Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1b

Learning with backpropagation and generalisation in multi-layer perceptrons

Côme Lassarat, Victor Sanchez and Marie-Ange Stefanos

February 9, 2022

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to design a Multi-Layer Perceptron (MLP) in order to solve classification and generalisation problems
- to get acquainted the backpropagation algorithm and to apply it in the designed MLP
- to monitor the performance of the previous computer-implemented process: error and robustness
- to identify the risks associated with backpropagation

2 Methods

In order to reach the intended goals, the different methods and algorithms were coded in Python, a wide-spread programming language. Several libraries were also imported and used:

- **Numpy** to handle multi-dimensional arrays and to generate linearly-separable and not linearly-separable data
- **Matplotlib** to construct graphs (of errors for example)
- **PyTorch** to build a neural network with 2 hidden layers

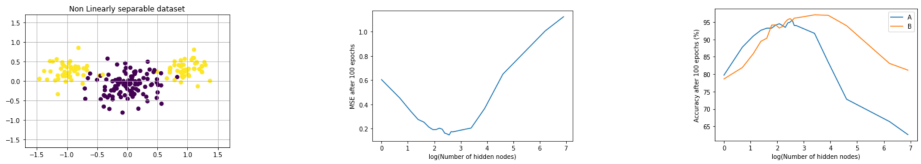
3 Results and discussion

3.1 Classification and regression with a two-layer perceptron

3.1.1 Classification of linearly non-separable data

Given a linearly non-separable dataset (see Figure 1), one can use a two-layer perceptron to perform classification and regression thanks to the backpropagation algorithm.

A first issue encountered is the choice of the number of nodes in the hidden layer. The mean-squared error (MSE) and classification accuracy of class A and class B can be seen on Figure 1. On this figure, the MSE is the lowest and the accuracy the highest for $N_{nodes} \approx 13$.



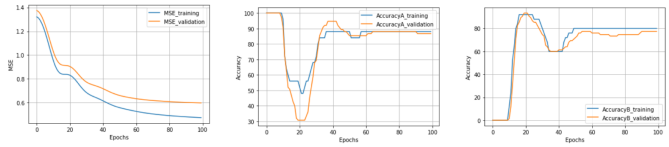
(a) (b) (c)

Figur 1: Data used (a), mean squared error (b) and accuracy according to the number of nodes in the hidden layer (logarithmic scale) (c) after 100 epochs

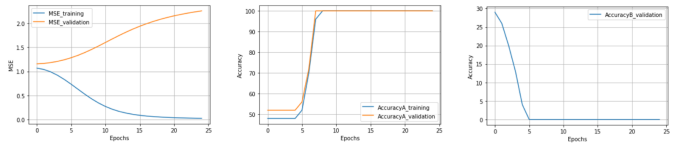
However, a more realistic approach is to sample the original dataset in a training set (to train the neural network using the backpropagation algorithm) and a validation set (to follow-up the generalisation ability of the network): the neural network can indeed perform well on the training data he used, but not necessarily on other data from the same set unused for training. Thus, the original dataset is subsampled in this way in different scenarios, to demonstrate the necessity to have a validation dataset well chosen. (Figure 2).

- Figure 2a highlights that training with data from both class A and B leads to a results with good accuracy (80-90%) and low MSE.
- Figure 2b underlines the need to train the neural networks with data from both classes: the $MSE_{training}$ is low but the $MSE_{validation}$ is high because the neural network wasn't trained to classify elements from class B. The same observation applies on the graphes of accuracy A and accuracy B.

When varying the number of hidden nodes in the most realistic sample scenario (Training = 25% from each class ; Validation = 75% from each class), similar results to those presented in Figure 1 are observed: an optimal of hidden nodes exists.



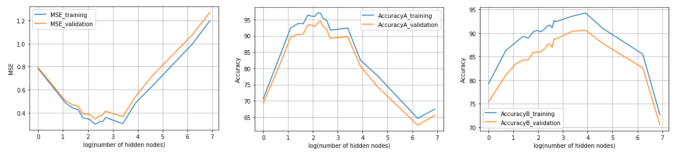
(a) Training = 25% from each class ; Validation = 75% from each class



(b) Training = 50% from classA ; Validation = 50% from classA

Figur 2: Sample scenario 1 (a) and sample scenario 2 (b). For each figure, from left to right: MSE, accuracy A and accuracy B according to the number of epochs

Finally, one can ask if there is any difference between a batch and sequential learning approach in terms of the validation performance: the validation in batch seems to be better because in on-line mode, the order of the samples counts, unlike the batch mode. Thus, the last samples studied might have more influence on the weight updates.

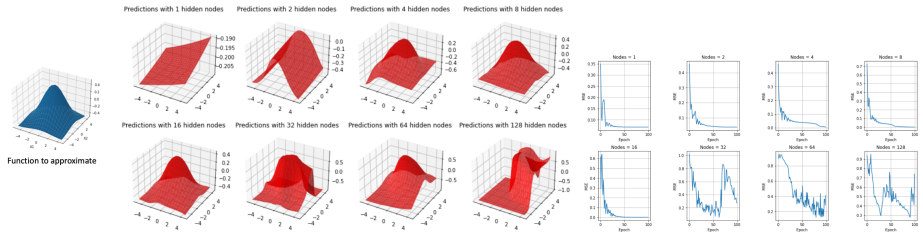


Figur 3: From left to right: MSE (training and validation), accuracy of A (training and validation) and accuracy of B (training and validation) according to number of hidden nodes after 100 epochs

3.1.2 Function approximation

A two-layer perceptron can also be used to approximate continuous functions. In this section, a neural network is designed to approximate $f(x, y) = \exp -(x^2 + y^2)/10 - 0.5$, by training it with input and output data. First, the influence of the number of hidden nodes on the final result is analyzed (Figure 5).

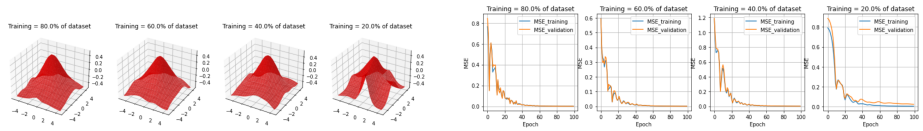
- Having 16 hidden nodes seems to leat to the best result
- For low numbers of hidden nodes (1 and 2), the perceptron seems to lack degrees of freedom to approximate the function. For high numbers of hidden nodes (> 32), the variance of the perceptron is high (overfitting).



(a) (b)

Figur 4: Output of the perceptron after training for different number of hidden nodes (red) compared to the target (blue) (a) and the associeted MSE (b)

Secondly, the original dataset is subsampled in to different set: a training set and a validation set. It will allow to check the generalisation of the trained perceptron (Figure 5).



(a) Output of the perceptron (b) Training and Validation MSE

Figur 5: Output of the perceptron after training for different sizes of training and validation sets for 16 hidden nodes. From left to right: (Training = 80% ; Validation = 20% of the set), (Training = 60% ; Validation = 40% of the set), (Training = 40% ; Validation = 60% of the set),(Training = 20% ; Validation = 80% of the set)

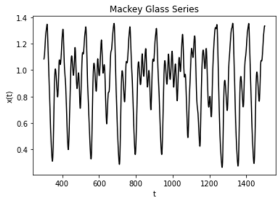
- The configuration (Training = 60% ; Validation = 40% of the set) seems to be the best trade-off: good generalisation and good speed of convergence.
- To increase the speed of convergence without compromising the generalisation, one can introduce a learning rate that varies through the epochs.

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

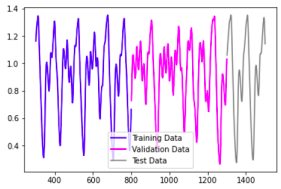
3.2.1 Three-layer perceptron for time series prediction - model selection, validation

The aim of this section is to predict the future points of Mackey Glass chaotic Series, that can be found on the right-hand side of the figure 6. In order to do that, we have built a neural network with 2 hidden layers by using *PyTorch* module. For this part, the whole data encompasses 2000 points that we divided as follows. The first 40% were used as the training set, the next 40% constituted

the validation set, and the rest of the data set was the testing set. These 3 groups can be visualized thanks to the left-hand side of the figure 6: the traning set is blue, the validation set is magenta and the test set is grey.



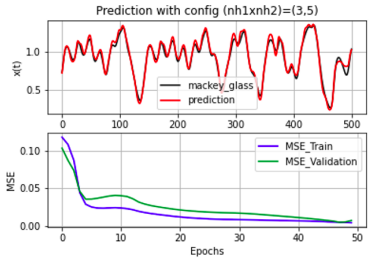
(a) Mackey Glass Series



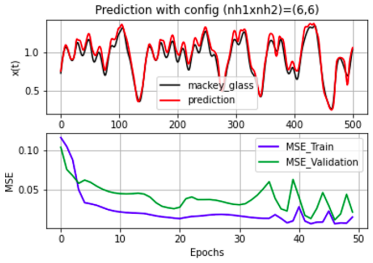
(b) Data splitting

Figur 6: The series to approximate and the data separation into different sets for training, validating and testing.

In order to know which architecture suits best for this problem, we tested several combination of number of hidden nodes in the first and the second layers. For each, we have averaged the prediction and the errors we obtained for training and validations sets on 10 iterations. You can find on the figure 7, the best (on the left-hand side) and the worst (on the right-hand side) results we got. For the architecture (nh1 x nh2) = (3, 5), we can see that the predicted curve is not only very close to Mackey Glass Series, but the errors for training and particularly for validation converges quite fast (no increase from epoch 20) to a very small value (lower than 0.01 after 50 epochs). However, for (nh1 x nh2) = (6, 6) architecture, the resulting prediction is not as similar to the original Series, compared to the previous architecture. Besides, the error has oscillations even after 40 epochs, which explains why we considered it as the worst result.



(a) Prediction with architecture (nh1 x nh2) = (3, 5)



(b) Prediction with architecture (nh1 x nh2) = (6, 6)

Figur 7: Predictions obtained with different architectures, with the corresponding training and validation errors.

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

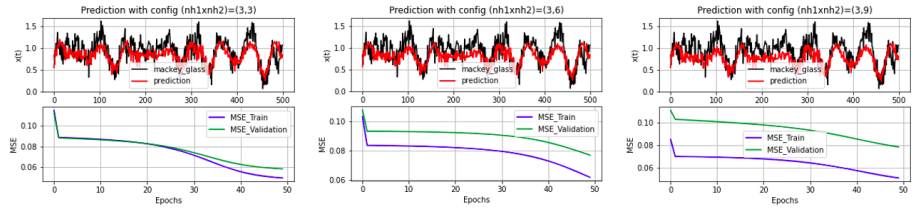
Even though predictions obtained in the previous subsections are widely satisfying, in the real world data are noisy. That is why we added some Gaussian

noise to the previous data set of Mackey Glass Series to have a new dataset to approximate. The noisy dataset can be seen in orange on the left-hand side of the figure 8, this is superposed with the dark curve, that corresponds to no noisy data. The σ coefficient has a value of 0.05 here. As in the previous subsection, this data set can be split into a training, a validation and a test sets, in the same proportion as before (see the right-hand side of the figure 8).

(a) Noisy data to predict (b) Noisy data separation

Figur 8: The noisy series to approximate and the splitting data into different sets for training (blue), validating (pink) and testing (grey).

We have seen in the previous subsection that the best architecture was architecture (nh1 x nh2) = (3, 5). That's why we kept constant the number of hidden nodes in the first layer nh1 = 3 and we changed the value of the number of hidden nodes in the second layer nh2 $\in \{3, 6, 9\}$. In order to obtain these results, we have set the *weight_decay* parameter of the optimiser function to 0.001. We can clearly see that the best results are obtained for the architecture (nh1 x nh2) = (3, 3), for which the error decreases the fastest. Moreover, the validation error is the lowest of the three results. This observation can also be seen directly on the top graph that shows that the prediction is way closer from the original noisy curve, compared to the two other architectures.



Figur 9: MLP network for time serie prediction with noise

4 Final remarks

To conclude, this lab allowed a deep understanding of the use of a two-layer perceptron. Its scope of application was highlighted by solving three different problems, underlying everytime its advantages, limits and design precautions. First, the classification problem showed the importance of choosing the right trade-off between the training and validation dataset size. The function approximation problem also pointed out this issue and went further by demonstrating

the importance of taking the right number of nodes in the hidden layer. Finally, given the learning from the two problems above, the time series prediction problem allowed the design of a three-layer perceptron by choosing the right configuration of hidden nodes and using a training, validation and test dataset. It also highlighted the strength of neural networks, where a signal polluted by noise can still be well approximated by such a tool.