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

Jordan Wu, Daniel Janischowsky

September 23, 2018

1 Main objectives and scope of the assignment

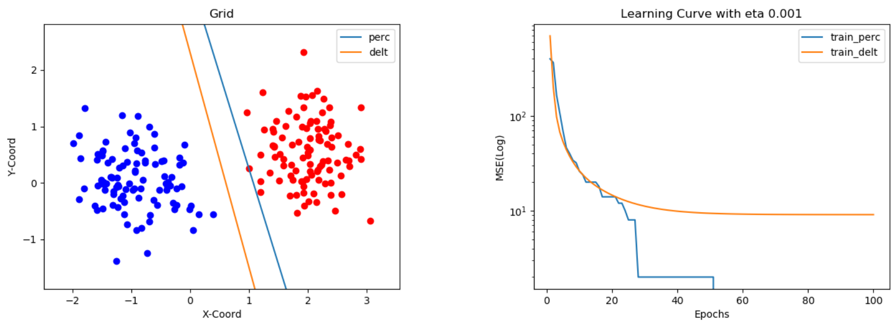
- Implement single layer networks with various learning schemes for classifying linearly separable data. Investigate how learning schemes affect performance, and the biasing term's role.
- Implement two-layer networks to perform classification of linearly non-separable data, encode data into lower-dimensional space, and function approximation. Assess network performance using a validation set, and investigate how two layer network's encode information such as the decision boundary.
- Implement multi-layer networks to perform time series prediction. Tune network using a hold-out validation set and various schemes of preventing overfitting and investigate their effects.

2 Methods

The entire lab was done in Python. For Part 1 of the assignment, we only used Numpy to construct the 1 and 2-layer perceptrons. For Part 2, we used scikit-learns's MLPRegressor, an implemented multi-layer perceptron for faster performance. We initially planned on using pytorch, but after reading the requirements of the lab, we found scikit's regressor to be just configurable enough.

3 Results and discussion - Part I

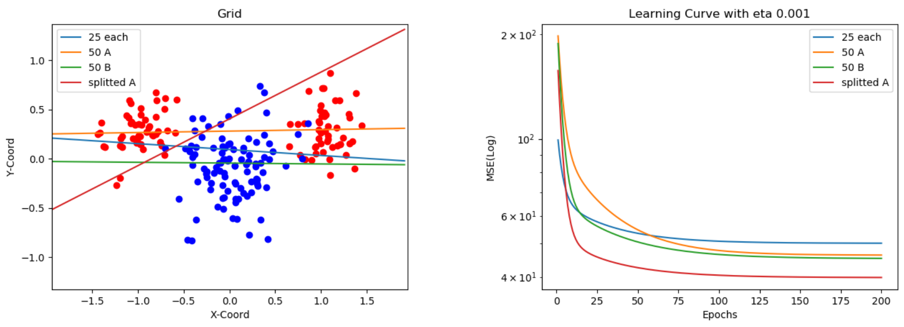
3.1 Classification with a single-layer perceptron



Figur 1: Perceptron vs. Delta Learning Rule

For all learning rules, we define convergence as $max(dW) < 10^{-6}$

Perceptron learning rule stops as soon as all the training data is classified correctly. The decision boundary will often lie very close and even on top of some points as seen in figure 1. Delta learning rule runs for longer, but the decision boundary is intuitively much "cleaner" and farther away from the data points after convergence as seen in figure 1. The sequential learning converges much faster in terms of epochs (25 vs 150) since it could use a larger learning rate. We didn't see any improvement in convergence speed with shuffling. Theoretically batch learning speed should not be influenced, but sequential learning speed should be improved slightly. Removing the biasing term prevents the decision boundary from translating, it can only rotate about the origin. The loss remains very high and not all points are classified correctly.

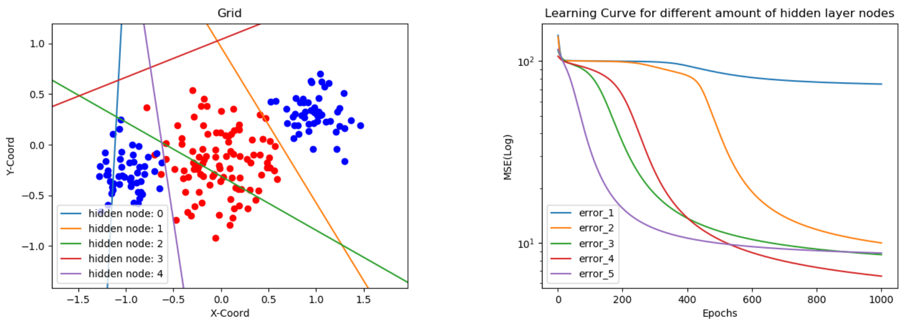


Figur 2: nonlinear separable data (perceptron)

"25 each" results in same error as without data removal. "splitted A" results best, due to 80% removal of $A(1,:) > 1$ (corresponds to right cloud)

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data



Figur 3: nonlinear separable data in two layer network

The data can be separated with a two layer network or more complex versions. Although the other hidden nodes will have weak weights towards the output node if we used more than two nodes in the hidden layer. In the left figure we can see a network with 5 hidden nodes, from which only number 1 and 4 have significant weights. The network could therefore be reduced to only two hidden nodes. The sequential learning approach can again converge for much bigger learning rates then batch learning. The error curve for the validation data set starts in both approaches to increase again at some point. After this point the network starts overfitting. Learning process should stop here.

3.2.2 The encoder problem

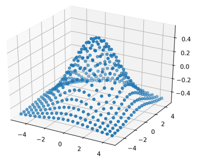
The network will always converge. The output of the hidden layer shows that the network encodes the input into ”binary” representation.

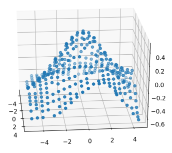
Input	Sign(Hidden)	Input	Sign(Hidden)
1 -1 -1 -1 -1 -1 -1 -1	-1. -1. -1.	-1 1 -1 -1 -1 -1 -1 -1	1. 1. 1.
-1 -1 1 -1 -1 -1 -1 -1	1. -1. 1.	-1 -1 -1 1 -1 -1 -1 -1	-1. 1. -1.
-1 -1 -1 -1 1 -1 -1 -1	1. -1. -1.	-1 -1 -1 -1 -1 1 -1 -1	-1. -1. 1.
-1 -1 -1 -1 -1 -1 1 -1	-1. 1. 1.	-1 -1 -1 -1 -1 -1 -1 1	1. 1. -1.

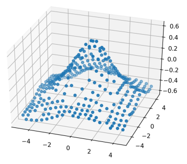
Figur 4: Encoding of Input to Binary

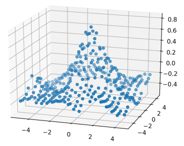
An 8-2-8 network will not converge, since it cannot encode all the variations of inputs. An auto encoder like this could be used for general data compression and dimension reduction.

3.2.3 Function approximation

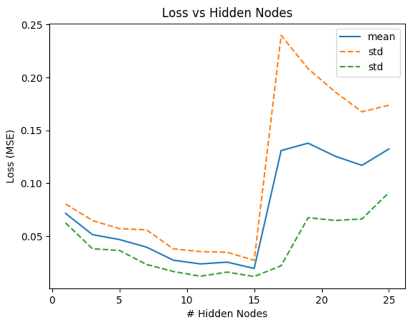

Target


5 Hidden Nodes ($\eta = 0.01$)

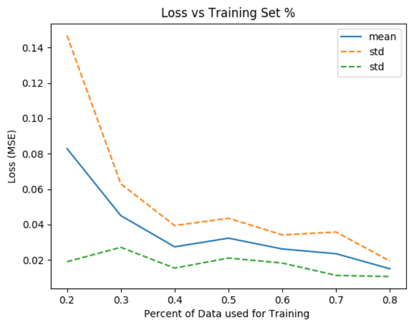

15 Hidden Nodes ($\eta = 0.001$)


20 Hidden Nodes ($\eta = 0.001$)

Figur 5: # Hidden Nodes vs Performance (1000 Epochs)



# Hidden vs Performance



Training Set Size vs Performance.

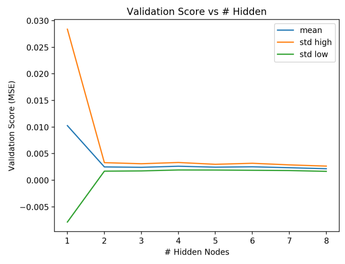
Figur 6: # Hidden Nodes vs Performance (1000 Epochs)

We can see that with 5 hidden nodes, we are underfitting the function, but with 20, we are overfitting, with excessive "ripples". In Fig 6, we can also see the sweet spot is at 15 hidden nodes, and that performance generally increases with increasing training size. A reasonable hypothesis here is that since our dataset is small, 400 samples, and we plenty of hidden nodes, any additional sample will help the network. For our best model, 15 hidden nodes, we couldn't increase the learning rate beyond 0.001 before diverging. However, momentum could be a solution in increasing convergence speed.

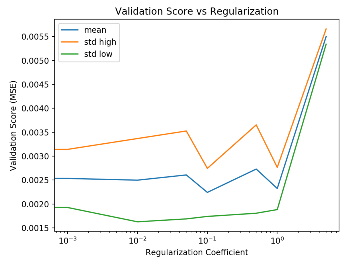
4 Results and discussion - Part II

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

Our implementation of early stopping is to stop if validation error does not decreased by 10^{-6} in 20 epochs. We can see in Fig. 7a that having more than 2 hidden nodes does not improve final performance. In our case, it only affected how fast it takes for the model to converge since we kept the same learning rate for all models. Fig. 7b shows how L2 regularisation (weight decay) affects validation error. The validation error improves until $\alpha = 0.1$, then starts getting worse. In Fig. 8 we see that as α value increases, weights are pushed towards 0.

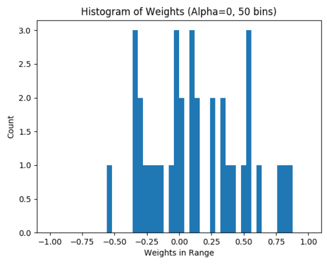


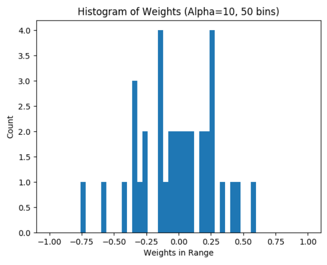
(a) # Hidden Nodes vs Performance (Early Stop)



(b) L2 Regularisation vs Performance

Figur 7: # Hidden Nodes vs performance

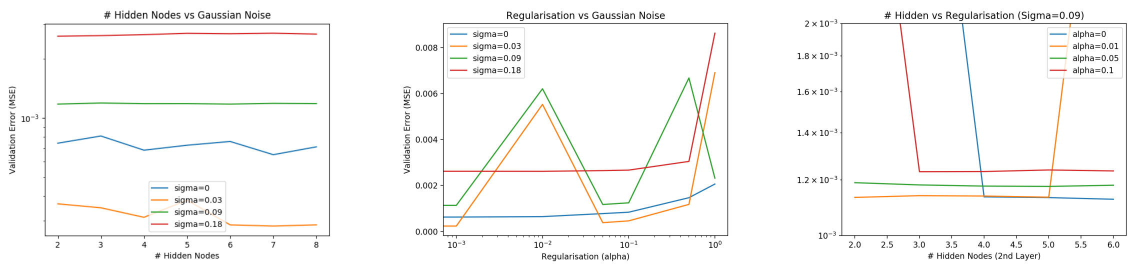




Figur 8: Histogram of Weights at various αs

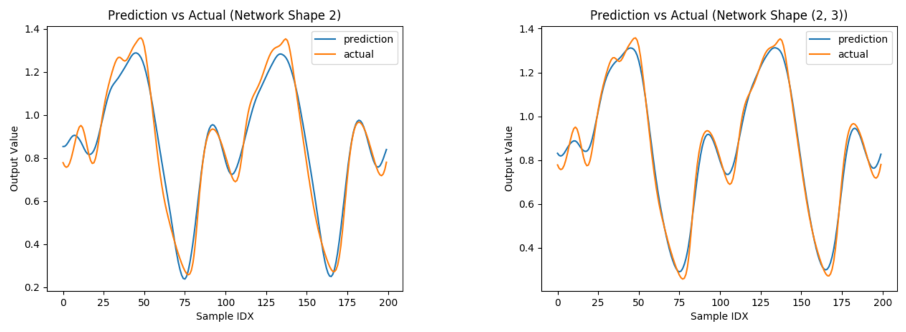
4.2 Comparison of two- and three-layer perceptron for noisy time series prediction

We can see $\sigma = 0.03$ give better results than no noise. Any higher noise gives worse results. With regularisation, $\sigma = 0.03$ also gives the best results. From both graphs we see high noise having some smoothing effect across the different levels of regularisation, but giving worse performance. A hypothesis for why this is occurring is that there is too much correction of overfitting, which is leading to underfitting. Finally, Fig. 10 shows the 3 layer network we've chosen takes about twice as long to finish training.



$\alpha = 0.05$ # hidden nodes = 4 α vs # Hidden, $\sigma = 0.09$

Figur 9: Noise vs Regularisation/#Hidden Nodes



2-Layer, 2 Hidden, $\alpha = 0.1$, $\sigma = 0.09$ 3-Layer, 2x3 Hidden, $\alpha = 0.01$, $\sigma = 0.09$

Figur 10: Prediction Vs Actual

Network Shape	α	σ	Test Loss Mean	Test Loss Std	Avg Train Time
(2,)	0.05	0	0.002818	0.000826	1.45088
(2,)	0.05	0.09	0.03119	0.04264	0.876391
(2,3)	0.01	0.09	0.001068	2.65009e-05	1.940052

Figur 11: Final Performance (10 Runs Each)

5 Final remarks

We found the lab to be very helpful in understanding the role various parameters and methods play in maximizing the performance of single and multi-layer perceptrons.