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

Robert Cedergren, Viktor Törnegren and Lingxi Xiong

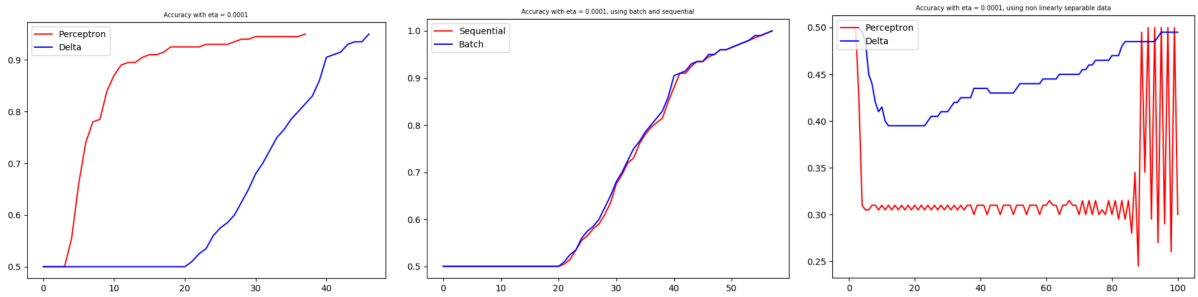
January 31, 2019

1 Methods

Python 3.6 is used to conduct the tasks and experiments in this lab. In part I, the basic Numpy library is used to implement the algorithm and the Matplotlib is used for visualization. In part II, tensorflow is the main library for building and training perceptron network along with those.

2 Results and discussion - Part I

2.1 Classification with a single-layer perceptron

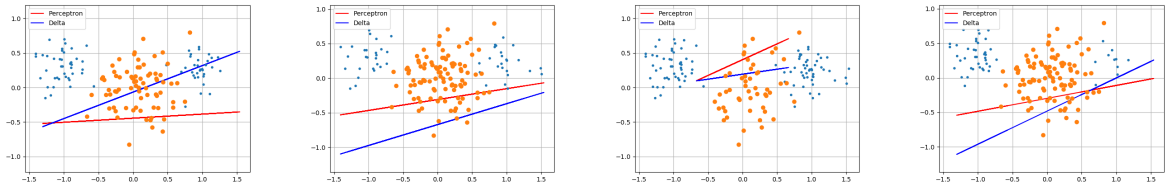


(a) Linearly separable data **(b)** Sequential and batch mode **(c)** Non linearly separable data

Figure 1: Comparison of accuracy

For linearly separable data: In Fig.1(a) we can see that the Delta rule converges much slower compared to the Perceptron rule. This is expected since we are setting the learning rate to 0.0001. In Fig.1(b), both batch and sequential mode seems to converge after the same number of epochs when setting the learning rate to 0.0001. Also, without bias the line will always pass through zero. With bias we can shift the line to the left and to the right, hence, with bias we can classify more complex data.

For non linearly separable data: In Fig.1(c) we can see that neither Delta nor perceptron can classify the non linearly separable data, they are both basically guessing. From Fig.2 and Table 1 we notice that the remove of 50% of class B in subsample3, which is the mainly non linear part of the data, improve both the performance and the localization of the decision boundary.



(a) Subsample 1 **(b)** Subsample 2 **(c)** Subsample 3 **(d)** Subsample 4

Figure 2: Decision boundaries of different subsamples

Table 1: Performance of different data subsampling manipulations

	sensitivity		**specificity**	
	Delta rule	Perceptron learning	Delta rule	Perceptron learning
Subsample1	0.640	0.253	0.573	0.933
Subsample2	0.180	0.500	1.000	0.870
Subsample3	0.950	0.900	0.280	0.160
Subsample4	0.780	0.720	0.950	0.850

2.2 Classification and regression with a two-layer perceptron

2.2.1 Classification of linearly non-separable data

In Table 2 we can see that both the training misclassification ratio and the validation misclassification ratio is decreasing until hidden nodes reaches 24 after that both of them are increasing. This seems reasonable since number of hidden nodes usually should be in between the number of input nodes and number of output nodes. It should be noted that we get similar result when testing different kinds of training and validation data. Moreover, after a certain amount of epochs the models is overfitted to the training data, hence, the misclassification ratio for the validation data will increase.

Nodes	1	2	4	8	12	16	20	24	30	40
train miss	0.246	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.246	0.253
val miss	0.260	0.040	0.040	0.040	0.040	0.040	0.040	0.040	0.260	0.280

Table 2: Misclassification ratio with 25% train and 50% validation data, batch mode with eta = 0.002 and epochs = 2000

Figure 3 shows two classes of data points (non-linearly separable) and the approximated decision boundary for a 2 layer perceptron where 0.0001, 0.9, 100000 and 20 were the learning rate, the momentum parameter, the number of epochs and the number of hidden nodes used, respectively. As can be seen, the data points can not be perfectly classified.

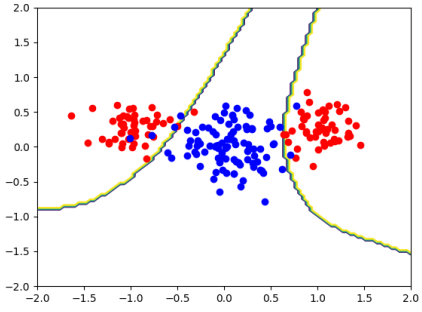


Figure 3: Plot of the approximated decision boundary, i.e. where the network output is 0, and the two classes.

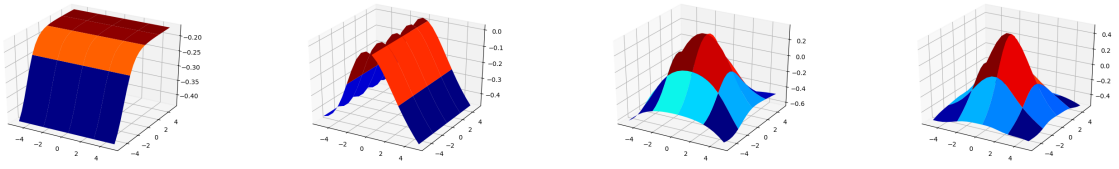
2.2.2 The encoder problem

Using a network with a hour-glass shaped topology, the network does not always converge and map inputs to themselves. It depends heavily on the learning rate, momentum parameter and of course the random initialized weights. However, for a learning rate of 0.5 and momentum parameter of 0.9 the network seems to converge most of the times within 5000 epochs. This is the case for a 8-3-8 architecture with or without bias as well as for a 8-2-8 architecture with bias. For a 8-2-8 architecture without bias the network does not converge.
The internal code represents the coordinates of the inputs in a lower dimensional space. The signs of the weighs (for the hidden layer and the output layer) are asymmetric, i.e. by transposing the weight matrix for the output matrix and comparing it to the weight matrix for the hidden layer the signs of the weights are equal.
Autoencoders serves a purpose for data compression as well as denoiseing images.

2.2.3 Function approximation

In this section 0.01, 0.9 and 1000 is used as learning rate, momentum and epochs, respectively.

The bell shaped Gaussian function $f(x, y) = e^{-(x^2+y^2)/10} - 0.5$ is approximated using a 2-layer perceptron network. This is done with different number of nodes in the hidden layers. The approximated functions for a given number of hidden nodes are visualized in Figure 4. It can be seen that 1 and 2 nodes are clearly insufficient to approximate the function underlying Gaussian. It can also be seen that the more hidden nodes the better the approximation. This is more clearly seen in Table 3 where the error decreases as the number of hidden nodes increases.



(a) Hidden nodes: 1 **(b)** Hidden nodes: 2 **(c)** Hidden nodes: 3 **(d)** Hidden nodes: 10

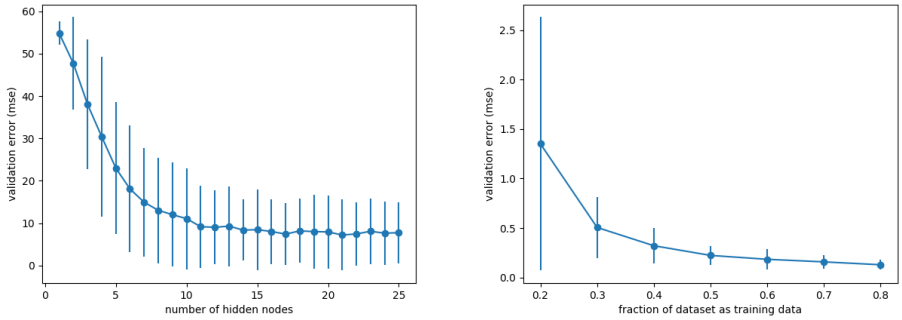
Figure 4: 3D plots of the approximated function

#Hidden nodes	1	2	3	4	10	20
Train error (μ)	48.62	31.01	3.79	2.21	0.18	0.01

Table 3: Training error for different number of hidden nodes using 2-layer perceptron network for approximating the underlying Gaussian. For every number of hidden node the training has been run 100 times using 1000 epochs.

Varying the number of hidden nodes in the hidden layer from 1 to 25, using a few example samples (20% of the entire data) as training data, it can be seen in Figure 5(a) that for very few hidden nodes (less than 5) the validation error and its standard deviation is high. For very many hidden nodes (more than 20) the validation error or the standard does not seem to change which means that one might as well choose a simpler model. It can be seen that the validation error and its standard deviation starts to settle around 14-17 number of hidden nodes. 17 hidden nodes provides the lowest mean and standard deviation.

For the best model, the 2 layer perceptron network with 17 hidden nodes, experiments are run with varying number of training samples, e.g. from 80% down to 20% of all the dataset. The results are shown in Figure 5(b) where it can be seen that the model generalizes better, i.e. the validation error and the corresponding standard deviation decreases, as the fraction of the data set used as training data increases from 0.2 to 0.8. The convergence can be accelerated by tuning the learning rate and the momentum parameter.



(a) **(b)**

Figure 5: (a) Validation error and its standard deviation for different number of hidden nodes in a 2 layer perceptron. For every number of hidden node the training and validation has been performed 300 times using 1000 epochs. (b) Validation error using 17 hidden nodes for different fractions of the dataset as training data. For every fraction the training and validation has been run 300 times using 1000 epochs.

3 Results and discussion - Part II

Note that all the result presented in the tables are with 30 times repetition. The learning rate is always 0.01.

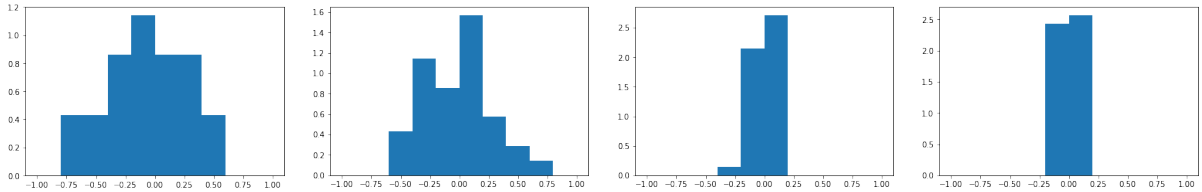
3.1 Two-layer perceptron for time series prediction - model selection, regularisation and validation

Table 4: Comparison of number of nodes on 2-layer perceptron (without regularization)

#nodes	Train loss(μ)	Train loss(S.D)	Val_loss(μ)	Val_loss(S.D)	Test err(μ)	Test err(S.D)
1	0.0635	0.0253	0.0678	0.0166	0.0724	0.0280
2	0.0609	0.0275	0.0779	0.0165	0.0695	0.0295
3	0.0462	0.0238	0.0811	0.0134	0.0527	0.0253
4	0.0469	0.0280	0.0816	0.0176	0.0520	0.0295
5	0.0511	0.0252	0.0779	0.0151	0.0569	0.0266
6	0.0481	0.0232	0.0797	0.0177	0.0554	0.0263
7	**0.0441**	**0.0199**	0.0812	0.0133	0.0504	0.0213
8	0.0458	0.0206	0.0791	**0.0139**	**0.0499**	**0.0206**

Table 5: Comparison of L2-regularization strength on 2-layer perceptron(#node=8)

λ	Train loss(μ)	Train loss(S.D)	Val_loss(μ)	Val_loss(S.D)	Test err(μ)	Test err(S.D)
0.001	0.4167	0.4184	0.4384	0.4119	0.3953	0.3900
0.01	0.4269	0.4085	0.4526	0.3982	0.3915	0.3934
0.1	**0.0298**	**0.0143**	0.0956	0.0071	**0.0239**	0.0139
1.0	0.0678	0.0048	0.0717	0.0046	0.0629	0.0079
10.0	0.0842	0.0007	0.0575	0.0016	0.0930	0.0019
100.0	0.0864	0.0002	0.0552	0.0006	0.0974	**0.0003**



(a) λ=0.001 **(b)** λ=0.01 **(c)** λ=0.1 **(d)** λ=1.0

Figure 6: Histogram of weights for different regularization strengths

From Table 4 we can see that within limit, the increment of the number of nodes in hidden layers leads to a better prediction result, i.e. generalization performance. From Table 5 we can see that using the best number of node for a 2-layer perceptron, a proper regularization(around 0.1) can improve the model performance. And in Figure 4, the distribution of weights in the hidden layer shrinks with larger regularization strength, i.e. lager λ value.

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

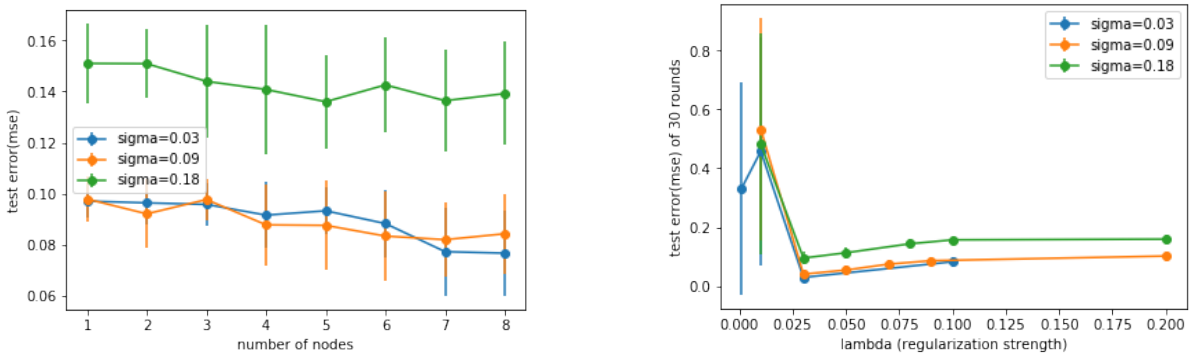
Figure 7 is a visualization of the variation of test error(mse) of the trained model with different configurations with respect to number of nodes in the hidden layer and strength of regularization from Table 6 and Table 7. In general, the generalization of models becomes worse with the increase of noise in original data. And with different levels of noise, the best network configuration varies.

Table 6: Comparison of number of nodes on 3-layer perceptron(#node in hidden layer 1=8, without regularization)

Noise	#node	train_loss	train_loss(S.D)	valid_loss	valid_loss(S.D)	test_err	test_err(S.D)
0.03	1	0.0867	0.0005	0.0538	0.0057	0.0971	0.0064
	2	0.0866	0.0012	0.0544	0.0075	0.0964	0.0086
	3	0.0867	0.0009	0.0550	0.0077	0.0958	0.0086
	4	0.0862	0.0019	0.0587	0.0113	0.0916	0.0130
	5	0.0867	0.0008	0.0574	0.0083	0.0933	0.0093
	6	0.0766	0.0118	0.0556	0.0026	0.0883	0.0134
	7	0.0667	0.0158	0.0588	0.0057	0.0773	0.0173
	8	0.0667	0.0149	0.0592	0.0062	**0.0767**	0.0165
0.09	1	0.0898	0.0081	0.0613	0.0034	0.0979	0.0089
	2	0.0844	0.0128	0.0632	0.0054	0.0921	0.0135
	3	0.0890	0.0076	0.0620	0.0023	0.0977	0.0083
	4	0.0797	0.0156	0.0656	0.0077	0.0878	0.0159
	5	0.0805	0.0171	0.0658	0.0062	0.0876	0.0176
	6	0.0751	0.0163	0.0677	0.0072	0.0834	0.0174
	7	0.0746	0.0143	0.0676	0.0083	**0.0820**	**0.0144**
	8	0.0764	0.0149	0.0673	0.0066	0.0843	0.0155
0.18	1	0.1081	0.0088	0.0787	0.0067	0.1510	0.0155
	2	0.1076	0.0080	0.0788	0.0049	0.1509	0.0136
	3	0.1038	0.0143	0.0817	0.0059	0.1440	0.0221
	4	0.1021	0.0153	0.0844	0.0104	0.1408	0.0254
	5	0.0985	0.0115	0.0844	0.0074	**0.1360**	**0.0185**
	6	0.1034	0.0115	0.0833	0.0056	0.1426	0.0188
	7	0.0993	0.0120	0.0840	0.0071	0.1364	0.0198
	8	0.1009	0.0122	0.0832	0.0082	0.1392	0.0202

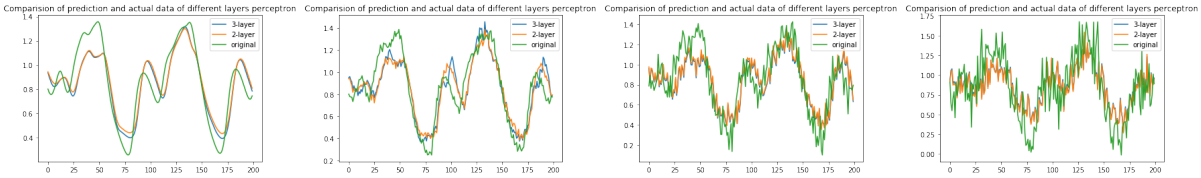
Table 7: Comparison of strength of regularization on 3-layer perceptron (#node in hidden layer 1=8)

Noise & #node	λ	train_loss	train_loss(S.D)	valid_loss	valid_loss(S.D)	test_err	test_err(S.D)
0.03 & 8	0.001	0.3474	0.3878	0.3486	0.3988	0.3320	0.3620
	0.01	0.5124	0.3808	0.5196	0.3904	0.4577	0.3860
	0.03	0.0436	0.0161	0.1013	0.0120	**0.0289**	0.0186
	0.1	0.0849	0.0033	0.0668	0.0148	0.0823	0.0171
	1.0	0.0872	0.0001	0.0525	0.0003	0.0987	0.0002
0.09 & 7	0.01	0.5895	0.3761	0.6035	0.3799	0.5327	0.3769
	0.03	0.0530	0.0113	0.1073	0.0050	**0.0401**	0.0092
	0.05	0.0766	0.0091	0.1020	0.0118	0.0540	0.0143
	0.07	0.0885	0.0052	0.0858	0.0147	0.0741	0.0163
	0.09	0.0919	0.0026	0.0751	0.0125	0.0858	0.0135
0.18 & 5	0.01	0.5034	0.3827	0.5107	0.3960	0.4815	0.3751
	0.03	0.0900	0.0112	0.1108	0.0101	**0.0948**	0.0208
	0.05	0.1042	0.0067	0.1041	0.0104	0.1127	0.0187
	0.08	0.1115	0.0020	0.0862	0.0109	0.1433	0.0172
	0.3	0.1130	0.0002	0.0763	0.0003	0.1591	0.0002



(a) variation of number of nodes **(b)** variation of regularization strength

Figure 7: Model generalization performance



(a) without noise **(b)** σ=0.03 **(c)** σ=0.09 **(d)** σ=0.18

Figure 8: Prediction comparison

The result of predictions from 2 and 3 layer perceptron and actual data are shown in Figure 8. The configuration of 2-layer perceptron is 8 nodes with 0.1 regularization strength and the model parameters of 3-layer perceptron keep the same with that in Table 7. With early stopping, we can see from Table 8 that in both 2 and 3 layers perceptron, the computation time decrease with the hidden nodes increase, then slightly increase again because of the increase of model complexity.

Table 8: Training time cost of varying number of hidden nodes in multi-layer perceptron

#layers	1	2	3	4	5	6	7	8
3	3.042	1.319	2.975	1.188	0.556	1.239	0.593	0.943
2	2.285	4.072	3.514	2.708	0.547	0.323	0.761	0.634