Neural Networks
Group 04
Andrea Jemmett (2573223)
Bas van der Drift (1707507)

Assignment 3 - Multi-Layer Perceptrons

Section A

The main Matlab script for this part of the assignment can be found attached as 'ass3a_clean.m'.

Netlab's multi-layer perceptron (MLP) function was used for three regression tasks involving datasets with a linear, a sinusoidal and an irregular correlation. The perceptron was trained using five-fold cross validation, in which the dataset is divided in five equal parts and then used for training and testing in each of the five possible configurations of four parts training, one part testing.

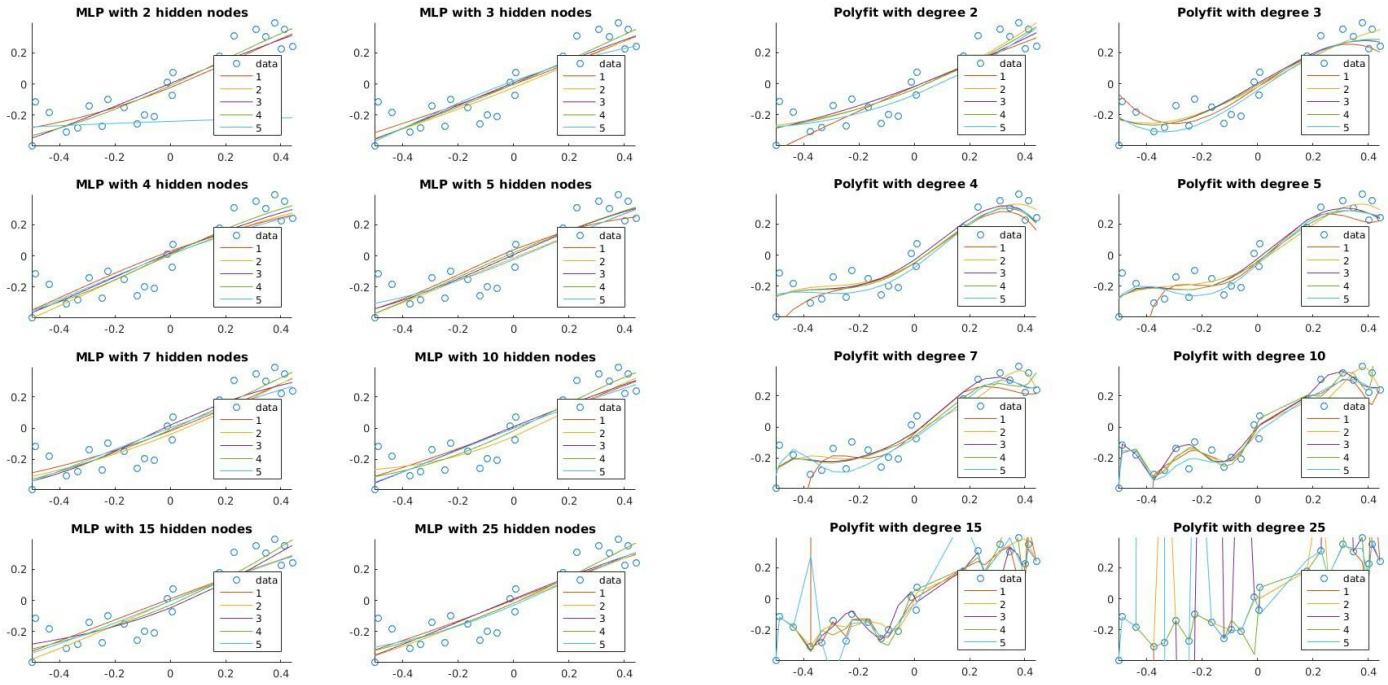


Figure 1. a) Regression lines found by MLP and b) found by polyfit function for line.mat dataset.

For the linearly distributed dataset (supplied as line.mat), a multi-layer perceptron was created with the standard Netlab learning rate. In order to test the influence of the number of hidden nodes we used either 2, 3, 4, 5, 7, 10, 16 or 25 hidden nodes. The solutions found by the MLP

for each of the different number of hidden nodes can be seen in figure 1a. Five different lines are visible because of the five-fold cross validation.

We created a script (named ‘ass3a_mlp.m’) to test an MLP model with different hidden nodes and perform the cross validation. First we did a bit of preprocessing by removing the mean from the dataset and then we shuffled the data and calculated the cross-validation indices (code in ‘ass3a_crossvalid.m’). To actually train the MLP we first instantiated the network with the ‘mlp’ function, then, inside the for-loop that controls the iterations we feed-forward the training examples to get the network output and then we backpropagate the error to obtain the error gradient that we use to update the network weights (proportionally to the learning rate, we used 0.01 for this experiment).

We calculated the root mean squared error for both the test and training set for each of the cross validation runs and here are presented the averages for a varying number of hidden nodes.

HNs	**RMSE train**	**RMSE test**
2	0.157244983671785	0.194202513305488
3	0.100704663022102	**0.109643715727938**
4	0.104428997138703	0.121842051885381
5	0.0983403839134731	0.119972065753754
7	0.100465860579716	0.121037019935402
10	0.0991991931694205	0.123811616182527
15	0.100197308939868	0.121741997420735
25	0.0995317214678983	0.115593952126008

We can seen that with a number of hidden nodes greater than 2, the RMSE doesn’t seem to be affected by the number of hidden nodes.

For comparison, the same data was also fitted with Matlab’s polyfit function. The code that takes care of trying different degrees and performs cross validation can be found in ‘ass3a_poly.m’. Figure 1b shows the model found for different degrees. As can be seen from the picture and from the following table, an higher degree makes the regression line overfit the training data.

Deg	**RMSE train**	**RMSE test**
2	0.0991134445050283	0.122159003810515
3	0.0900729663741482	0.114214333192959
4	0.0848233455256092	**0.110960317257274**
5	0.0827168279351561	0.222846176297594
7	0.0812195749655233	0.266353004473788
10	0.0612661972386335	1.77686529790181
15	0.0457050188659241	633.868416874156
25	0.0168211359644200	42486.9617371826

The best performance on this dataset is gained by using an MLP with three hidden nodes, although differences are small. Generally speaking, the MLP and polyfit algorithm are comparable in terms of performance when selecting sensible polyfit degrees and numbers of hidden neurons.

Next we tried the same experiment with the dataset contained in ‘sinus.mat’. The resulting boundaries can be seen in figure 2.

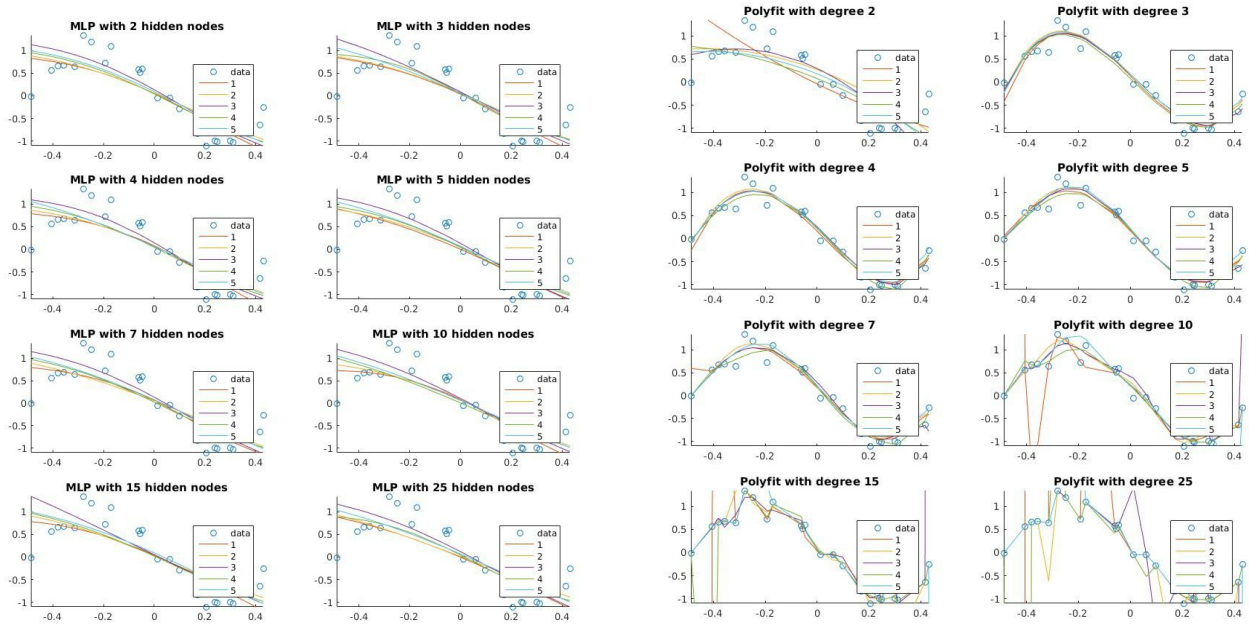


Figure 2. Regression lines found by a) MLP and b) polyfit for dataset sinus.mat

As done for the line.mat dataset, we measured the average RMSE over the cross validation runs on the training and test sets. The following table shows the measurements for the MLP.

NHs	RMSE train	RMSE test
2	0.420937036431322	0.430493652470619
3	0.428396302779223	0.441406667757591
4	0.422229028690075	0.436049309755750
5	0.412926499799509	**0.427550792741538**
7	0.424905153733118	0.440583569263976
10	0.439981958241695	0.448763612211926
15	0.437545571966222	0.438920052385564
25	0.441833885542325	0.450372101709986

We can see that the error stays mostly unchanged but we have a minimal error in both training and test sets with a number of hidden neurons of 5. The following instead is the RMSE for the polyfit function applied to the same data.

Deg	RMSE train	RMSE test
2	0.432187271328455	0.556071096209377
3	0.186701831815967	**0.224125394312804**
4	0.172607405971503	0.227495391589642
5	0.167148860610134	0.240006219711552
7	0.165723685053948	0.250804988562173
10	0.147390418338294	1.52704832616645
15	0.0812870998239886	179.995677423807
25	0.0386534719773166	2887.27639574831

We can observe a behaviour identical to that of the line dataset: for higher degrees the model overfits the training data and produces a huge error in the test set. On this dataset the polyfit algorithm performs significantly better than the MLP. Especially when using an intermediate number of polyfit degrees (between 3 and 7), performance is great. Unexpectedly, the MLP performs almost equally well with two hidden neurons when compared to higher numbers of hidden neurons.

Finally we tested the MLP and polyfit models against the irregular.mat dataset. The resulting boundaries can be seen in figure 3. We used a scaled conjugate gradients optimisation algorithm to improve performance (backpropagation error for this dataset diverge to infinity).

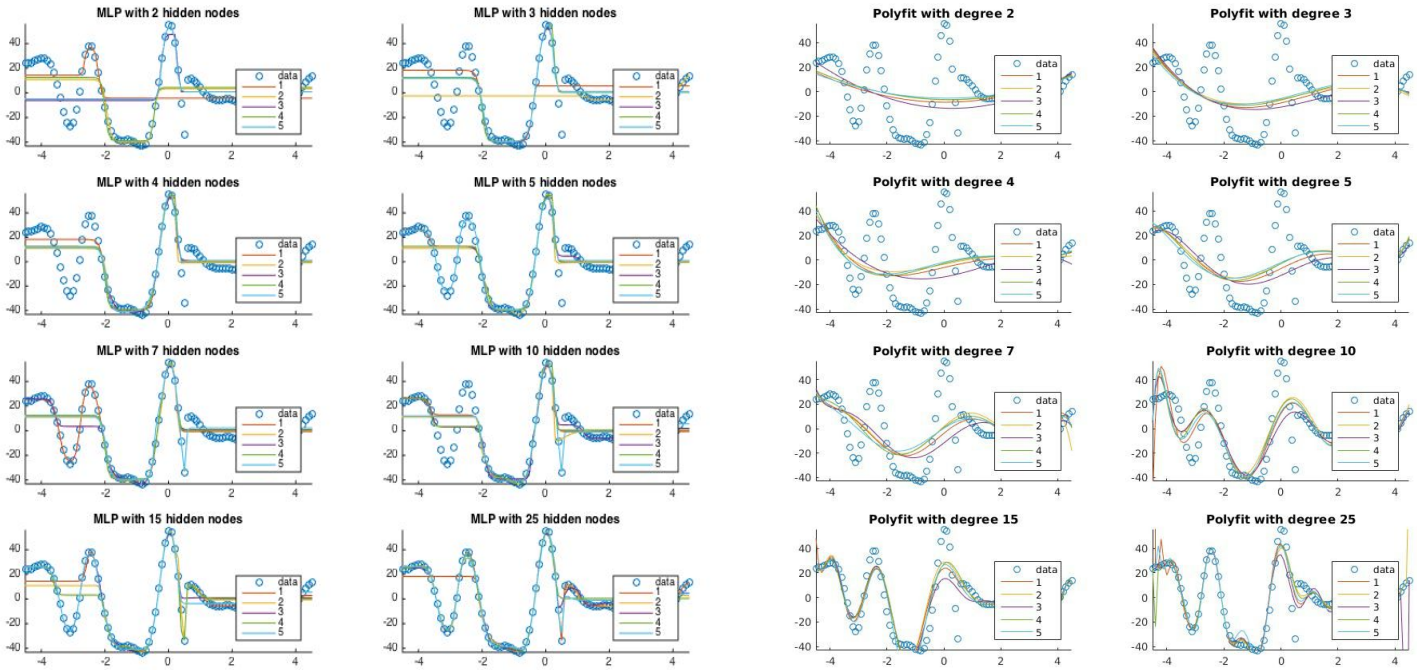


Figure 3. Regression line produced by a) MLP with scg and b) polyfit for irregular.mat dataset

As expected, any number of hidden nodes or polyfit degrees produce poor results on this dataset. This is confirmed by the error rates of both the MLP and the polyfit algorithm:

MLP:

NHs	RMSE train	RMSE test
2	18.5897120966248	19.3982121718174
3	15.0932105148217	15.1023088023445
4	11.9886632904678	12.5413814213187
5	10.6259739337255	11.6917259969272
7	10.6958658921444	9.80504764701466
10	11.0063876872239	11.9632489426111
15	7.96585066358963	10.8664178957092
25	5.24559133027412	**8.13582584563696**

Polyfit:

Deg	RMSE train	RMSE test
2	22.0587378678033	22.0645639258639
3	21.0998798658508	21.5540189173625
4	20.9483265587458	21.6244355556047
5	20.4409363911211	21.1599096435111
7	19.8934710367310	21.2796669098943
10	15.0620255781735	17.5622081895065
15	10.8157016183550	**11.7552206582657**
25	6.26342423611620	59.0248613316108

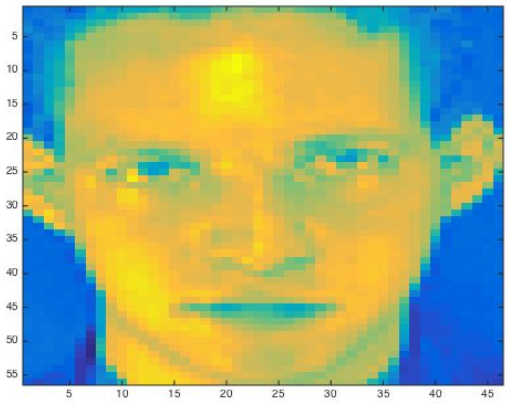
While the MLP performs comparably well to the polyfit algorithm for low numbers of hidden nodes and polyfit degrees, a high number of nodes performs much better than a high number of polyfit degrees. The error rate on the test set of the polyfit algorithm shows its behaviour nicely: Low numbers of degrees perform badly, high numbers of degrees perform badly, but the sweet spot (around 15 degrees in this case) performs quite well. This shows the polyfit algorithm needs a sufficient number of degrees in order to map the data, but a number that is too high will lead to overfitting. The same behaviour does not occur for the MLP. This shows that a higher number of hidden nodes enable the MLP to learn more complex boundaries. While the MLP does not necessarily perform a lot better on this dataset than the polyfit algorithm, it is certainly more robust in terms of parameter settings.

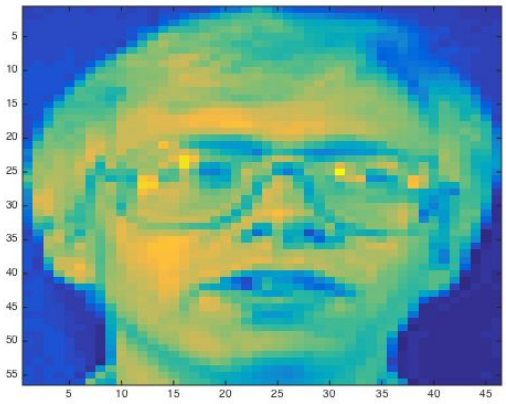
Section B

The main Matlab script for this part of the assignment can be found attached as ‘ass3b_clean.m’.

For this part of the assignment we trained a multi-layer perceptron with 2576 input nodes (the number of pixels in each sample image in the supplied pics.dat), a varying number of hidden nodes and one output node. The goal of this training was to enable the perceptron to classify an image of a face, with the classification criterion being the absence of presence of a pair of glasses on the subject of the image.

The number of hidden nodes is varied between 2, 3, 4, 5, 7, 10, 15, 25, 50, 100, 200 and the number of examples in the 10 folded training set (360). Ten-fold cross validation was used so all observations are used for both training and validation, and each observation is used for validation exactly once. This means the folds can be chosen randomly. A sample image from each of the classes (no glasses, glasses) can be seen below.





To actually train the MLP we again instantiated the network with the ‘mlp’ function, then, inside the for-loop that controls the iterations we feed-forward the training examples to get the network output and then we backpropagate the error to obtain the error gradient that we use to update the network weights (proportionally to the learning rate, we used 0.01 for this experiment).

A total of 400 images is present in the dataset, of which 281 are of people without glasses and 119 are of people with glasses. Since the network has very few ways of ‘understanding’ what the concept of ‘wearing glasses’ would mean for each of the images, we expected performance to be not so great and mostly skewed towards classifying images as ‘not glasses’, because those examples are overrepresented in the dataset. This expectation is confirmed by the percentage of correctly classified images:

Accuracy for 2 hidden nodes: 0.5
Accuracy for 3 hidden nodes: 0.6215
Accuracy for 4 hidden nodes: 0.5405
Accuracy for 5 hidden nodes: 0.6215
Accuracy for 7 hidden nodes: 0.662
Accuracy for 10 hidden nodes: 0.6215
Accuracy for 15 hidden nodes: 0.6215
Accuracy for 25 hidden nodes: 0.5405
Accuracy for 50 hidden nodes: 0.581
Accuracy for 100 hidden nodes: 0.581
Accuracy for 200 hidden nodes: 0.6215
Accuracy for 360 hidden nodes: 0.581

While the classifications are generally higher than chance level, they are not practically useful. When looking at the confusion matrices for a few of the more extreme accuracy rates, our earlier hypothesis was confirmed:

2 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	47.6	112.4
	No glasses	71.4	168.6

The perceptron classifies most images that actually contain glasses as having no glasses, and classifies the same percentage wrong in the ‘no glasses’ class. This shows it does not actually learn any features, but instead simply learns ratios of classes present in the training set. Considering the fact that only two hidden neurons are present, no better behaviour could be expected on a relatively complicated dataset. Decimals are due to taking the average of every fold in the ten-fold cross validation.

7 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	11.9	28.1
	No glasses	107.1	252.9

This time, the perceptron seems learns something slightly more complicated. It learns that most images do not contain glasses. Still, the ratio between predicted glasses that are actually glasses and predicted glasses that are not actually glasses favors the incorrect option and the majority of the correct calls comes from ‘not glasses’ being correctly classified.

200 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	23.8	56.2
	No glasses	95.2	224.8

The perceptron shows practically the same behaviour as in the case of seven hidden nodes, yet it classifies more samples as containing glasses.

As the perceptron performs rather poorly on all cases and it's reasonable performance is mostly due to the overrepresentation of 'not glasses' cases in the dataset, we feel the data must be simplified and the most informative features have to be extracted before the perceptron can make any meaningful predictions. A higher learning rate did not have any meaningful influence on the results.

Feature extraction is done applying a Gabor filter on the images before they are fed to the perceptron for training. A Gabor filter is a linear filter used for edge detection. Frequency and orientation representations of Gabor filters are similar to those of the human visual system, and they have been found to be particularly appropriate for texture representation and discrimination. This filter reduces dimensionality of the images, while increasing the information present in the inputs. The perceptron no longer needs to work with just colour values of the images, but is now supplied with things such as orientation and spatial frequency.

Because of this, we expected a performance increase for the perceptron after processing the images with a Gabor filter.

Accuracy for 2 hidden nodes: 0.5715
Accuracy for 3 hidden nodes: 0.6265
Accuracy for 4 hidden nodes: 0.66875
Accuracy for 5 hidden nodes: 0.70925
Accuracy for 7 hidden nodes: 0.69775
Accuracy for 10 hidden nodes: 0.586
Accuracy for 15 hidden nodes: 0.7535
Accuracy for 25 hidden nodes: 0.70125
Accuracy for 50 hidden nodes: 0.69525
Accuracy for 100 hidden nodes: 0.7835
Accuracy for 200 hidden nodes: 0.7665
Accuracy for 360 hidden nodes: 0.82225

Clearly performance increases when using the Gabor filter, up to a correctly classified percentage of 82.2% for 360 hidden nodes. Also, this shows quite clearly that for data with high dimensionality, an MLP with a high number of hidden nodes performs better due to being able to model the data better.

The confusion matrices of 2, 7 and 360 hidden nodes can be found below for comparison to the MLP without Gabor filtering. Again, higher learning rates do not increase performance.

2 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	48.0	100.4
	No glasses	71.0	180.6

Interestingly enough, the perceptron classifies quite a few cases as ‘glasses’, even though they are not actually containing glasses. While the error rate is not much better than the MLP without Gabor filter (57.2% versus 50%), the perceptron is more eager to classify cases as ‘glasses’.

7 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	49.4	51.3
	No glasses	69.6	229.7

Already performing better than any of the MLP’s without Gabor filter, this 7 hidden node multi-layer perceptron has a hard time distinguishing glasses, but is fairly good at classifying images without glasses as such.

360 hidden nodes	Actual class		
Predicted class		Glasses	No glasses
	Glasses	57.8	9.9
	No glasses	61.2	271.1

With an accuracy rate of of 82.2%, this MLP with 360 hidden neurons performs best out of any of the settings we tried. It is excellent at classifying images without glasses as such, and performs reasonably well on images with glasses. We suspect this is due to the generally small frames of the glasses worn in the pictures, making them not very obvious features.

We can easily conclude that by means of a Gabor filter we were able to maximize the information content of the training examples fed to the network and so improve its performance.