Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1b

Learning with backpropagation and generalisation in multi-layer perceptrons

Tristan Perrot, Mathis Pernin, Romain Darous

January 31, 2022

Please be aware of the constraints for this document. The main intention here is that you learn how to select and organise the most relevant information into a concise and coherent report. The upper limit for the number of pages is 8 with fonts and margins comparable to those in this template and no appendices are allowed.
These short reports should be submitted to Canvas by the authors as a team before the lab presentation is made. To claim bonus points the authors should uploaded their short report a day before the bonus point deadline. The report can serve as a support for your lab presentation, though you may put emphasis on different aspects in your oral demonstration in the lab. Below you find some extra instructions in italics. Please remove them and use normal font for your text.

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 configure and monitor the behaviour of learning with the *error backpropagation* algorithm for MLP networks
- to recognise risks associated with backpropagation and minimise them for robust learning of MLPs.

2 Methods

We used the programming language **Python** and the programming environment **Jupyter Notebook**. We used the python library **NumPy** and **PyTorch** to perform the matrix computations and generate data. Concerning the display of the plots such as dataset visualization and learning curves, we used the libraries **Matplotlib** and **tqdm**. This allowed us to have a dynamic display of the algorithms, with a progress bar to observe the convergence of the separating line step by step.
For the first part, we created a Python object to describe as precise as possible the model and easily reuse the architecture with some parameters given in argument to test it.
Concerning the second part, we used **tensorflow** and **keras** python libraries to build the neural network.

3 Results and discussion

Make effort to be ***concise and to the point*** *in your story of what you have done, what you have observed and demonstrated, and in your responses to specific questions in the assignment. You should skip less important details and explanations. In addition, you are requested to add a* ***discussion*** *about your interpretations/predictions or other thoughts concerned with specific tasks in the assignment. This can boil down to just a few bullet points or a couple of sentences for each section of your results. Overall, structure each Results section as you like, e.g. in points. Analogously, feel free to group and combine answers to the questions even between different experiments if it makes your story easier to convey.*

Plan your sections and consider making combined figures with subplots rather than a set of separate figures. ***Figures*** *have to condense information, e.g. there is no point showing a separate plot for generated data and then for a decision boundary, this information can be contained in a single plot. Always carefully describe the axes, legends and add meaningful captions. Keep in mind that figures serve as a support for your description of the key findings (it is like storytelling but in technical format and academic style.*

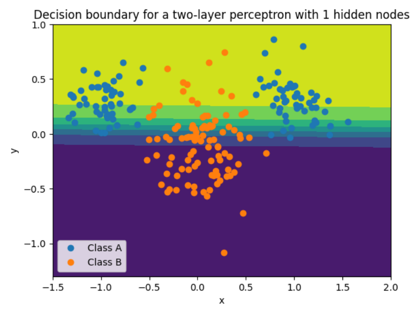
Similarly, use ***tables*** *to group relevant results for easier communication but focus on key aspects, do not overdo it. All figures and tables attached in your report must be accompanied by captions and referred to in the text, e.g. "in Fig.X or Table Y one can see ....".*

When you report quantities such as errors or other performance measures, round numbers to a reasonable number of decimal digits (usually 2 or 3 max). Apart from the estimated mean values, obtained as a result *of averaging over multiple simulations, always include also* ***the second moment****, e.g. standard deviation (S.D.). The same applies to some selected plots where* ***error bars*** *would provide valuable information, especially where conclusive comparisons are drawn.*

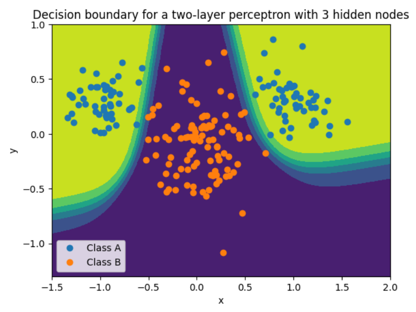
3.1 Classification and regression with a two-layer perceptron *(ca.2 pages)*

3.1.1 Classification of linearly non-separable data

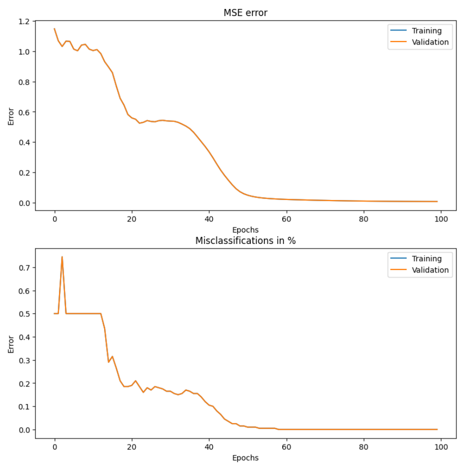
At first we created the model and implement the algorithms (forward pass, backward pass and weight update). After that we generate the dataset described in the paper and fixed the seed to be able to redo the same runs.
For the first part we didn't have a validation set so we use just for plotting the validation to be also the training set (just for plotting).
By adjusting the number of hidden nodes (with a learning rate of 0.1 and an alpha in momentum of 0.9), we saw that the MSE after 3/4 hidden nodes decreased drastically from 0.50 with 1 hidden node to 0.003 with 3 hidden nodes.



(a) Decision boundary with 1 hidden node



(b) Decision boundary with 3 hidden nodes

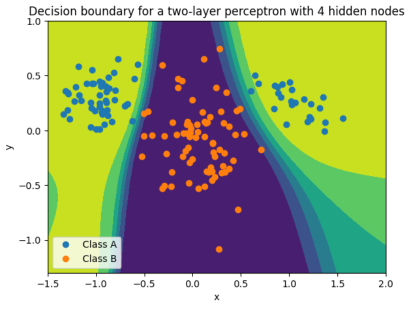


(c) Errors with 3 hidden nodes

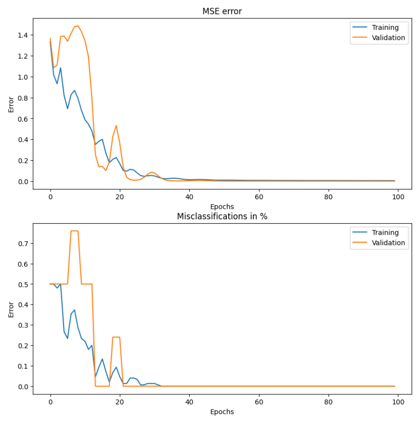
Figur 1: Evolution of the decision boundary with the number of hidden nodes

As we can see on the figure 1b, with 3 hidden nodes, the decision boundary is already very good and seems to provide a good untested generalization. We have also noticed that after adding more than 3/4 hidden nodes to the hidden layer, the MSE seems to be stabilized and the decision boundary seems to stagnate.

After that we removed parts of the initial dataset and use it as validation set. Then we trained our model with the same hyper-parameters and we obtained this :

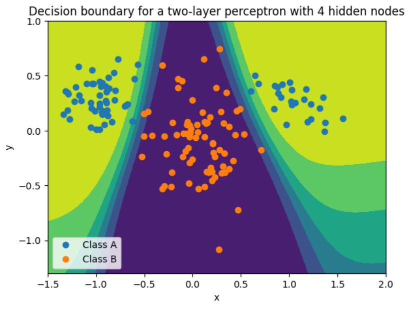


(a) Decision boundary - Batch

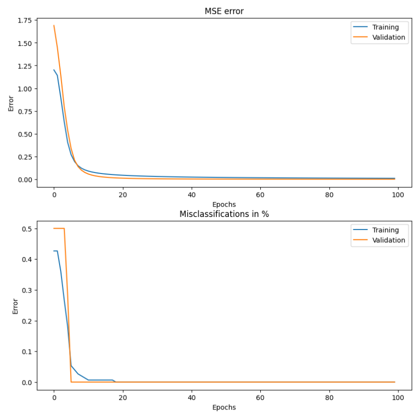


(b) Errors

Figur 2: Training with 50% of the dataset removed randomly - batch mode

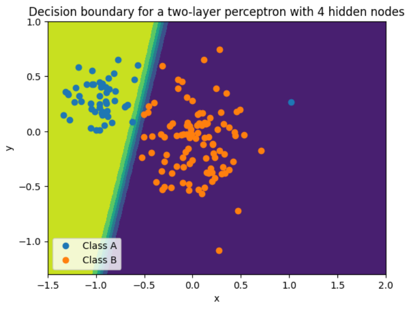


(a) Decision boundary - Seq

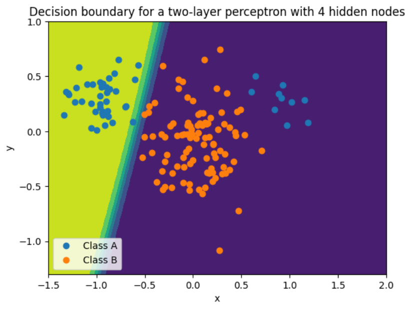


(b) Errors

Figur 3: Training with 50% of the dataset removed randomly - sequential mode



(a) Training with 50% of class A removed randomly



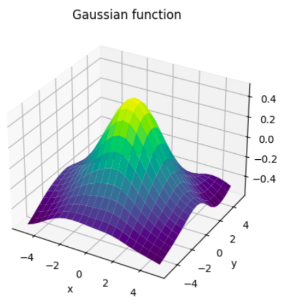
(b) Training with 20% of right and 80% of left of class A removed randomly

Figur 4: Training with only class A removed randomly

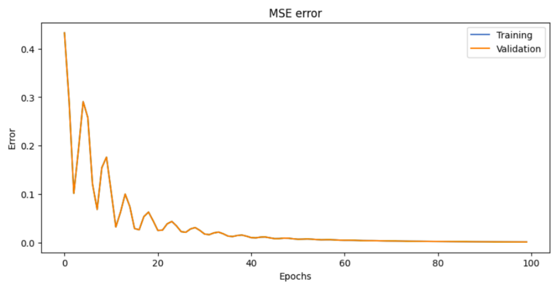
There are some things to say here. Firstly we can see that the validation error is always slightly above the training error (MSE and Miss-classification) [2b]. That is predictable because the algorithm update the weight relatively to the training. If we remove 50% randomly of the dataset, the model seems to predict the same decision boundary so it seems to work pretty (also with fewer epochs, due to the number of data) [2a]. Also, we have a slightly faster convergence in **sequential mode** [3b] than in **batch mode** [3a]. Furthermore, it's important to notice than by removing some precise points of the class A, the training take into accounts only the biggest cluster for training and therefore missclass a lot of points (this is what unlucky happen also for the 2nd case (4a) where we deleted 50% of class A randomly). We did not put the errors curves in the report for these cases because it seems pretty obvious that we will have strongly different validation and training error due to the fact that the validation and training set are totally different (one represent in majority the left and the other the right) [4b].

3.1.2 Function approximation

After generating the mesh and training the network on this mesh using the values of the bell-shaped Gauss function on the mesh points, we obtain a satisfactory approximation of the bell-shaped Gauss function on the domain and the MSE error is fairly low. It can be seen that when the number of units in the hidden layer is increased, the training error decreases, which makes sense as it increases the capacity of the network.

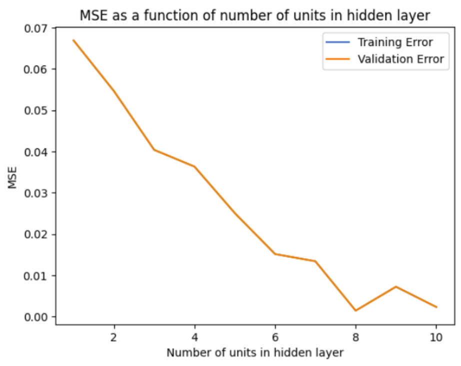


(a) Approximation of Gaussian bell function



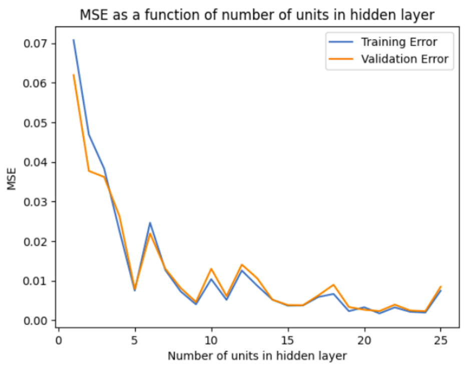
(b) Training MSE

Figur 5: Approximation of the Gaussian bell function with a MLP



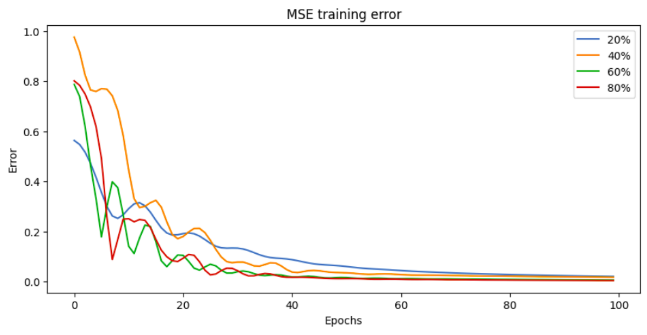
Figur 6: MSE as a function of the size of the hidden layer

By increasing the number of units in the hidden layer, the training error and the validation error decrease. It can be seen that when the number of units is too low (<7 units), the capacity of the network is too low and so the predictions fail to match the data (this is the phenomenon of under-fitting). Conversely, the MSE error is not optimal when the network capacity is too high (>23 units): the algorithm sticks too closely to the training data and generalises poorly. This is known as under-fitting. The optimal network capacity is a compromise between error minimisation and network complexity: in this case it is around 12 units. This is the model we have chosen for the rest of the study.

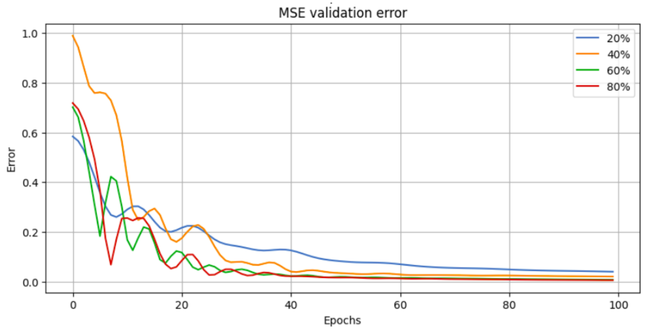


Figur 7: MSE as a function of the size of the hidden layer

When we vary the proportion of data used to train the model, we notice, as expected, that the higher the proportion, the better the model generalises. In particular, the validation error is much worse with only 20% of the data used for training. Nevertheless, all the attempts resulted in a fairly low MSE validation error, so the generalisation of the model was still satisfactory. However, convergence is fairly slow.



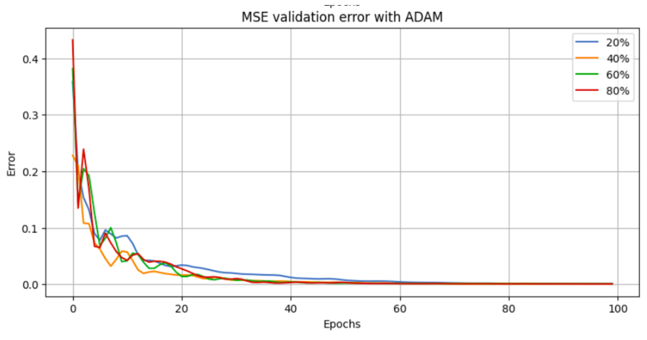
(a) Training MSE for different proportions of training data



(b) Validation MSE for different proportions of training data

Figur 8: Generalisation capacity of the MLP with different proportions of training data

To improve this without compromising the model's ability to generalise, we used gradient descent with ADAM, which adapts the learning rate according to past gradients by combining the momentum technique with the AdaDelta technique. We observed a clear improvement in convergence time, whatever the proportion of data used for training, without deteriorating the MSE validation error. We could have used other techniques as well, such as early stopping (which would have required another hold-out set) or other adaptive learning rate techniques.



Figur 9: Validation MSE using adaptative learning rate technique (ADAM)

3.2 Multi-layer perceptron for time series prediction *(ca.3 pages)*

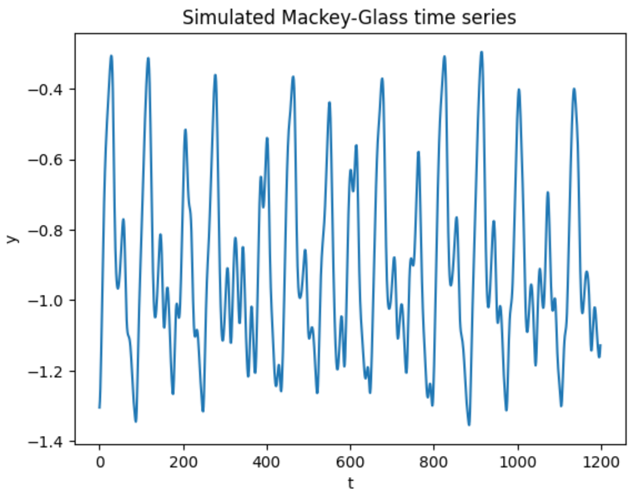
We will just quickly recall the structure of the MLP used :

- Dimension of the data points : 5
- Dimension of the output : 1
- Number of hidden layers : 2,
- Activation function : sigmoids in the hidden layer, linear for the output,
- Weight initialization : random
- Optimizer : Adam (chosen so that the learning rate, which is not studied in this lab, is not a problem), with initial learning rate of 0.001 (default value)
- Regularisers : early stopping (for free noise data) or weight decay (for noisy data)
- Number of epochs : 150 max

Moreover, as there is randomness due to weight initialization, and later because of the noise added to the data, when performing a grid search, we will take a look at the results for each parameter by training **10 times** the same model. When looking at loss values to compare models, we will look at the mean and the standard deviation of those models.

3.2.1 Three-layer perceptron for time series prediction - model selection, validation

First we generated the Mackey-Glass time series with the parameters given in the lab. We get the following time series :

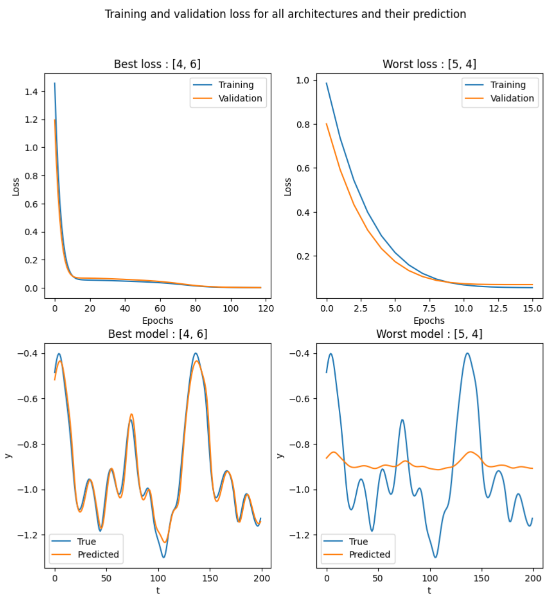


Figur 10

To make the dataset used for training the models, we just considered our data to be a list of 1200 points of dimension 5. The features are 5 consecutive values of the times series, and the associated label is the next value of the time series. Then, the output of the neural network is the next value of the time series, given the five previous ones.

Note : For this part, will use early stopping as a regulariser.

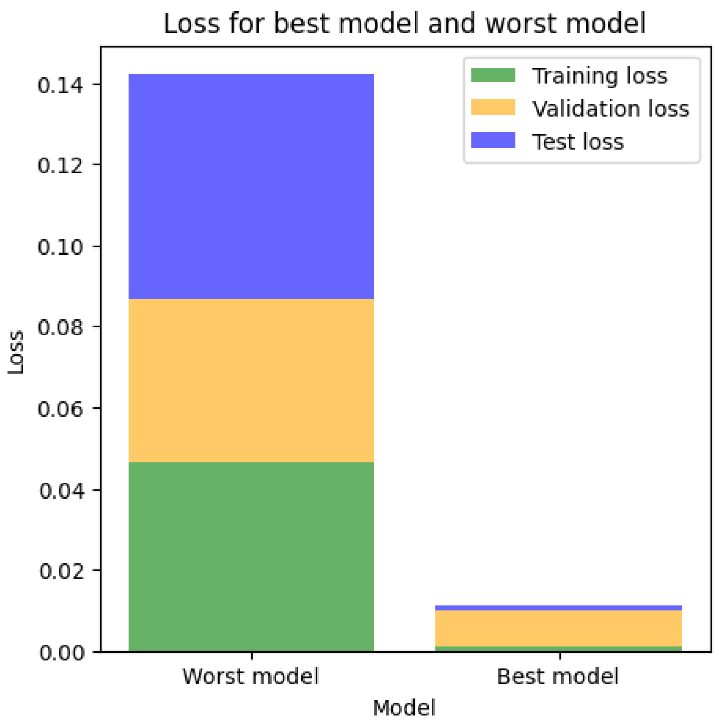
As said above, to find out which architecture are the worst and the best for this time series prediction, we performed a grid search on all the possible combinations of node number (nh1 = 3, 4, 5, nh2 = 2, 4, 6).
Then we chose the models that maximize and minimize the mean validation error after training 10 times each model with the same parameters.
An example of training is shown below with those models. The number of each node is also specified :



Figur 11

We can see that early stopping is working as the two trainings stop before the maximal number of epochs (150). In addition, we observe that the worst model clearly underfits the data, while the best one seems to give an accurate prediction of the time series.

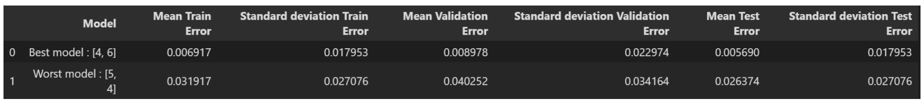
A chart providing a visualization of the different errors of those models is also provided below. We can clearly observe how lower is the error of the best model, compared to the worst, for all the datasets :



Figur 12

Note : Even if the graphs here can let think that the neural network can predict the whole time series, the MLP can only predict the next value of the time series, given the **exact** 5 previous ones. That's why the results seem so good.

To check the **robustness** of those models, we can take a look at the mean errors on the different sets we have over each 10 times trained model, as well as their standard deviation :



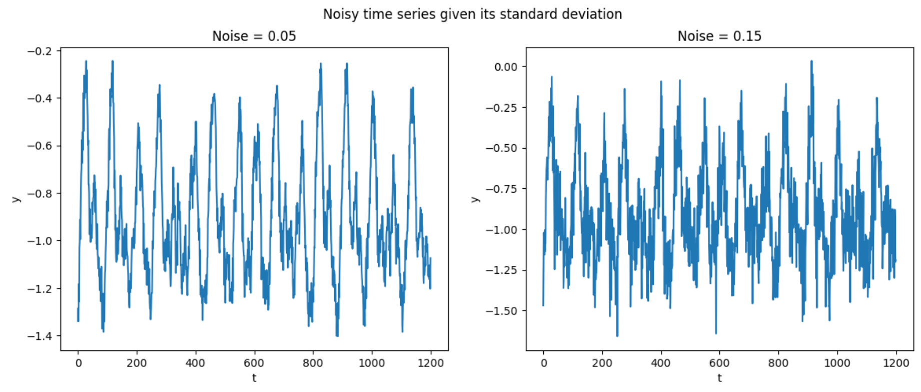
	Model	Mean Train Error	Standard deviation Train Error	Mean Validation Error	Standard deviation Validation Error	Mean Test Error	Standard deviation Test Error
0	Best model : [4, 6]	0.006917	0.017953	0.008978	0.022974	0.005690	0.017953
1	Worst model : [5, 4]	0.031917	0.027076	0.040252	0.034164	0.026374	0.027076

Figur 13: Statistics about the worst and best models for free noise data

Both models have a rather low mean error and standard deviation error when looking at the test set. We can notice that the mean error of the best model is particularly low (almost ten times less than the worst model), which makes it particularly robust. Indeed, we ran the simulations several times and the model was giving recurrent convincing results, despite the random weight initialization.

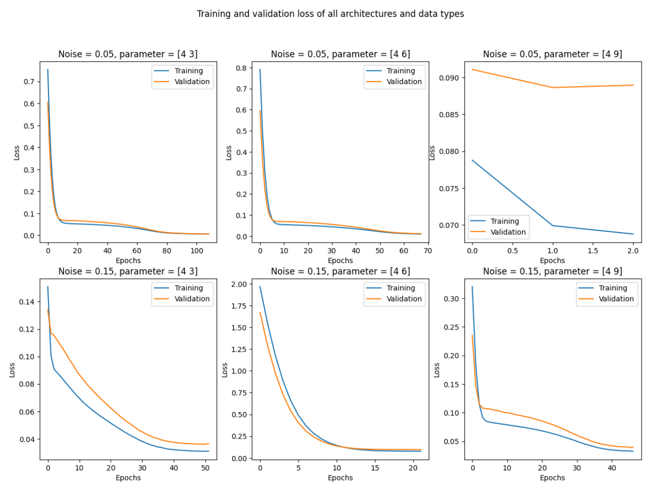
3.2.2 Three-layer perceptron for noisy time series prediction with penalty regularisation

For this part, we have to generate noisy data with two different santard deviation values. By adding the noise, we get the two following time series (the parameter values are specified in the title of the graphs) :

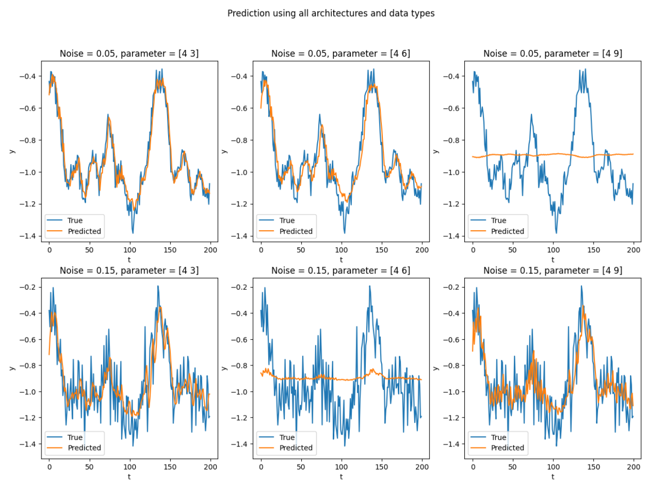


Figur 14

For this part, we keep the number of nodes of the first layer of the best model for free noise time series, that is, 4. We want to take a look how the number of nodes on the second hidden layer impacts the results on noised time series. We will replace here the early stopping by weight decay, with a first default value of 0.001.
We did the same as before, that is, performing a grid search on every layer configuration, training each model 10 times to get a mean and standard deviation of the errors on each dataset. It allows to have a better idea of their performance, as noise adds more randomness. Here is an example of results with all the architectures :



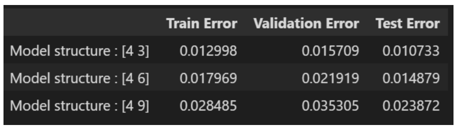
Figur 15



Figur 16

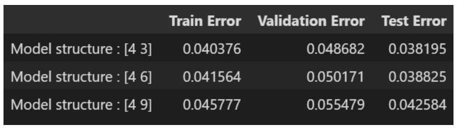
Most of the architectures seem to fit the data quite well, except two of them. It must be mainly due to weight initialization though, and does not reflect the average performance of the models. However, we can notice that the model with the biggest number of nodes on the second layer tends to overfit the datapoints when the standard deviation of the noise is higher.

Let’s now take a look at the statistical insights of the models :



	Train Error	Validation Error	Test Error
Model structure : [4 3]	0.012998	0.015709	0.010733
Model structure : [4 6]	0.017969	0.021919	0.014879
Model structure : [4 9]	0.028485	0.035305	0.023872

(a) Standard deviation of the time series : 0.05



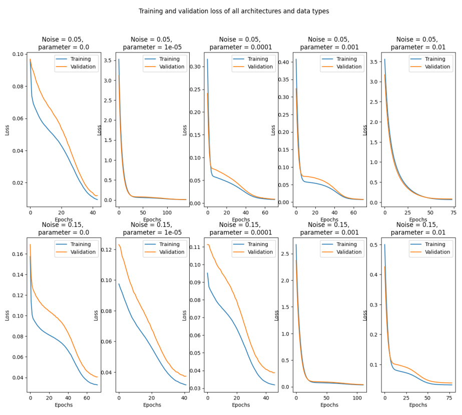
	Train Error	Validation Error	Test Error
Model structure : [4 3]	0.040376	0.048682	0.038195
Model structure : [4 6]	0.041564	0.050171	0.038825
Model structure : [4 9]	0.045777	0.055479	0.042584

(b) Standard deviation of the time series : 0.15

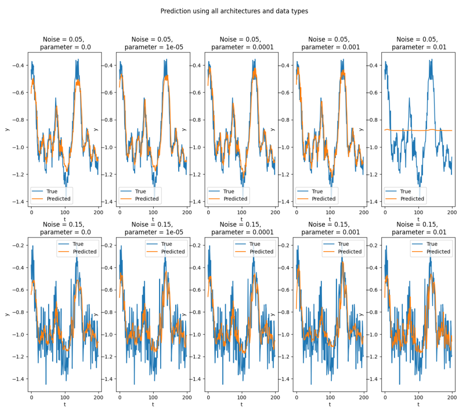
As expected, the errors increase with the noise.

By taking a look at the mean errors and the standard deviation values of the errors, we can notice that having a second layer with 3 nodes provides the best results overall (when looking at the validation error).
We will keep this structure to study the impact of the regulariser on the model results.

To study the impact of the regulariser (the parameter of the weight decay), we proceed as usual : we performed a grid search on several values of the regularisation parameter to sutdy the behavior of the models (regularisatoin values studied : 0, 0.00001, 0.0001, 0.001, 0.01). As before, we trained each model 10 times and stored the mean error values and their standard deviation. Here is an example of results with all the regularisation parameters :



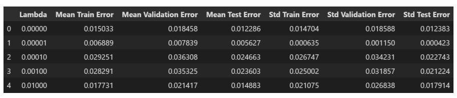
Figur 18



Figur 19

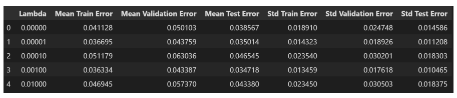
The results of the training give rather good prediction curves, except one of them. However, it still must be due to weight initialization rather than the model parameters. However, We don't really observe any differences when increasing the value of the regulariser. We would expect smoother curves as the regulariser increasing.

Let's now take a look at the statistical insights of the models :



	Lambda	Mean Train Error	Mean Validation Error	Mean Test Error	Std Train Error	Std Validation Error	Std Test Error
0	0.00000	0.015033	0.018458	0.012286	0.014704	0.018588	0.012383
1	0.00001	0.006889	0.007839	0.005627	0.000635	0.001150	0.000423
2	0.00010	0.029251	0.036308	0.024663	0.026747	0.034231	0.022743
3	0.00100	0.028291	0.035325	0.023603	0.025002	0.031857	0.021224
4	0.01000	0.017731	0.021417	0.014883	0.021075	0.026838	0.017914

(a) Standard deviation of the time series : 0.05

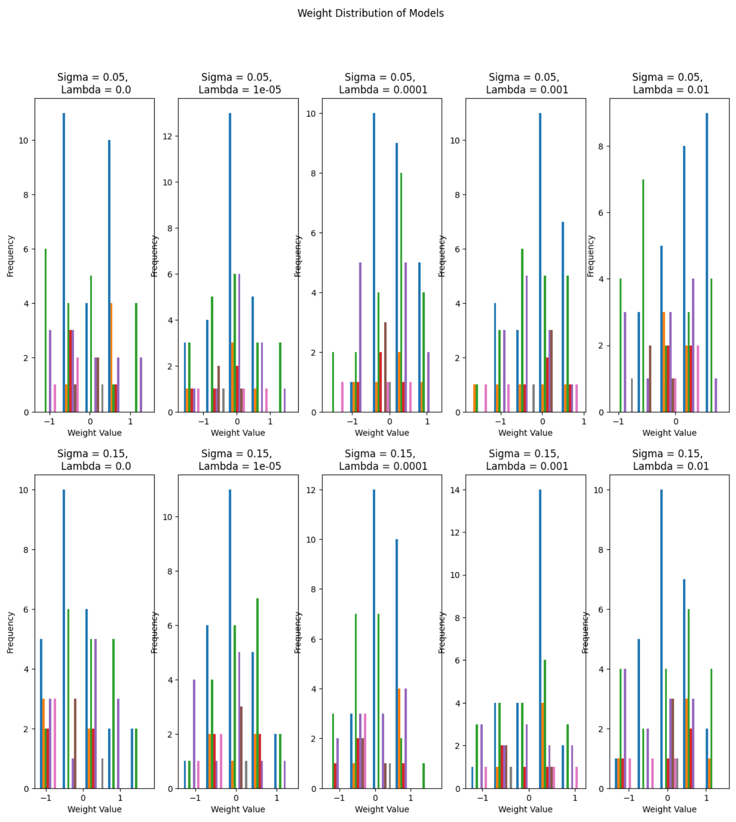


	Lambda	Mean Train Error	Mean Validation Error	Mean Test Error	Std Train Error	Std Validation Error	Std Test Error
0	0.00000	0.041128	0.050103	0.038567	0.018910	0.024748	0.014586
1	0.00001	0.036695	0.043759	0.035014	0.014323	0.018926	0.011208
2	0.00010	0.051179	0.063036	0.046545	0.023540	0.030201	0.018303
3	0.00100	0.036334	0.043387	0.034718	0.013459	0.017618	0.010465
4	0.01000	0.046945	0.057370	0.043380	0.023450	0.030503	0.018375

(b) Standard deviation of the time series : 0.15

We can observe that using the lowest non zero value of the regulariser (0.00001) gives the best results overall for a small standard deviation of the time series, as the mean error and its standard deviation are ten times lower than for other parameters. When increasing the noise of the time series, it turns out that choosing a higher regulariser value (0.001) gives slightly better results.

Finally, let's look at the weight distribution of the examples of trainings from above :



Figur 21

Same as before, when increasing the regulariser value, the weights are supposed to become smaller and close to zero, but we don't observe a clear trend on this graph.

4 Final remarks *(max 0.5 page)*

In the first part, our exploration focused on MLPs for classification of non-linearly separable data and function approximation. We firstly observed that adjusting the number of hidden nodes significantly influenced the Mean Squared Error (MSE) when classifying. Optimal performance was achieved with a moderate number of hidden nodes, showcasing the importance of finding a balance in model complexity. The function approximation part emphasized the role of the hidden layer size too. We identified an optimal capacity, balancing error minimization and model complexity. Varying the proportion of training data highlighted the trade-off between faster convergence and model generalization. Then, introducing an optimizer evidenced the significance of adaptive learning rates. This technique improved convergence speed without compromising generalization, providing a practical approach for enhancing MLP training.

Regarding the second part, the MLP provides good results for noise free data : the best model fits well the data and is robust, with its low standard deviation. The results remain satisfying for noisy data as well, but we couldn't observe major changes when changing the regulariser parameter, even if increasing the regulariser parameter slightly improves the training results. It might be due to the choice of the powerful optimizer Adam, as it might counterweight the regulariser when the loss gets too high.