
Part 2 - TensorFlow, Keras

3 Results and discussion - Part I

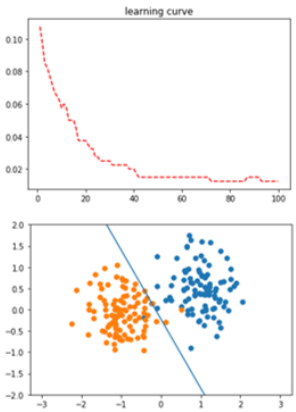
3.1 Classification with a single-layer perceptron

Below is a table showing the average number of epochs until convergence on the linearly separable data for the three different learning techniques (Figures 1-3). Different values for the learning rate are tested. Each learning technique is tested on 3 random initialisations for each learning rate, with the average number of epochs until convergence being calculated and recorded. Each test is run for a maximum of 100 epochs. Convergence is defined by the mean squared error remaining almost unchanged for 3 epochs.

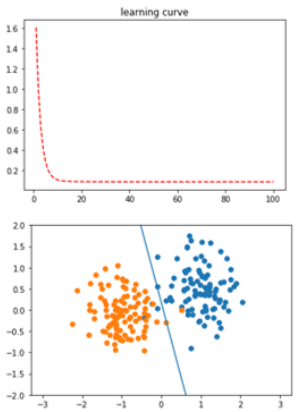
Learning Rate	0.005	0.001	0.0001
Perceptron Batch	44.8	42.7	50.9
Delta Batch	12.3	10.6	14.4
Delta Sequential	16.6	13.1	17.8

Tabell 1: Number of epochs taken to converge for different learning rates for each learning technique

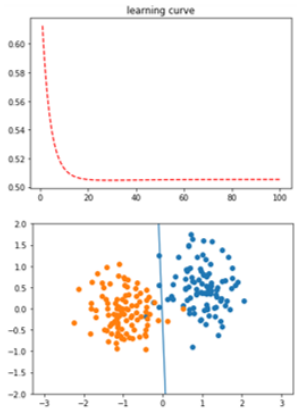
Below are examples of the learning curves and decision boundaries that are produced for each learning technique at a learning rate of 0.001, after 100 epochs. The two different coloured points represent the two different classes. On the learning curve, we have plotted mean squared error against number of epochs.



Figur 1: Learning curve Perceptron



Figur 2: Learning curve Delta Rule Batch



Figur 3: Learning curve Delta Rule Sequential

As you can see from the results in Table 1, learning rates of 0.005 and 0.001 give the fastest converge time for each learning technique, with a learning rate of 0.0001 always returning the slowest convergence time. This is because the steps it takes are too small. A learning rate larger than 0.005 would sometimes not converge because the steps it takes are too big, and it would miss the minimum.

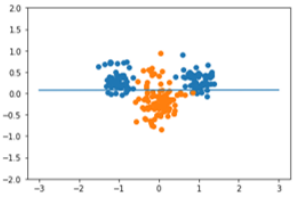
Comparing the learning curves of the three learning techniques and looking at Table 1, we can see that Perceptron takes the longest time to converge, followed by Delta sequential, and then Delta batch converges the fastest. Different random insitialisations of the weights show the same overall trends and does not have a huge affect on results. The main factor that affects the speed of convergence is the different learning rates. When bias is removed and the Delta batch rule is used to classify the data, it can only do so if the decision boundary can go through the origin (0,0). If it is not able to do this, then it can not classify the data.

3.2 Classification on non-linearly separable data

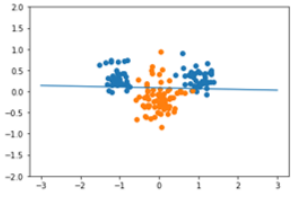
We will now test the Delta batch rule on non linearly separable data, using a set learning rate of 0.001, run for 100 epochs. Both the accuracy (Table 2) and decision boundaries produced (Figure 4-6) are shown. Accuracy is calculated as (number of correctly classified points in the class) / (total number of points in the class)

Dataset	Accuracy of Class A	Accuracy of Class B
Nothing removed	0.90	0.85
Remove Random 25(%) from each class	0.92	0.85
Remove Random 50(%) from class A	0.66	0.88
Remove Random 50(%) from class B	0.98	0.80
Remove 20(%) from a subset of classA for which classA(1,:)¡0 and 80(%) from a subset of classA for which classA(1,:)¿0	0.78	0.89

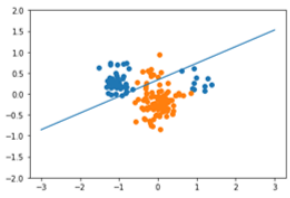
Tabell 2: Accuracy on Non Linearly Separable Data



Figur 4: Nothing Removed



Figur 5: Remove 25(%) from each class



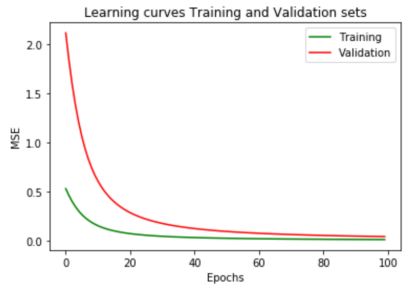
Figur 6: Remove subsets of class A

When points are removed from a class, the boundary is less influenced by that class and the accuracy for it decreases. It is obvious from the results that these linear boundaries can not successfully separate the two classes, so a two layer perceptron is needed.

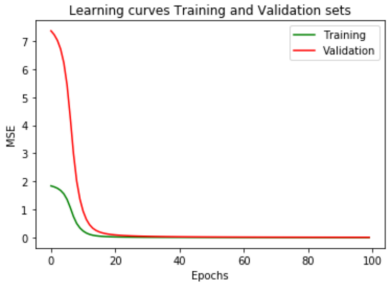
3.3 Classification and regression with a two-layer perceptron *(ca.2 pages)*

3.3.1 Classification of linearly non-separable data

We investigated how the number of nodes affects convergence. When a large learning rate is used, convergence can be seen when 2 nodes are used. As the number of hidden nodes increases, convergence is faster. However, too many hidden nodes leads to overfitting. A learning rate of 0.009 was used. This value would avoid the error diverging, while still being high enough that it would not remain in a local minma. The validation error always began as higher than the training error, but both the training sets and the validation sets reach a mean squared error of 0, and this occurred faster when the number of nodes was higher, as seen in Figure 7 and 8. Below we plotted the learning curves for testing and validation sets using the MSE. Both used a learning rate of 0.009 and ran for 100 epochs, but used either 4 or 25 nodes.



Figur 7: Learning curves, 4 nodes



Figur 8: Learning curves, 25 nodes

3.3.2 The encoder problem

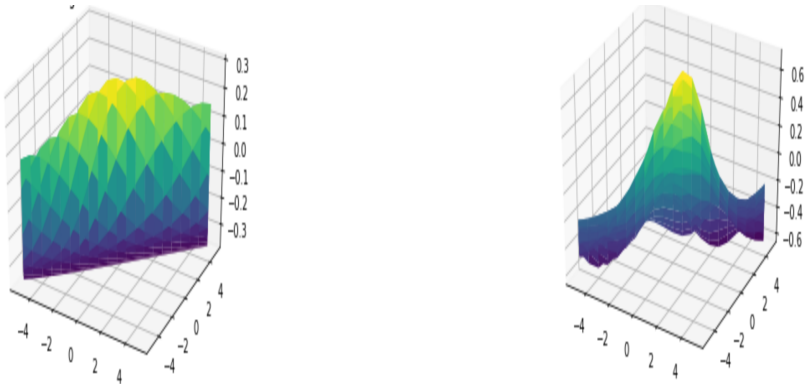
Our 8-3-8 encoder didn't always converge when we kept the value of learning rate low(0.001) but noticed if we increase the learning rate(0.9), it started to converge between 20-30 epochs.

The internal code represents the activation of the hidden layer. The weight matrix has some negative values which means those nodes were not used for activation of the hidden layer, but over all epochs, each weight has a positive value, hence they all were used.

Changing the number of input nodes from 3 to 2 makes the algorithm take more epochs to converge. For the same learning rate as above, it takes now 40-50 epochs to converge.

Auto-encoders can be used for dimensionality reduction.

3.3.3 Function approximation



(a) 2 Nodes in Hidden layer

(b) 25 Nodes in Hidden layer

Figur 9: Function approximation with the 2-layer perceptron.

When the hidden nodes are < 5, the network doesn't approximate the Gaussian function very well. Increasing the number of hidden nodes to 20-25 yields a much better approximation of the Gaussian function because the network takes more information and learns better.

Minimizing the number of epochs and increasing learning rate η to 0.08, the convergence can be sped up slightly, without losing generalization.

Visually, the best model is the one which is less complex and generalises better. The model between hidden nodes 20-25 generalise well. Increasing the hidden nodes to more than 25 shows no significant change.

Amount of Training data(%)	Error(MSE)
80	0.004
60	0.006
40	0.008
20	0.01

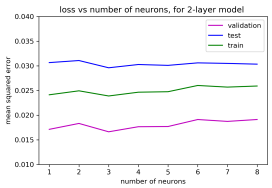
Tabell 3: Performance of 22 node model. η - 0.009

4 Results and discussion - Part II

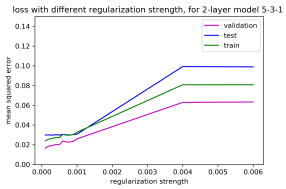
4.1 Two-layer perceptron for time series prediction - model selection, regularisation and validation

We split the data into three subsets. Training data: 900 samples, Validation data: 100 samples, Testing data: 200 samples.

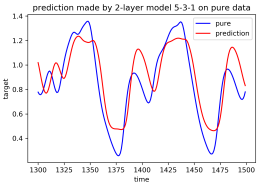
1)How does the number of neurons in the first hidden layer influence the validation performance? When the number of neurons in the first layer is 3, the lowest MSE on the validation data has been reached (Fig 10). So we choose it as three. The MSE on the test data set is always larger than MSE on train and on validation. Because the test data is unseen during training.



Figur 10: regularization strength = 0.0001



Figur 11: 2layer model 5-3-1



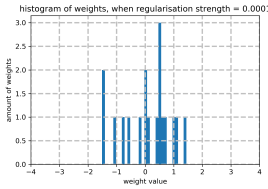
Figur 12: 2layer model 5-3-1, regularization strength = 0.0001

2)How does the regularization strength influence the validation performance? We choose the Lasso L1 norm as the regularization technique. Lasso L1 adds "absolute value of magnitude" of coefficient as penalty term to the loss function. We evaluate the MSE on validation data set, for regularization strength = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.004, 0.006]. Fig 11 shows that, the validation MSE increases when regularization strength increases. Because this model not complex. Small regularization strength is suitable in this case.

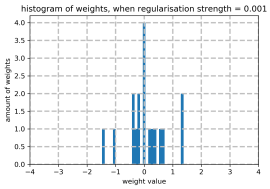
3)What is the effect of regularisation on the distribution of weights? We fix the architecture of the 2 layer model as 3-1. We change the regularization strength and get the distribution of weights (Fig 13,14,15). The larger the regularization strength is, the more amount of weights are penalized. So more weights are close to 0.

4.2 Comparison of two- and three-layer perceptron for noisy time series prediction

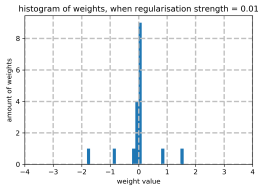
1) The visualization of predictions are shown in Fig 18,19,20. They are predicted by a 3-layer model 3-2-1 on the data which has different noise standard deviation (std = 0.03, 0.09, 0.18).



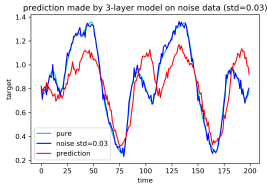
Figur 13: regularization strength=0.0001



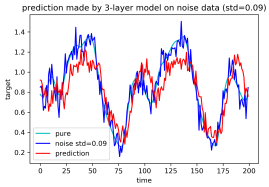
Figur 14: regularization strength=0.001



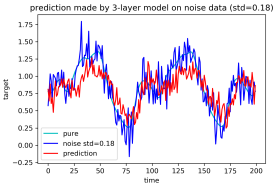
Figur 15: regularization strength=0.01



Figur 16: noise std=0.03



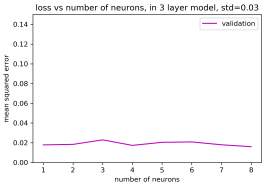
Figur 17: noise std=0.09



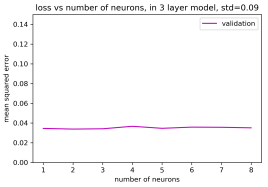
Figur 18: noise std=0.18

2)How does the number of neurons in the second hidden layer influence the validation performance? When the standard deviation of noise changes, the optimal number of neurons also changes (Fig 19,20,21). When the std deviation of noise is 0.09, 2 neurons provides the lowest MSE (Fig20). Therefore the optimal 3 layer model is 3-2-1.

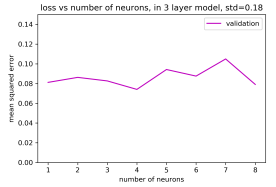
3)How does the noise influence the MSE? Large noise makes the MSE on the validation data set to be large.



Figur 19: noise std=0.03



Figur 20: noise std=0.09



Figur 21: noise std=0.18

4)We compare the 2 layer mode 3-1 with 3 layer mode 3-2-1, on the data set which has noise std deviation as 0.09. Each network has been trained for 10 times. We take the average of 10 outcomes. Because we did not run it for a large number of times, it is hard to draw a conclusion.

model	parameters	validation MSE	test MSE	computational time (sec)
2 layer 3-1	lr=0.001, regu=0.0001	0.0305	0.0417	6.5209
3 layer 3-2-1	lr=0.001, regu=0.0001	0.0291	0.0393	6.8346

Tabell 4: comparison between the best 2-layer model and 3-layer-model,lr=0.001, regu=0.0001