Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1b

Learning with backpropagation and generalisation in multi-layer perceptrons

Isabella Rositi, Gustav Thorén and Nicolas Guy Albert Wittmann

2 Febraury, 2023

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to design MLP networks and use them for classification, function approximation, and generalization problem
- to track the behavior of learning for MLP networks using backpropagation
- to identify limits of backpropagation and to build robust MLP models

The first section of this lab will concentrate on training a two-layered perceptron for binary classification and function approximation. We will implement a generalized delta rule, also known as error back-propagation algorithm, which is based on the idea of gradient descent.

The second section focuses on using MLPs to resolve the issue of chaotic time series prediction with the goal of providing a solid solution with high generalization abilities.

2 Methods

In order to create a MLP class in Python we used the *Numpy* library and used *Matplotlib* to produce figures. Moreover, for the time series prediction we used the *Tensorflow* library

3 Results and discussion

3.1 Classification with a two-layer perceptron

3.1.1 Classification of linearly non-separable data

We split the data into a training and validation set using 3 different approaches, however two of them provide validation sets drawn entirely from class A, which results in higher errors and lower accuracies when validating the model. Therefore, we will focus on presenting the results for the balanced split: 25% of randomly chosen data points from class A and 25% from class B as the test set.

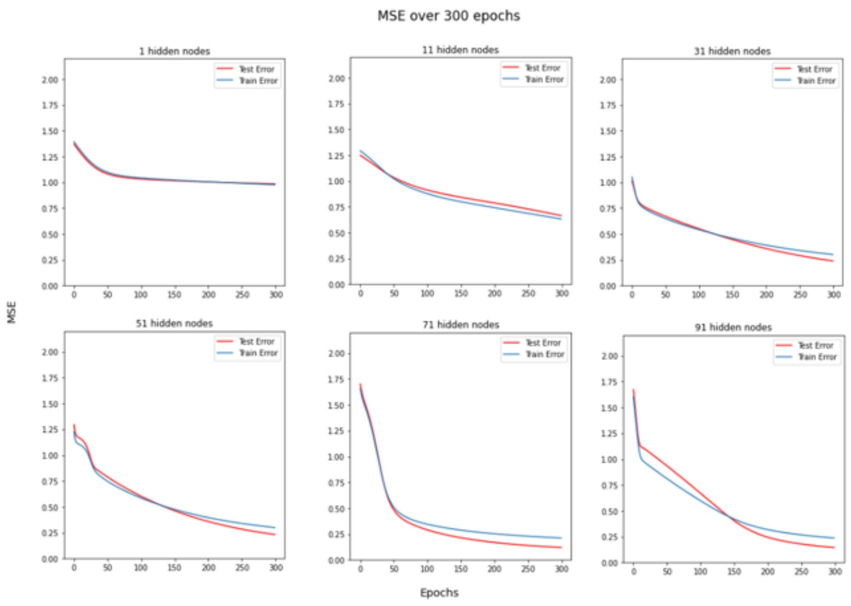


Figure 1: Training and test error over 300 epochs for number of hidden nodes going from 1 to 91

From Fig. 1 we can see that for 1 hidden node both the test and training error remain constantly high over the 300 epochs, meaning that the model is not complex enough to learn the data (not even the training data). For the other number of hidden neurons, we observe that both the training and test errors decrease with the number of epochs, meaning that the model has not started overfitting yet, in fact both error are low. Moreover, if we focus on the test error after 300 epochs for each number of hidden nodes, we can affirm that as the number of hidden nodes increases, so does the performance of the of the MLP model. However, if the number was to increase too much, we would observe an increase of the test error due to overfitting. The difference between training and test error shown here is mainly due to randomization, in fact the small size of the data sets (150 and 50 patterns) makes the learning very susceptible to both weight initialization and choice of random seed. The reason for the test error generally being lower than the training error can be seen from the distribution of the test set, Fig 2. Typically the test error should be higher than the training error but since the the test set just happened to be drawn in a way which is easily separated compared to the training set the test error is consistently lower.

Finally we can see in Fig. 2 the raw decision boundary produced by the MLP. We observe that all except one data point was correctly classified by the model.

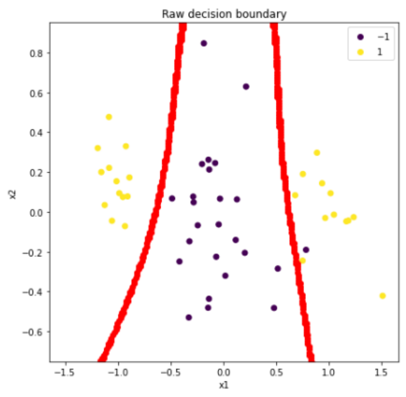


Figure 2: Raw decision boundary for a two-layer perceptron ran for 300 epochs, with 71 hidden nodes, learning rate equal to 0.001 and momentum equal to 0.25

3.1.2 Function approximation

The network was trained to approximate a two dimensional Gaussian bell function, Fig. 3. This was done by training the network with a sequence of coordinates as input and the function value as targets. The best performing model (where the model is specified as initial weights and number of hidden nodes) was found by training many networks with the same learning rate and momentum ($\eta = 0.01$ and $\alpha = 0.1$).

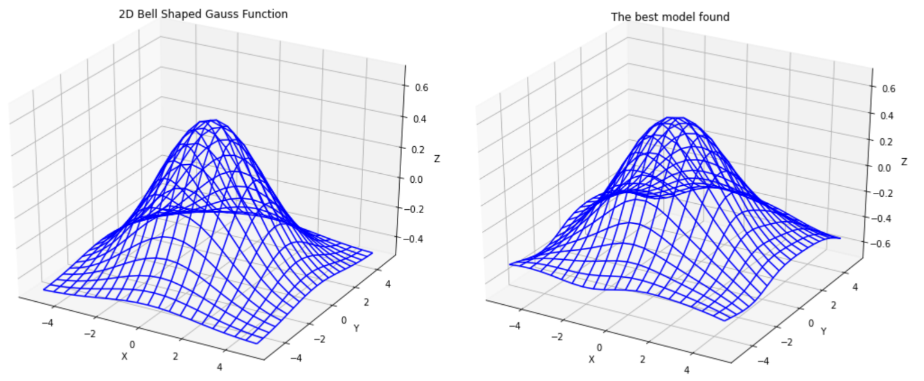


Figure 3: The desired function and the best found function when training over 200 epochs

All networks were trained over 200 epochs but with different initial weights and number of hidden nodes. The model with the lowest MSE was saved as the best model. The function approximation of the best model can be seen in Fig. 3 and has 6 hidden nodes.

Fig. 4 shows the best performing model with different amount of hidden nodes. The plot shows that models with few but not too few hidden nodes perform the best. If the model has too few hidden nodes it will lack the expressiveness required to fully capture the desired function. In the extreme case of only 1 hidden node it will essentially behave as one layered perceptron. On the other hand, if there are two many hidden nodes the model will also perform badly. This is mainly an effect of a model with many hidden nodes having more local minima that it tends to get stuck in. It could also be that the expressiveness that comes from having more nodes also requires more data to properly train. In Fig. 4 it seems as if the performance starts improving at 25 hidden nodes however this is simply a case of having good initial weights.

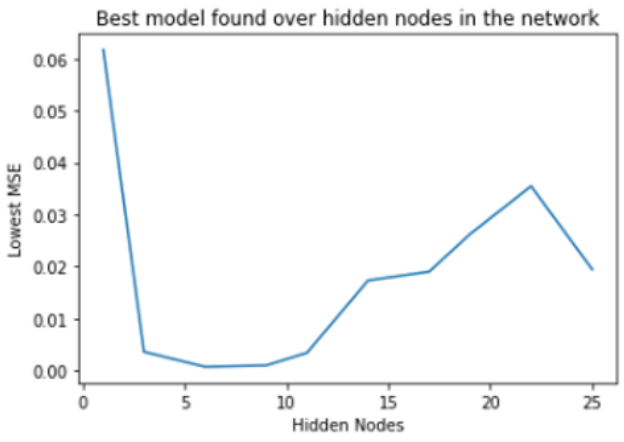


Figure 4: Best performing model compared to how many hidden nodes.

After finding the best model, the generalization of the network was tested. The training data was randomly split and the best model was trained for 500 epochs on the split data. The result of this can be seen in Fig. 5. The model performed very consistently both in terms of mean and standard deviation of the MSE up until less than 40 percent of the data was available. After that the model performance quickly deteriorated as there was no longer enough data to easily approximate the true function. Training the model for less epochs made the performance worsen earlier.

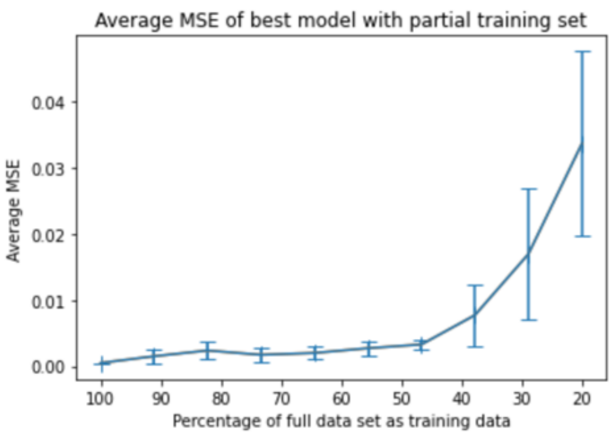


Figure 5: Performance of the best found model when trained on a partial data set.

3.2 Multi-layer perceptron for time series prediction

3.2.1 Three-layer perceptron for time series prediction - Model selection and Validation

For this section we decided to use the *TensorFlow* library to perform chaotic time-series prediction. 1200 points were generated from

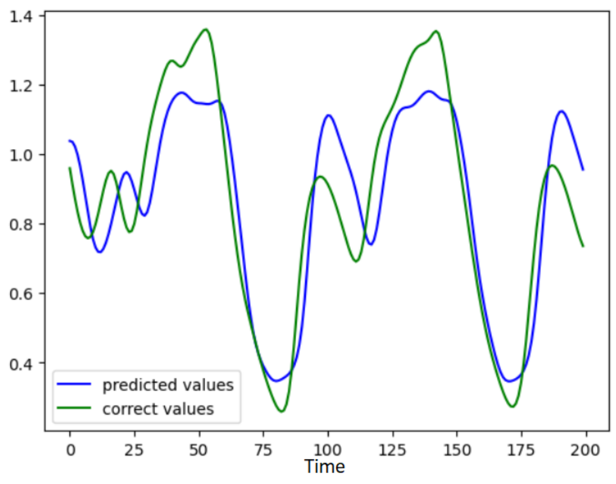
$$x(t+1) = x(t) + \frac{0.2x(t-25)}{1+x^{10}(t-25)} - 0.1x(t)$$

Our model is a MLP model with two hidden layers with sigmoid activation functions in the hidden layers and a linear activation function in the output layer. We used stochastic gradient descent as an optimisation method with a learning rate of 0.1 and a weight decay of 10^{-6}. We use early stopping on the condition that if the validation error does not improve after 3 learning epochs, we stop the learning process.

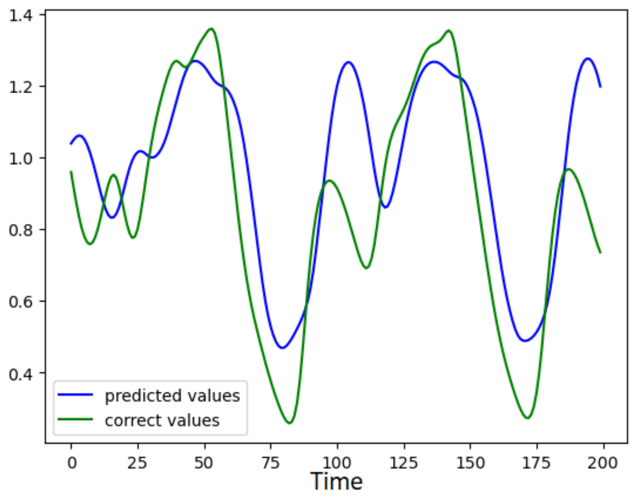
In order to find the best model in terms of hidden nodes in each hidden layer we tested all possible combination of hidden layer size from the ensemble $n_{h1} * n_{h2}$ where $n_{h1} = [3, 4, 5]$ is a set of different numbers of hidden nodes in the first hidden layer and $n_{h2} = [2, 4, 6]$ is a set of different numbers of hidden nodes in the second hidden layer. We ran the model several times to measure robustness in relation to random initialization of weights.

We obtained best results for (4,2) hidden nodes with an average mean squared error of 0.015, and the worst results for (3,6) hidden nodes with an average mean squared error of 0.032 on the validation set. The performance of the best found network can be seen in Fig. 6a and the performance of the worst network found can be seen in Fig. 6b. The graphs show that even the worst performing model is capable of estimating the general form of the time series, although there are large differences between the magnitude of the predicted signal and the real signal.

It is difficult to say why these models performed best and worst. One theory is that since the performance was averaged over many iterations the models with low amount of hidden nodes (and hence less local minima) are easier to consistently train to a decent performance. On the other hand, models with many hidden nodes are capable of performing very well in the global minima but have many more local minima which they tend to get stuck in and therefore perform worse in general.



(a) Best model



(b) Worst model

Figure 6: Predicted and correct values with worse and best model

When evaluating the trained models on the test set, we obtain an average MSE of 0.018 with (4,2) architecture and 0.042 with (3,6) architecture. The variance we obtain with (4,2) architecture is also smaller (1.6*10^{-5}) compared to the one with (3,6) architecture (3.6*10^{-4}). So the best performing architecture is also more robust in relation to random weight initialisation.

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

By adding Gaussian noise to the training data we can make it more difficult for the model to grasp and learn the underlying function. Tab. 1 shows the results of training the model with varying noise and regularization parameter values. The results show that increasing the regularization parameter increases the performance of complex models. This is because the regularization (L2 norm) forces the weights to be lowered which leads to less overfitting when the model can no longer capture as much noise. There seems to be an incentive to increase the regularisation parameter when there is more noise.

	$\lambda = 10^{-6}$		$\lambda = 10^{-4}$	
Hidden Nodes	σ =**0.05**	σ =**0.15**	σ = **0.05**	σ = **0.15**
3	0.010	0.022	0.016	0.026
6	0.014	0.018	0.017	0.020
9	0.020	0.030	0.018	0.022

Table 1: Average MSE for different values of noise, regularization parameters and number of hidden nodes. The numbers of hidden nodes refer to the second hidden layer, in the first layer there are always 4 hidden nodes.

Finally, for medium noise (sigma=0.09) we obtain best result with (4,6) architecture and a MSE of 0.16 on test data. This is worse than what we obtained on clean data for best models but reasonably close.

4 Final remarks

The number of hidden nodes in a two-layer MLP influences the performance of the model for both classification tasks and function approximation. In both scenarios too few hidden nodes are not able to capture the complexity of the data or the function, resulting in high errors when predicting and poor approximations. On the other hand, high numbers of hidden nodes bring the model to get stuck in more local minima, which also increases the errors and lowers the approximation power of the model.

For the time series the results encountered were similar, accounting for a general worse performance of models with higher numbers of hidden nodes. Moreover, when Gaussian noise is added to the training data the performance worsen, however increasing the regularization parameter also increases the performance of complex models.