Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1b

Learning with backpropagation and generalisation in MLP's

Emil Hed, Ivar Karlsson, Elias Kollberg

January 29th, 2025

1 Main objectives and scope of the assignment

Our major goals in the assignment were

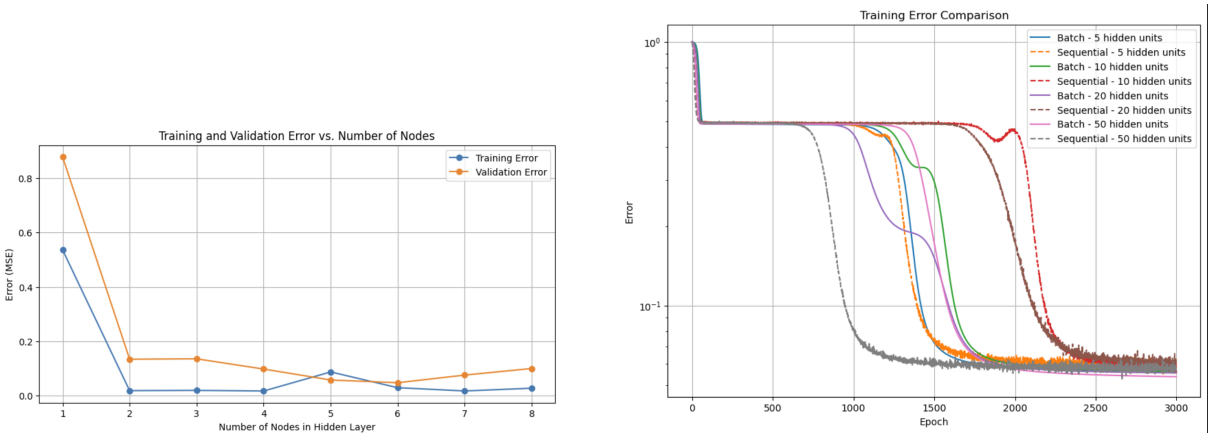
- To implement a multi-layer perceptron (MLP) capable of classifying non-linearly separable data
- To use the MLP to approximate the Gauss function
- To employ a three-layer perceptron with varying numbers of nodes in the hidden layers for the time-series prediction of the Mackey-Glass function.

The scope of the assignment has been to evaluate and demonstrate the models ability to learn complex patterns and its capability to approximate functions. And in the second part we have used a more complex model with three layers, examining the effects of the network architecture.

2 Methods

For this assignment we have used Python and Jupyter Notebooks as our primary programming tools, with Visual Studio. We used libraries including Numpy for numerical operations, Matplotlib for data visualization, and for the second part we used PyTorch for building the networks.

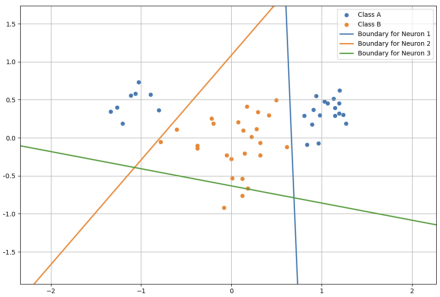
3 Results and discussion



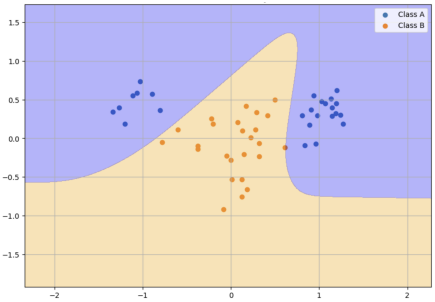
(a) MSE on the model, dependent on hidden layer size

(b) Sequential VS Batch mode with different amount on nodes in the hidden layer

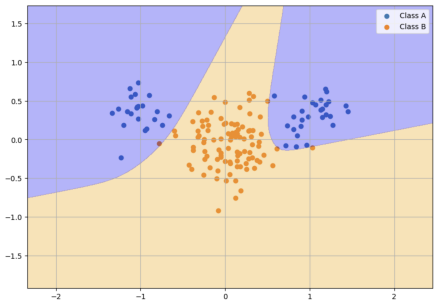
Figure 1: MSE and training error for different sizes of hidden layers



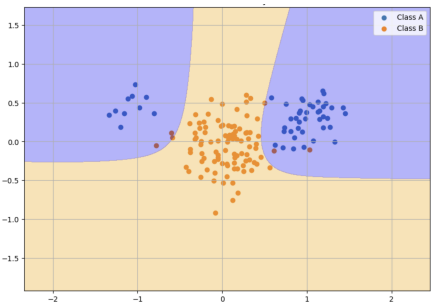
(a) The weights of the (3) nodes represented as individual boundaries



(b) The total boundary for the system with the training data on the 20% subset

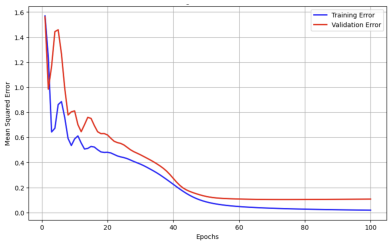


(c) The total boundary for the system with the training data, for the 50% subset

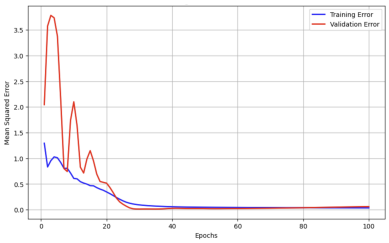


(d) Total boundary of the system with the training data for the 20/80 subset

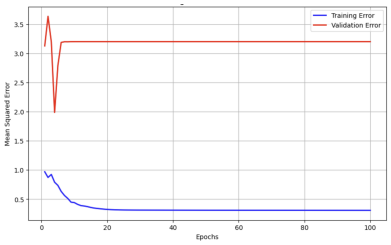
Figure 2: Graphs for the boundaries of the three subset scenarios



(a) MSE for training data and validation data on the 25% subset

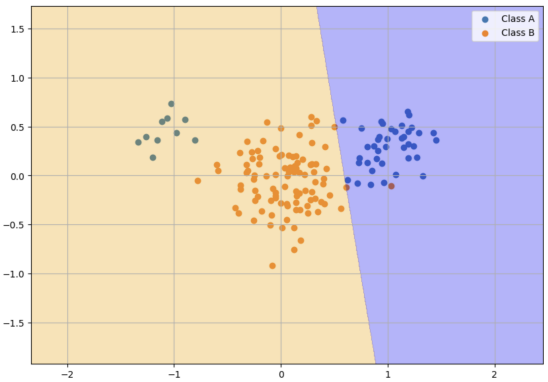


(b) MSE for both training and validation data, on the 50% subset

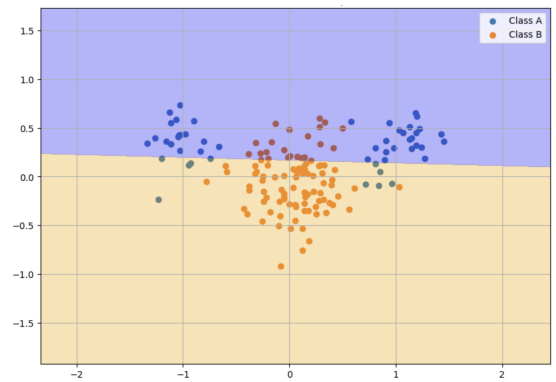


(c) MSE for both training and validation data, on the 20/80% subset

Figure 3: Loss functions (MSE) for the three subset scenarios

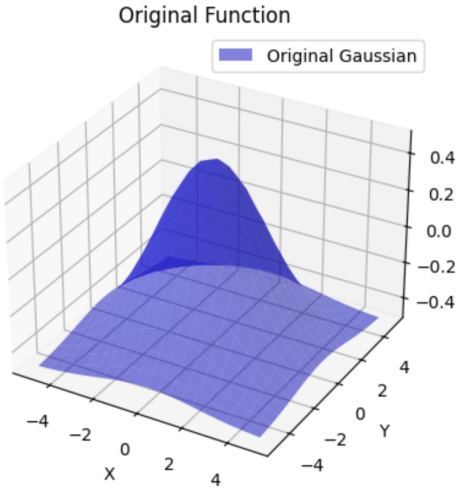


(a) Local minimum scenario on the 20/80% case

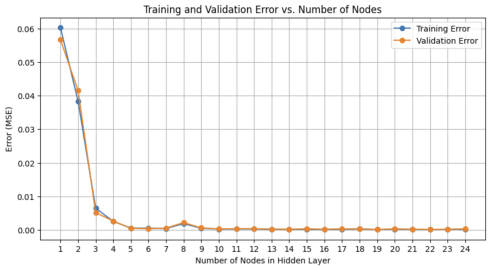


(b) Local minimum scenario for the 50% case

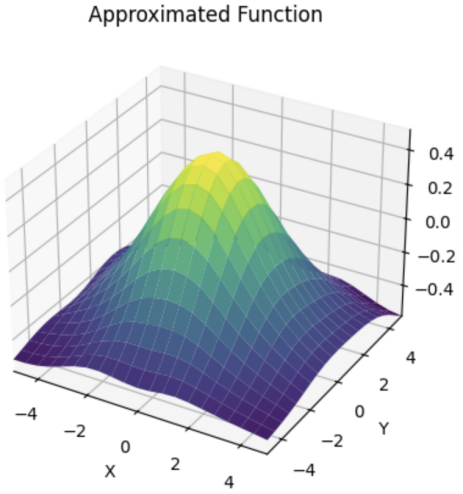
Figure 4: Local Minimum cases



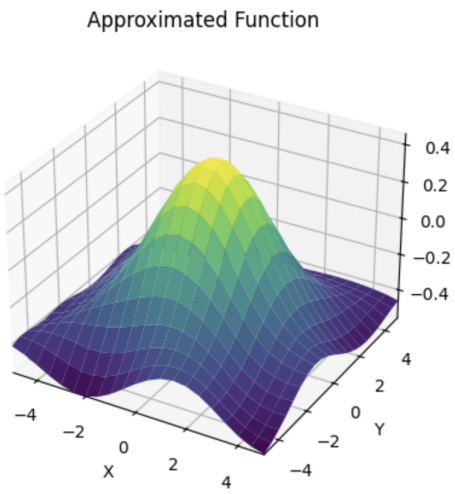
(a) The original Gaussian function



(b) The final converged MSE for models off different sizes, from 1 node to 25.



(c) The approximated gaussian where 80% of the data was used as training data



(d) The approximated gaussian where 20% of the data was used as training data

Figure 5: Graphs from the function approximation assignment

3.1 Classification and regression with a two-layer perceptron

3.1.1 Classification of linearly non-separable data

Lets begin by calling the scenario where we have a subset containing of 80% of the right cluster and 20% of the left cluster from class A the 20%/80% case, the scenario where we have selected 25% of each class for the 25% case, and the scenario where we have selected 50% of class A for the 50% case for future ease.

Initialization matters a lot for this model, especially in the 20%/80% case. It is quite common (regardless of the amount of nodes in the hidden layer and other hyper-parameters) that the model fails to capture the small cluster on the left side, which can be seen in figure (4a). It can also happen from time to time for the other models, for example in the 50% case, where a perfect horizontal line seems to get stuck, this can be seen in figure (4b).

Considering that we have the dataset that we have, we can quickly see by intuition that two straight lines should suffice to be able to split up the clusters well. If we try to increase the amount of nodes in the

hidden layer, the boundaries around the clusters don't really change that much, the only considerable difference we can see is that the line is a bit more complex, especially around the edges. This means that 2 nodes is good enough for our dataset, and adding more nodes after this does not help much. However, if we were to add a third cluster of class A beneath cluster B, a third node would indeed make a massive difference, just like we get when increasing the number of nodes from 1 to 2 in the normal case. Figure (1b) shows how the 2 nodes are enough to separate the two classes successfully.

The learning curves for the different subsets are quite similar, except for the 20%/80% case, where the graph seen in figure (3c) often get a validation error that is high, depending on whether the model succeeds in finding both clusters. It should be mentioned that this figure represents the local minimum scenario seen in figure (4a), where the validation data has reversed size of the two clusters for class A, meaning that if we don't capture the small cluster in the training data, the entire big cluster in the validation data will be miss-classified, which explains the high MSE for the validation data.

Looking at figure (1b), it can be hard to see a clear pattern. For some models, the batch mode is faster than the sequential mode, and for others it is the other way, and they all seem to converge towards the same final loss. Given this, it is difficult to draw clear conclusions from these results.

3.1.2 Function approximation

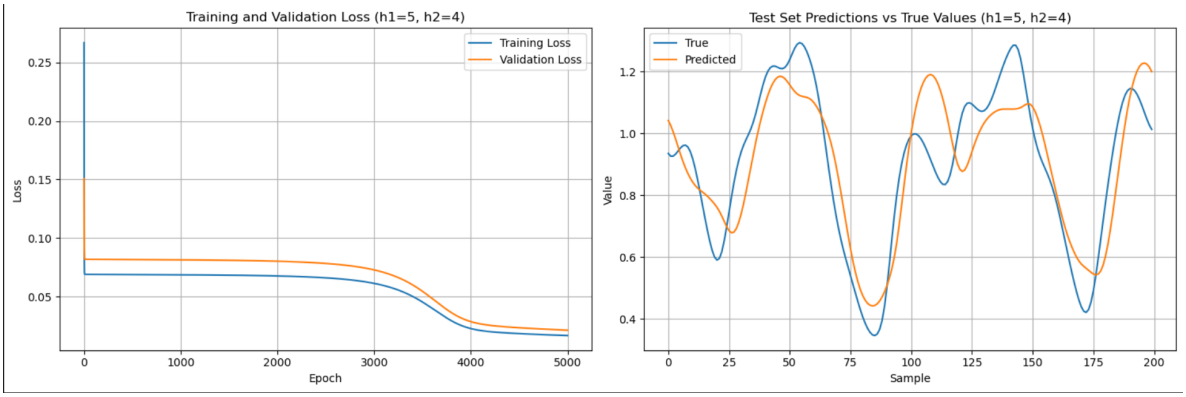
Quite like the earlier task, we can see that having an overly complex model doesn't really help much, and the error seems to be the same for models between 5 and 25 nodes in the hidden layer. However, we can see that for the case where we have an overly simplistic model (one or two nodes), the model fails completely in approximating the function, which is expected. We selected 10 nodes in the hidden layer as the best one, and tried running the experiment with two different scenarios, 80% of the data used as training data and 20% used as training data. These can be seen in figure (5c) and figure (5d). Both were quite fine, except that the edges of the graph that only used 20% of the data for training was more uneven, which is expected.

We can speed up the convergence without compromising the generalization performance in several ways, for example learning rate. Adjusting the learning rate is one of the simplest ways to speed up convergence, even if this has to be done carefully. One more safe way to do this could be to use an adaptive learning rate, such as Adam, Adagrad, etc... Other upgrades that could be implemented are batch normalization, early stopping, more proper initialization techniques, etc...

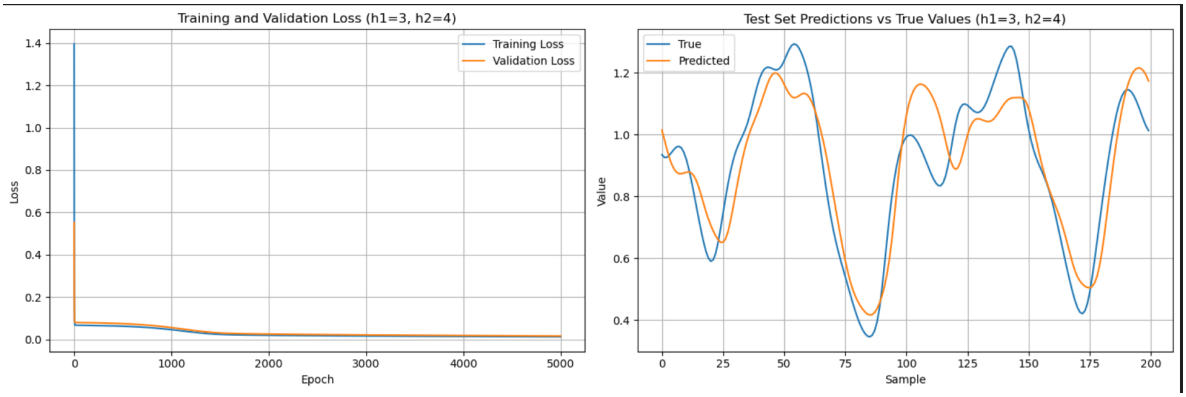
3.2 Multi-layer perceptron for time series prediction

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

The model selection process consisted of a grid search with varying sizes of the hidden layer. The learning rate was set to 0.1 and the number of epochs was 5000 for all experiments. The loss metric used was mean squared error and the optimizer was Stochastic Gradient Descent (SGD). For the first hidden layer the size was varied between 3,4 and 5 and for the second hidden layer between 2,4 and 6. The training loss and validation loss were tracked for all of the nine configurations and the best and worst performing configurations were noted. The best configuration was $[nh1 = 3, nh2 = 4]$ and the worst was $[nh1 = 5, nh2 = 4]$. A general trend that can be seen is that all the different configurations converge to a average mean squared error within a small interval between 0.016 and 0.011. Also when inspecting the predicted series of the different models after training, they were all similar in performance. However, the gulf in performance between the best and the worst is notable. The worst-performing model misses the patterns in certain sequences that the best-performing model captures. Both the learning rate and prediction power of the worst and best model is showcased below in figure (6).



(a) The worst configuration.



(b) The best configuration.

Figure 6: Comparison between best and worst configuration of hidden layer sizes

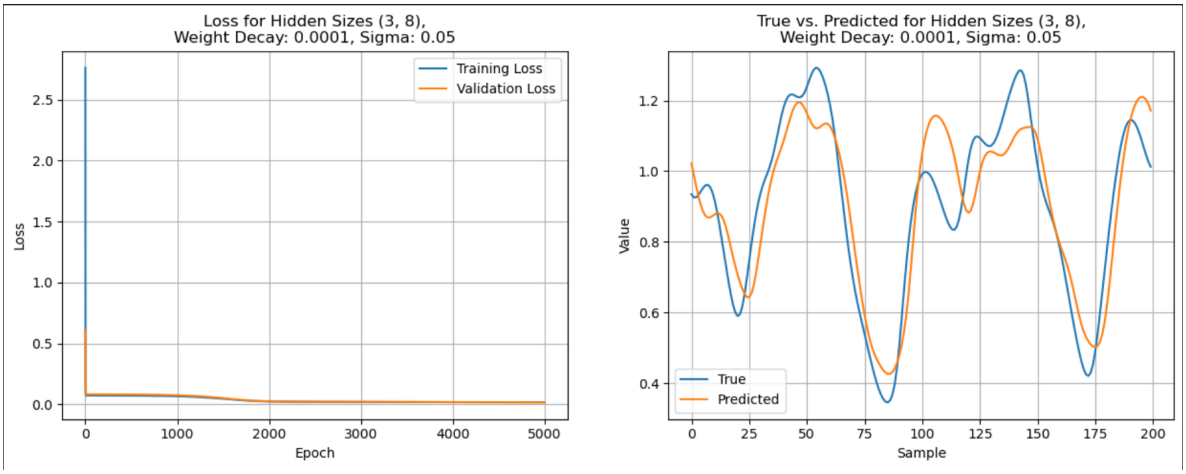
	nh1=3	nh1=4	nh=5
nh2=2	0.0116	0.0123	0.0126
nh2=4	0.0114	0.0118	0.0161
nh2=6	0.0130	0.0117	0.0125

Table 1: A table displaying the final average MSE per element for each configuration in the grid search.

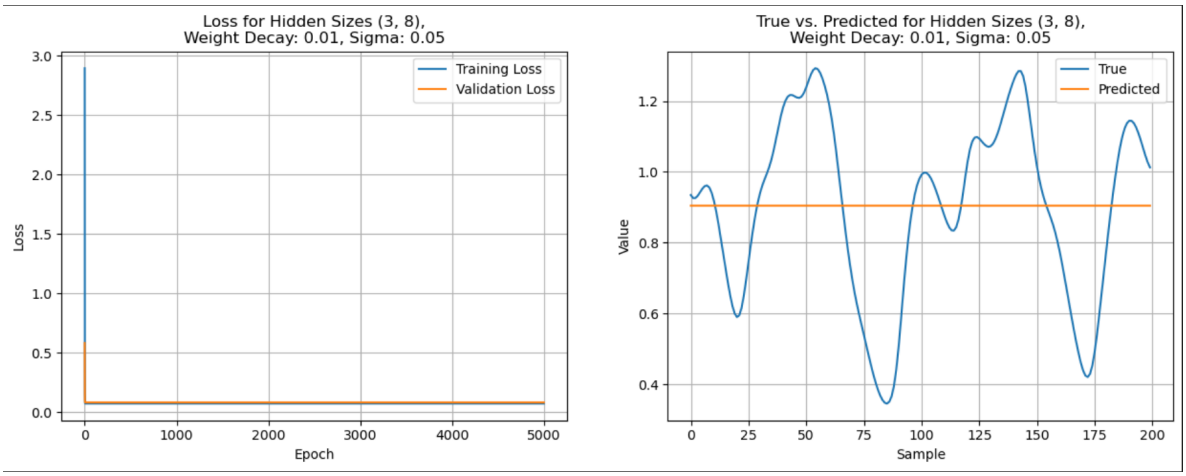
It should be noted that the best configuration was different in different runs, which indicates that the random aspect of initialization plays a significant role in what configuration that comes out on top. This indicates that the sizes of the hidden layer is not the main contributor to what configuration comes out on top, and thus is not that robust to random initialization. Early stopping was implemented, but was never triggered as the validation error never increased in enough consecutive epochs for it to be activated.

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

From the previous step the best result was given by the network architecture with a 3x4 node setup and the same hyper-parameters. The Mackey-Glass series was produced with some added zero mean Gaussian noise with varying deviation to simulate real-world conditions. The model was then modified to inlclude L2-regularization (weight decay) with different values of λ. However, our results suggest that a weight decay only have a meaningful impact when λ is set very low (10^{-4}) figure (7a). When λ was increased beyond this ($10^{-}2$) figure (7b), the models predictions degraded, converging to a constant output regardless of the input. This suggests that the network is highly sensitive to regularization, and that strong weight decay forces the weights close to zero, preventing the model from learning the patterns. This abrupt transition between useful and degrading regularization indicates that while weight decay can be useful for noisy data, its implementation in this specific case requires careful tuning.



(a) Successful L2-regularization.



(b) Failed L2-regularization.

Figure 7: Best L2-regularization, and failed L2-regularization

4 Final remarks

The lab covered a range of topics related to backpropagation through different models. The first part was closely related to the first part of lab 1 and it felt repetetive. Howver, the second part generated many insights as to how different heuristics affect convergence speed, what local minima is found and how to prevent overfitting. It was very educational to try out different configurations of parameters of both the network and learning algorithms. This part encouraged free experimentation of the configuration, which led to being able to isolate all the different heuristics to see what effect it had on the training and also how they can work in synchrony to create powerful models.