KTH Royal Institute of Technology

DD2437 Artificial Neural Networks and Deep Architectures

Lab 1

Alexandros Ferles *Email: ferles@kth.se*

Pantelis Myriokefalitakis *Email: pmy@kth.se*

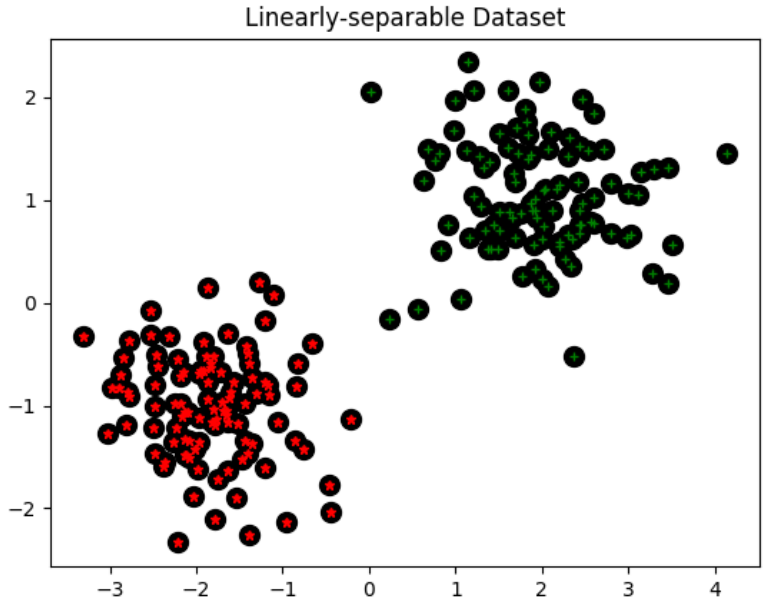
George Zervakis *Email: zervakis@kth.se*

February 5, 2018

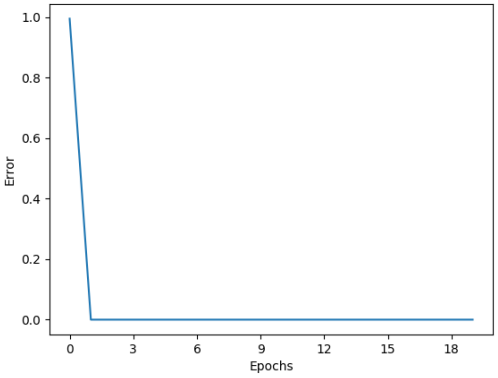
3 Assignment - Part I

3.1 Classification with a single-layer perceptron

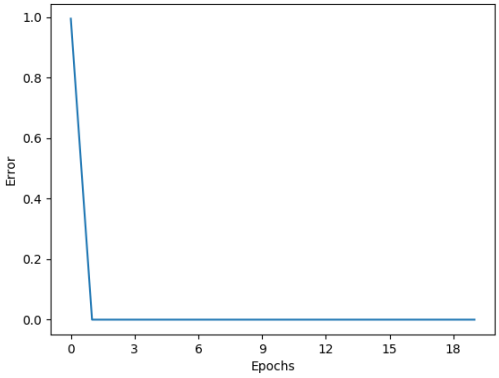
3.1.1 Generate linearly-separable data



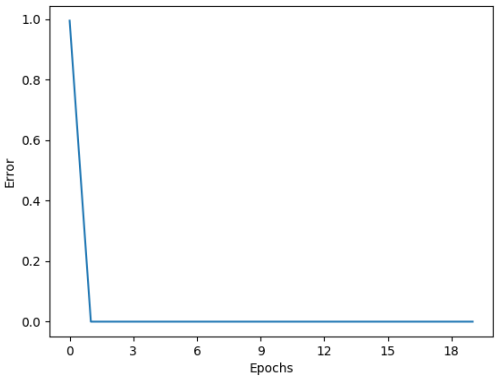
3.1.2 Perform classification with a single-layer perceptron and analyse the results



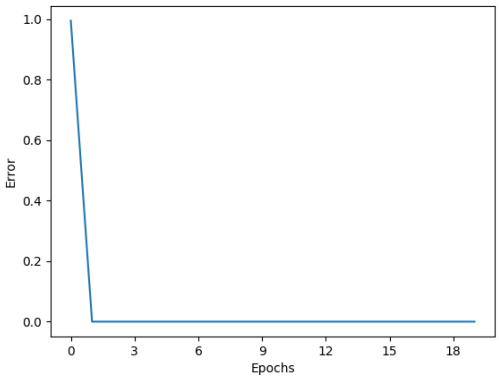
Perceptron (sequential)



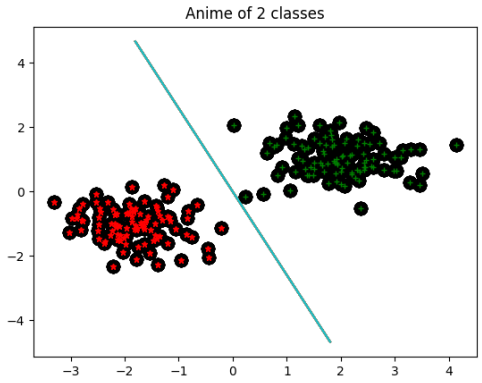
Perceptron (batch)



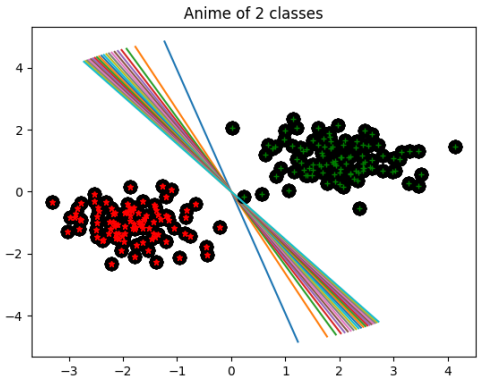
Delta Rule (sequential)



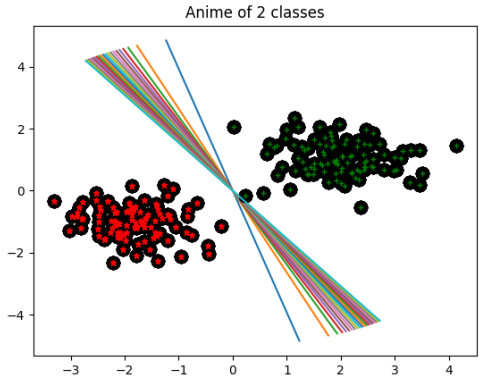
Delta Rule (batch)



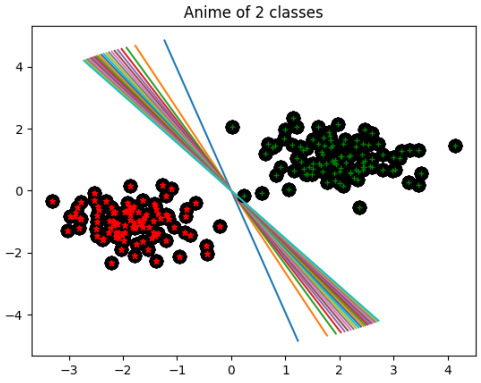
Perceptron Rule (batch)



Delta Rule (batch)



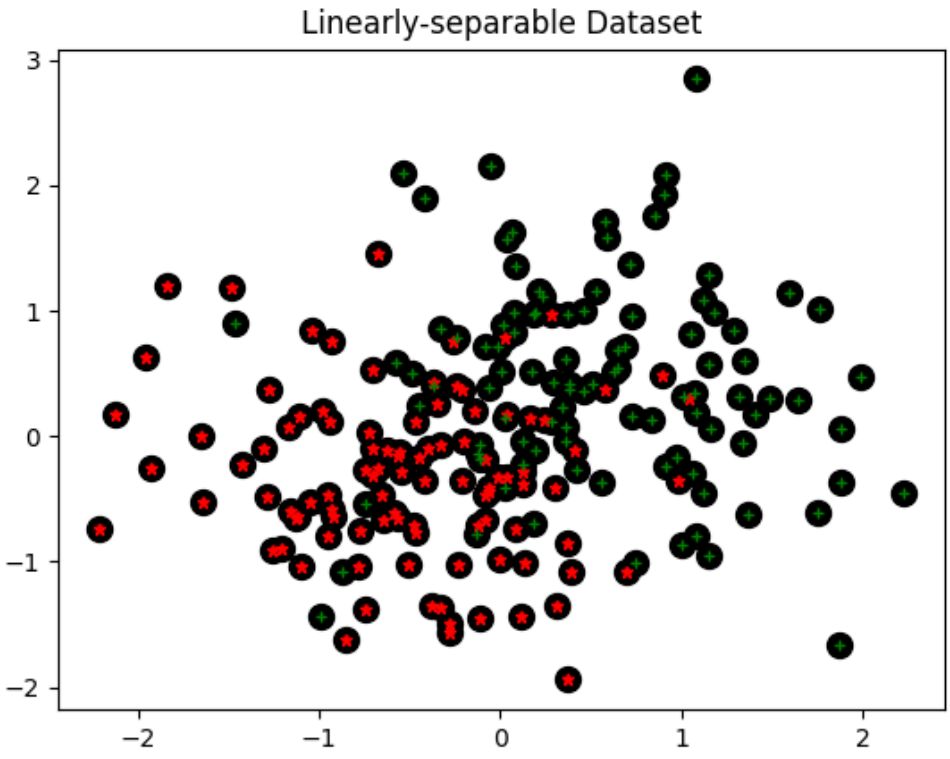
Perceptron Rule (sequential)

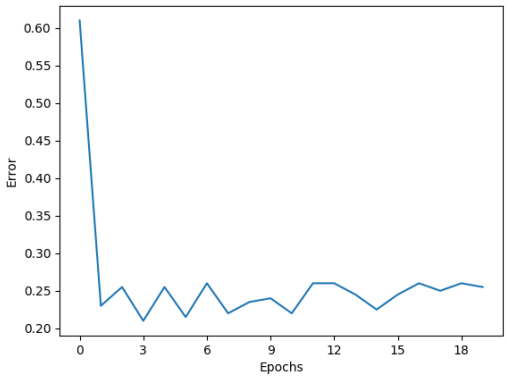


Delta Rule (sequential)

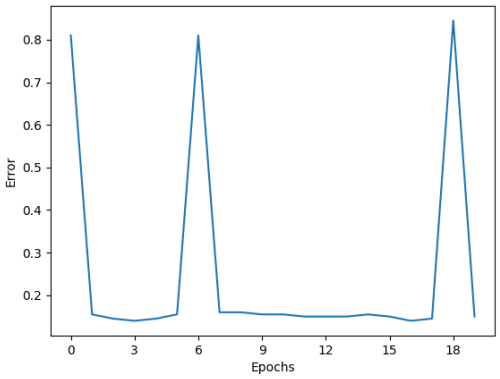
Classification with a single-layer perceptron	Perceptron rule (sequential)	Delta rule (sequential)	Perceptron rule (batch)	Delta rule (batch)
Number of epochs	1.5	1.5	1.5	1.5
Learning rate(η)	0.0005	0.0005	0.0005	0.0005
Error ratio	*0.00%*	*0.00%*	*0.00%*	*0.00%*
Misclassified samples	0/200	0/200	0/200	0/200

3.1.3 Classification of samples that are not linearly separable

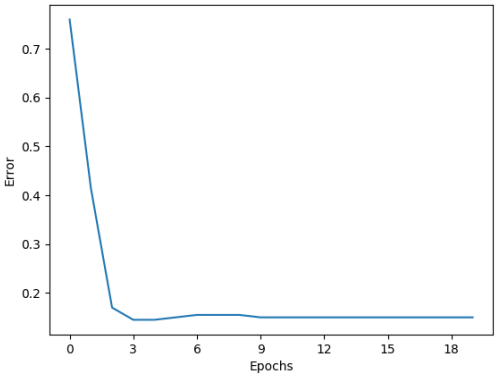




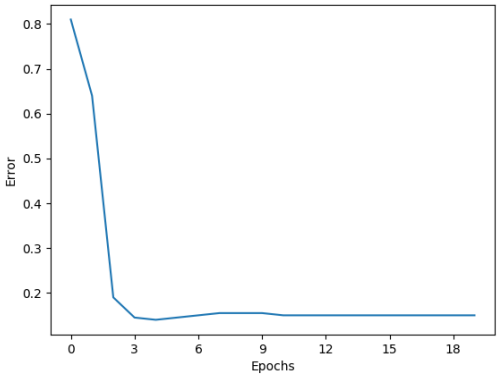
Perceptron (sequential)



Perceptron (batch)

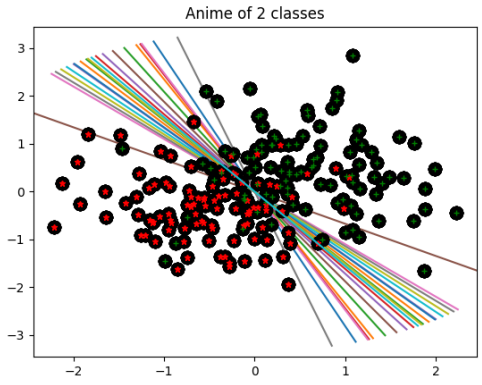


Delta Rule (sequential)

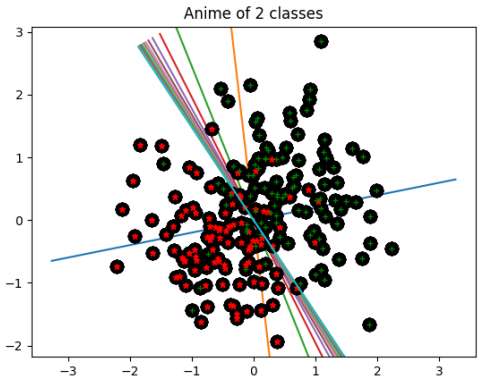


Delta Rule (batch)

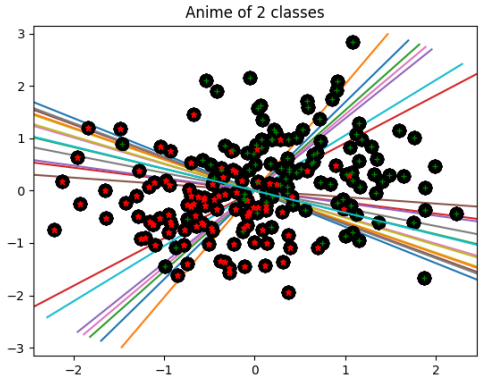
Classification with a single-layer perceptron	Perceptron rule (sequential)	Delta rule (sequential)	Perceptron rule (batch)	Delta rule (batch)
Number of epochs	exceeded	15	exceeded	15
Learning rate(η)	0.0005	0.0005	0.0005	0.0005
Error ratio	*0.21%*	*0.14 %*	*0.14 %*	*0.14%*
Misclassified samples	42/200	29/200	28/200	28/200



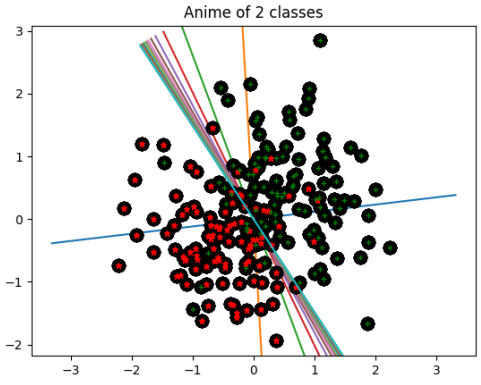
Perceptron Rule (batch)



Delta Rule (batch)



Perceptron Rule (sequential)



Delta Rule (sequential)

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

- **Test a two-layer perceptron trained with backprop and verify that it can solve the problem (separate the two classes).**

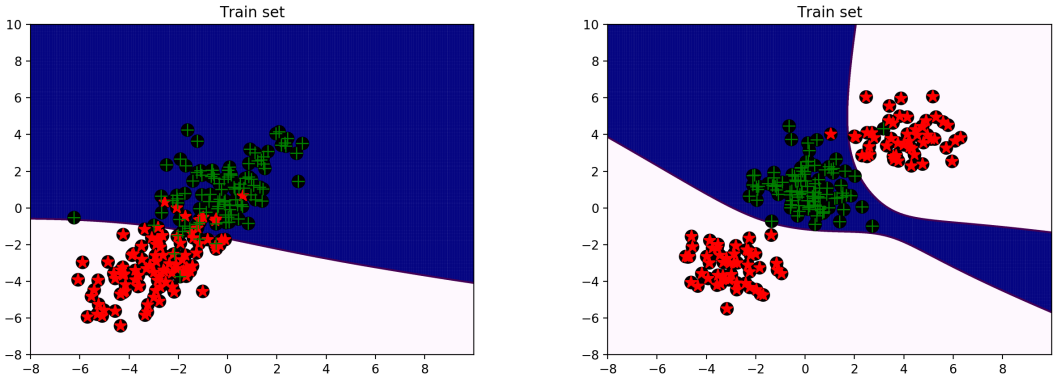


Figure 11: Random non-linear separable datasets. We can notice a fair classification on the training data.

- **Modify the number of hidden nodes and demonstrate the effect the size of the hidden layer has on the performance**

  The following results were derived with $\eta = 0.001$ a number of 100 epochs.

# Hidden nodes	# Missclassified data	MSE8
2	**18**/200	0.17%
3	**18**/200	0.18%
4	**20**/200	0.17%
5	**16**/200	0.16%
6	**17**/200	0.16%
7	**16**/200	0.16%
8	**17**/200	0.16%

- **How many hidden nodes do you need to perfectly separate the available data (if manageable at all given your data randomisation)?**

  The best solution, given the randomisation of our data, was obtained from a two-layer perceptron with 5 and 7 nodes.

- **How do the training and test learning curves compare?**

  We can see from the diagram, that both curves have the same behaviour, that is, the error is decreasing after each epoch. However, the test error is slightly lower that the training and this is reasonable, since i) the test data are less than the training, and ii) the samples for the test set, are drawn from the same gaussian, thus not all samples are "unseen" on the test set.

- **How do the training and test classification results depend on the size of the hidden layer?**

Again, we run the simulations for $\eta = 0.001$ and a number of 100 epochs.

# Hidden nodes	# Missclassified data (Tr.)	MSE (Tr.)	# Missclassified data (Val.)	MSE (Val.)
2	**14**/200	0.14%	**8**/100	0.07%
3	**13**/200	0.13%	**7**/100	0.05%
4	**14**/200	0.13%	**8**/100	0.12 %
5	**13**/200	0.13%	**7**/100	0.10%
6	**14**/200	0.13%	**8**/100	0.12 %
7	**14**/200	0.13%	**8**/100	0.12%
8	**13**/200	0.13%	**8**/100	0.10%

- **How many epoch iterations do you need for convergence?**

  We choose the 'optimal' number of hidden nodes (5) in order to locate the number of epoch that is required for convergence.

From the above diagram we estimate this number to be around 40 epochs.

- **Is there any difference between batch and sequential learning approaches?**

  Generally, we observed that the computational time is greater for the sequential learning approach.

3.2.2 The encoder problem

Your task is to study what type of representation is created in the hidden layer. Please, use your implementation of the generalised Delta rule to train the network until the learning converges (it does not necessarily have to imply that the mean squared error is 0 but that the rounded outputs match the corresponding inputs).

- **Does the network always succeed in doing this?**

  The auto-encoder defined by our neural network is usually successful in serving as the identity function of the input data. This can be verified from the following diagram.

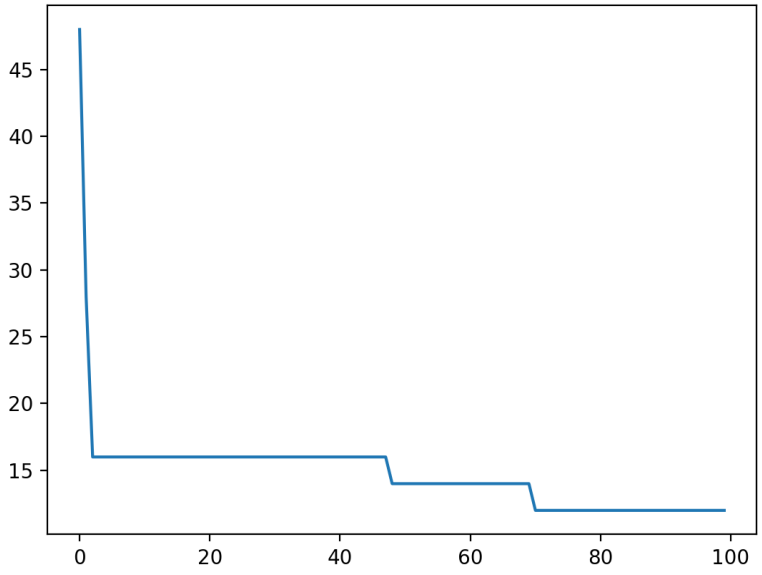


Figure 12: Encoder result for a randomised input

  However, we have to note that this is not a typical auto-encoder behaviour. On the contrary, we expect to observe a greater error as the input data of the validation set become more dissimilar to the training data[**?**]. In our case, we cannot observe this difference because our validation data, even though randomised, are pretty similar to the training data.

- **How does the internal code look, what does it represent?**

  The internal code represents two stages:

  1. The *encoding* stage, where the input is compressed to fit a lower dimensionality than its original one, thus merging those input features that are correlated to each other.

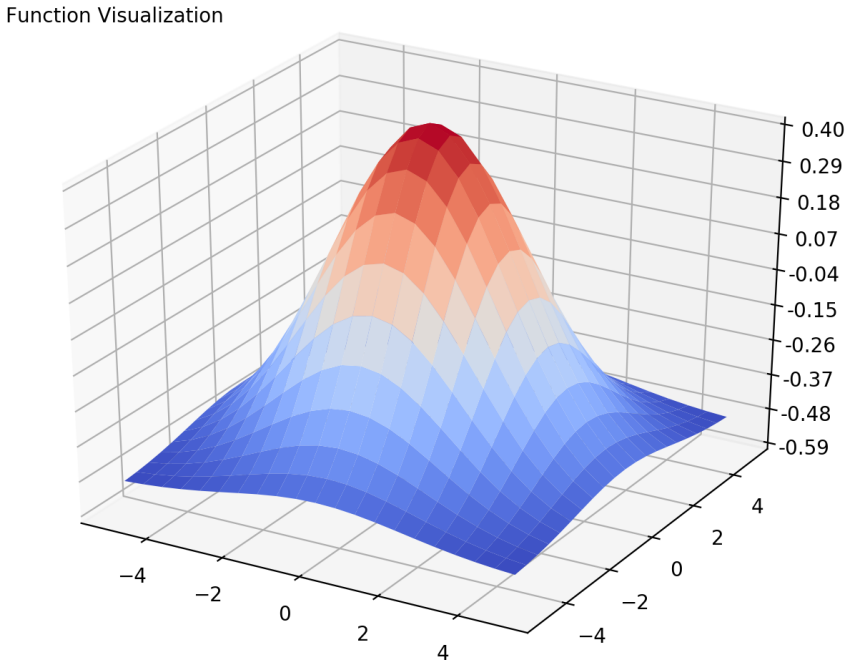
2. The *decoding* stage, where the compressed information containing the important features of the input is decompressed to recreate it in its original form in the output.

- **You could also examine the weight matrix for the first layer. Can you deduce anything from the sign of the weights?**

  The sign of the weights on the first (hidden) layer is recalculated through the training process in order to assign the corresponding importance to the features of the input. A positive weight value means using some features as "important" in order to create the output. Similarly, a negative weight value means that there are features of the input that are not taken into account when trying to recreate the input.

3.3 Function approximation

3.3.1 Generate function data

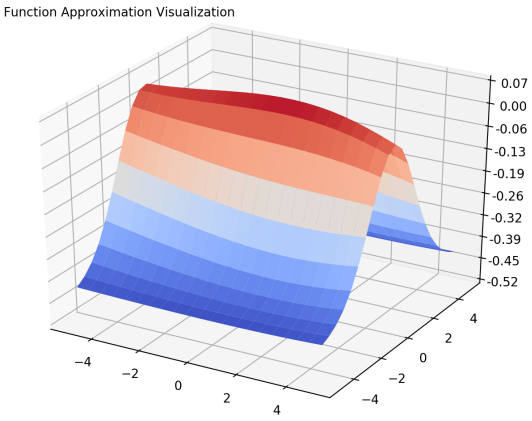


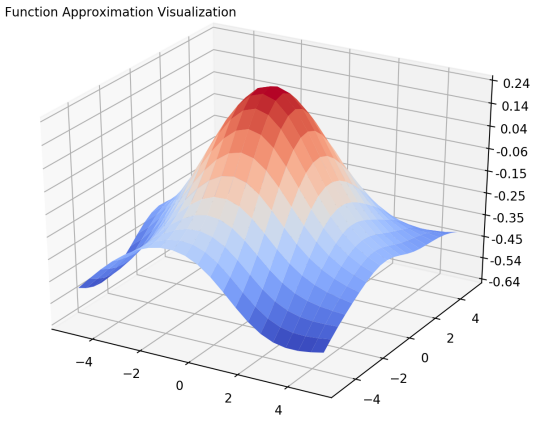
Visualization of the output function

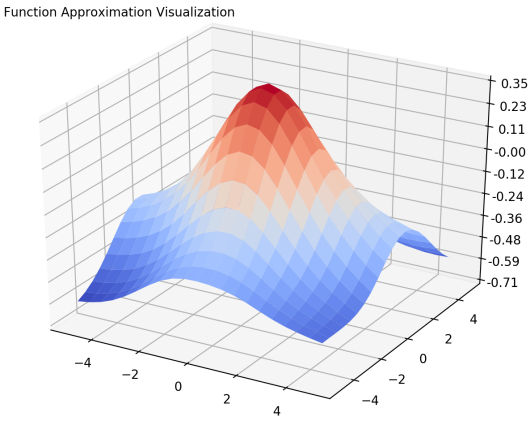
3.3.2 Train the network and visualise the approximated function

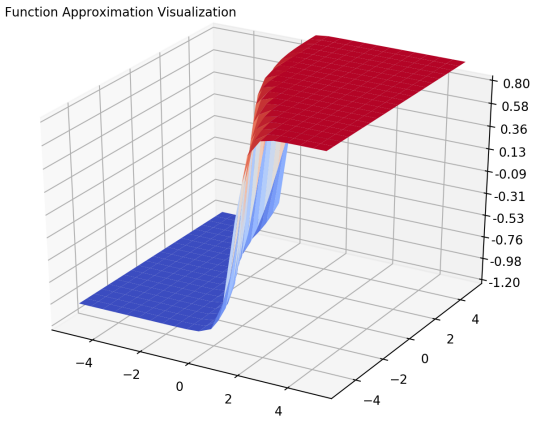
- **Experiment with different number of nodes in the hidden layer to get a feeling for how this parameter affects the final representation**.

We trained our network with different number of nodes in the hidden layer and we observed that the results where satisfying with the use of 10-60 nodes, while any values more or less than that, resulted in a bad approximation.









3.3.3 Evaluate generalisation performance

- **Test with n = 1 up to n = 25. Vary the number of nodes in the hidden layer and try to observe any trends.**
- **What happens when you have very few (less than 5) or very many (more than 20) hidden nodes? Can you explain your observations?**

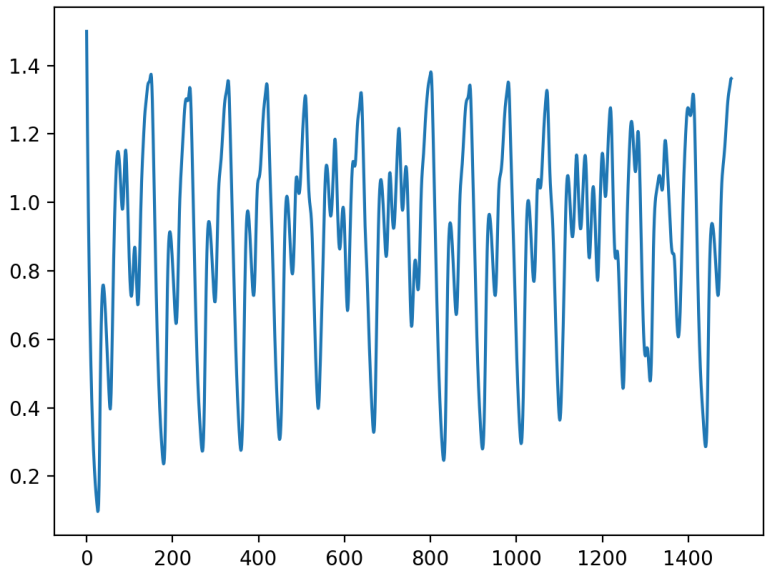
  We observed that for very few nodes, e.g. < 5 the network was not able to capture a lot of information, hence it couldn't produce a good approximation of the Gaussian. As we increase

the number of input patterns, e.g. $10 - 18$, the model starts to approximate the bell shape, since now we have enough inputs from every region to capture all the information needed to approximate the function. However, as the number of patterns gets higher and higher $>> 20$, the approximation gets too complex because the system takes into account more information that it actually requires to well-represent the gaussian.

4 Assignment - Part II

4.1 Data

- **Data generation and plotting the resulting time series.**



Mackey-Glass series representation

4.2 Network configuration

- **Training a neural network for different configurations (e.g., the number of hidden nodes, strength of regularisation etc.) with the use of early stopping and a regularisation technique of your own choice.**

4.3 Simulations and evaluation

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

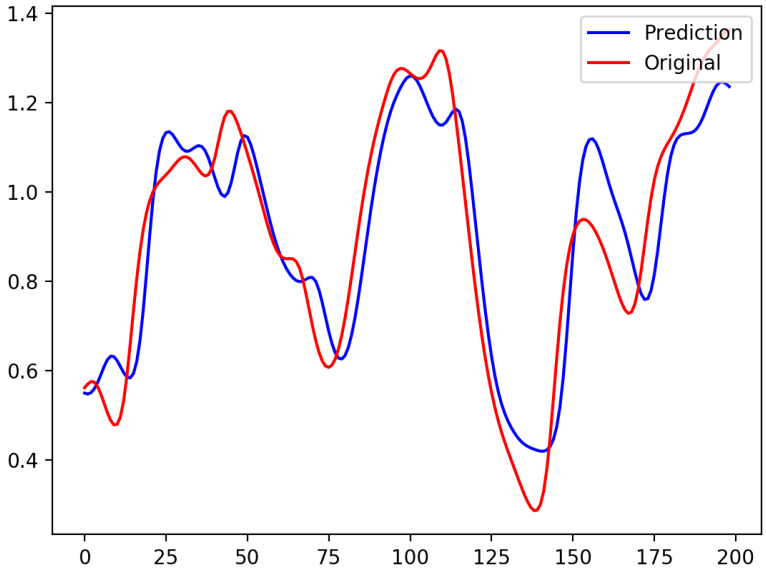
Use the available data to design and evaluate a two-layer perceptron network for Mackey-Glass time series prediction, which involves:

- **Validation of different network configurations and estimation of the generalisation error on a hold-out validation set, comparing different models and selecting one for further evaluation. What is the effect of regularisation strength and the number of hidden nodes on the validation performance?**

# Hidden nodes	# epochs	# Regularisation strength	e_{train} %	e_{val} %
2	1052 / 1068	0.1 / 0.0001	0.01	0.10
5	815 / 798	0.1 / 0.0001	0.01	0.10
8	691 / 441	0.1 / 0.0001	0.01	0.10

- **Final evaluation of the selected model on a test set - the conclusive estimate of the generalisation error on the unseen data subset. Plotting these test predictions along with the known target values (and/or plotting the difference between the predictions made by your multi-layer perceptron and the corresponding true time series samples).**

A diagram of a the selected model: $\eta = 0.0001, nodes = 5, \beta = 0.0001$, is given above.



Test error = 0.11%

4.3.2 Three-layer perceptron for noisy time series prediction - generalisation

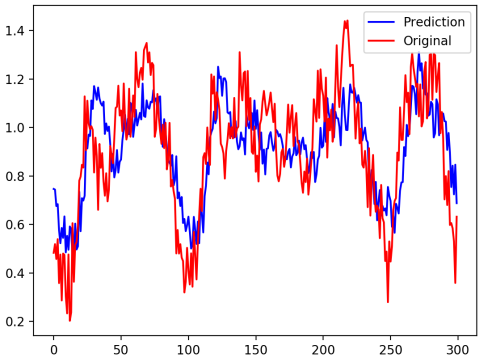
- **Examine how the validation prediction performance (estimated on a hold- out set) depends on the number of nodes in the second hidden layer for different amount of noise (experiment with three values of the std dev of the additive Gaussian noise, σ = 0.03, 0.09 and 0.18). Are there any strong trends or particular observations to report?**

# Hidden nodes	# ∼epochs	σ	e_{train} %	e_{val} %
5 + 2	2441	0.03	0.015	0.13
5 + 5	1870	0.03	0.015	0.13
5 + 8	1676	0.03	0.015	0.13
5 + 2	4000	0.09	0.032	0.17
5 + 5	4000	0.09	0.028	0.16
5 + 8	4000	0.09	0.029	0.18
5 + 2	4000	0.18	0.068	0.25
5 + 5	3000	0.18	0.066	0.25
5 + 8	2000	0.18	0.07	0.25

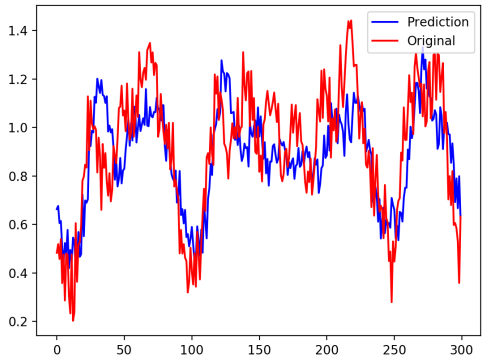
- **What is the effect of regularisation? How does the regularisation parameter**

interact with the amount of noise (as the amount of noise increases, are there any incentives to change the regularisation paremeter)?

The regularisation, functions like a penalty term for the weights. We specifically selected our shrinkage penalty to be the $\mathcal{L}_2$ norm of the weight vector. As we increase the regularisation parameter β, we allow more and more weights to impact the input patterns (less penalty), thus we increase the bias of our model. Therefore, small values of b, lead to forcing the weights towards zero, thus increase of variance. By adding Gaussian noise to all of our data, we want to establish a "balance" between bias-variance trade-off in order for our model to generalise well on unseen data. According to the amount of noise σ, we want to adjust the parameter β (along with the other parameters that affect the complexity), to achieve good level of bias-variance for our model, e.g. high $\sigma \Rightarrow$ high β.

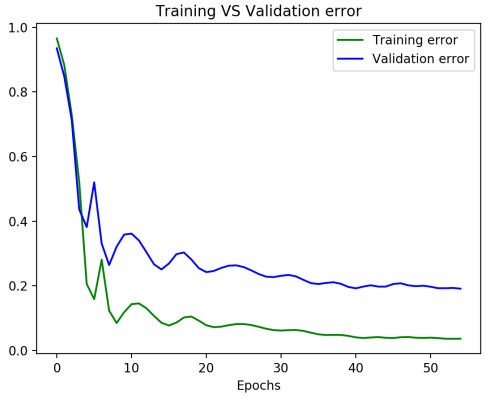


$beta = 0.0001, \sigma = 0.09, \#epochs = 51$



$beta = 0.1, \sigma = 0.09, \#epochs = 55$

$beta = 0.0001, \sigma = 0.09, \#epochs = 51$



$beta = 0.1, \sigma = 0.09, \#epochs = 55$

- **For each configuration of noise choose the best three-layer model. Then compare**

the selected models with the two-layer network in the first task (trained here from scratch on the corresponding noisy data) in terms of generalisation error estimated on the evaluation test set. Is any of the two networks superior irrespective of the amount of noise? Discuss the effect of noise on the generalisation performance.

# Hidden nodes 2-layer	# Hidden nodes 3-layer	σ	e_{2gen} %	e_{3gen} %
5	5 + 5	0.03	0.14	0.195
5	5 + 5	0.09	0.18	0.17
5	5 + 5	0.18	0.26	0.26

- **What is the computation cost (time) of backprop learning involved in scaling the network size (from two- to three-layer perceptron and for three- layer perceptrons with varying number of hidden nodes)?**

# Hidden nodes 2-layer	# Hidden nodes 3-layer	σ	$cost(time)$	$cost(time)$
5	5 + 5	0.03	1.6sec	1.73sec
5	5 + 5	0.09	1.64sec	1.72sec
5	5 + 5	0.18	1.73sec	1.76sec
5	5 + 5	0.03	1.60sec	1.73sec
10	10 + 10	0.03	1.68 sec	1.80sec
10	10 + 10	0.09	1.65sec	1.78sec
10	10 + 10	0.18	1.69sec	1.76sec