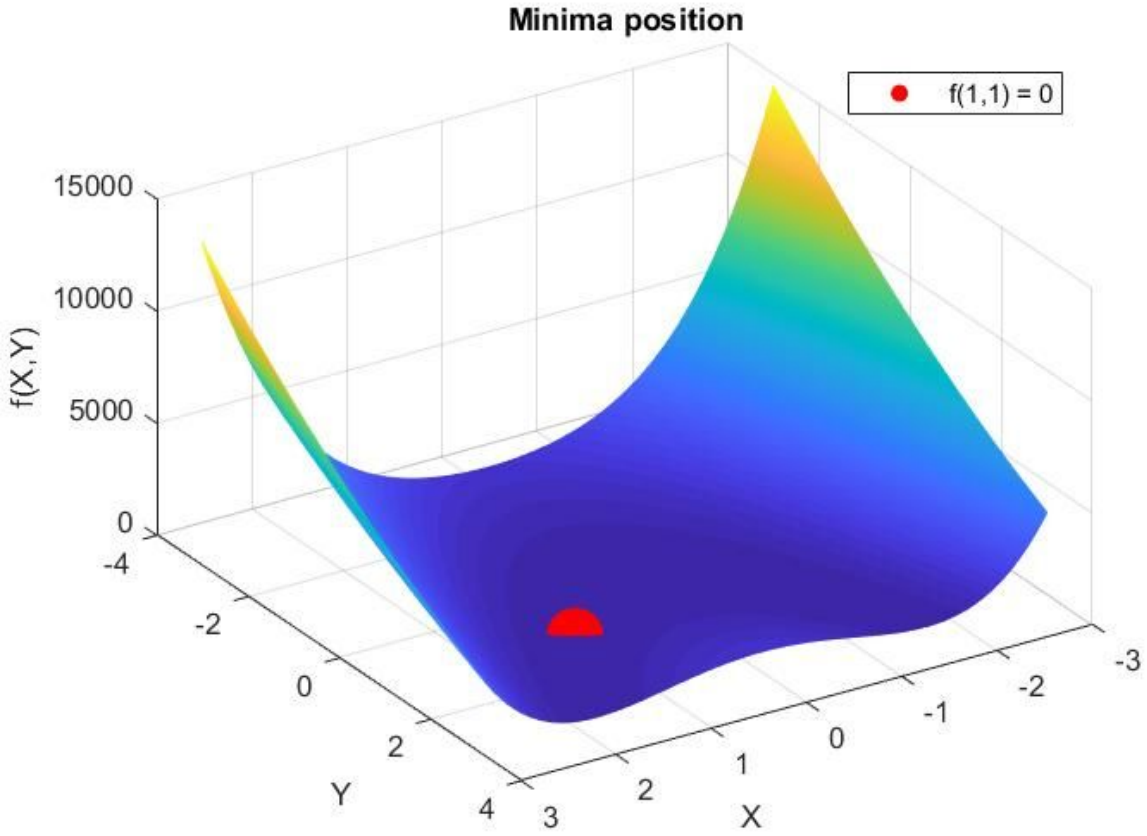
NN coursework 2: Brian Formento: A0196190Y

Rosenbrock's Valley Problem

a) Using Matlab appendix 1

$$f(1,1) = (1-1)^2 + 100(1-1^2)^2 = 0$$



b) Using Python appendix 2

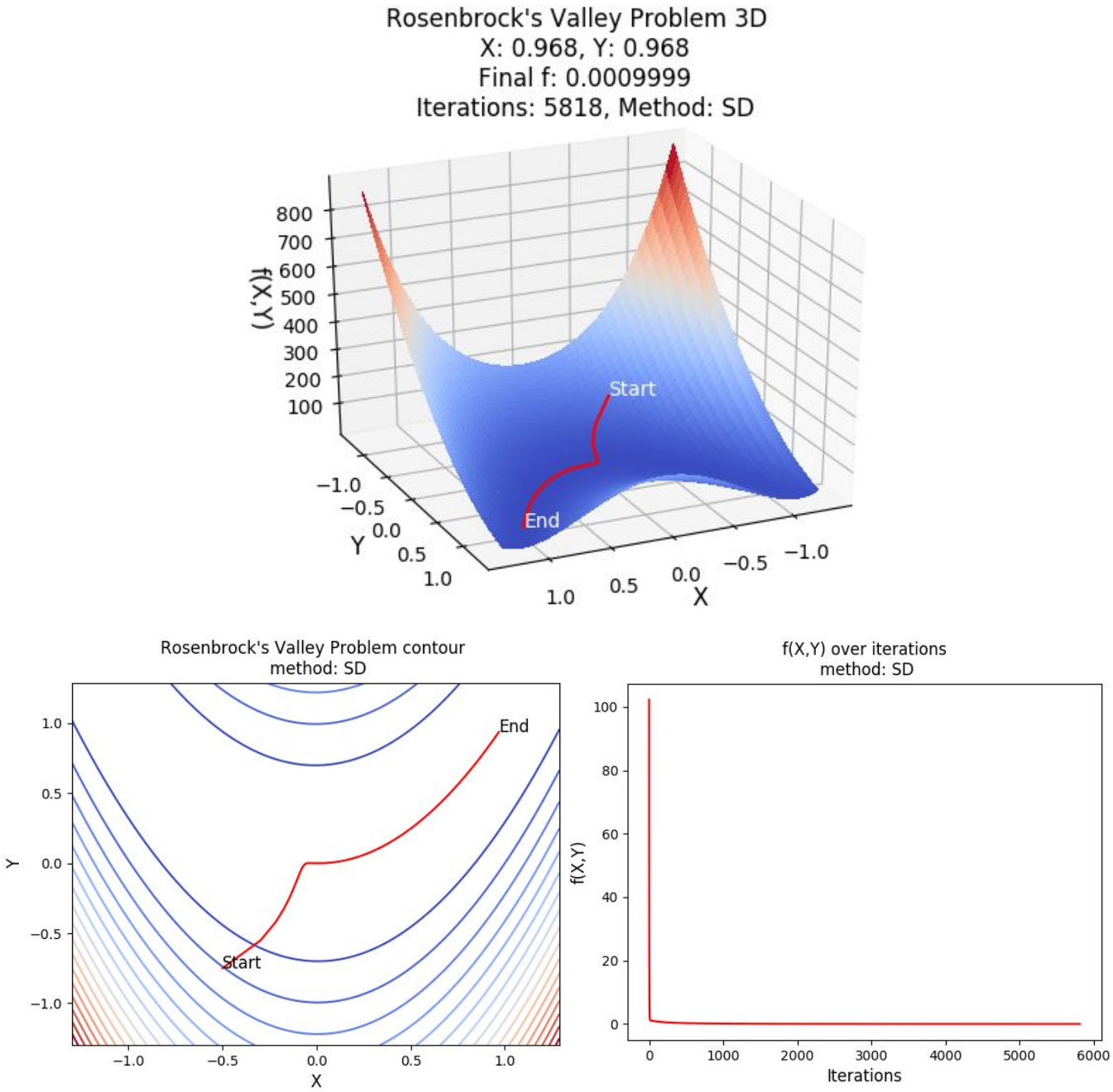
We need to calculate the partial derivatives of f(x,y).

$$\frac{df}{dx} = 2(200x^3 - 200xy + x - 1)$$
$$\frac{df}{dy} = 200(y - x^2)$$

Using the above derivatives we can update both x and y at each iteration using

$$w(k+1) = w(k) - \eta g(k)$$

Using the gradient_descend function in the program in appendix 2 we can generate the following graphs, where the starting condition for x and y was picked randomly at [-0.5,-0.75]:



c)

For the Newton method:

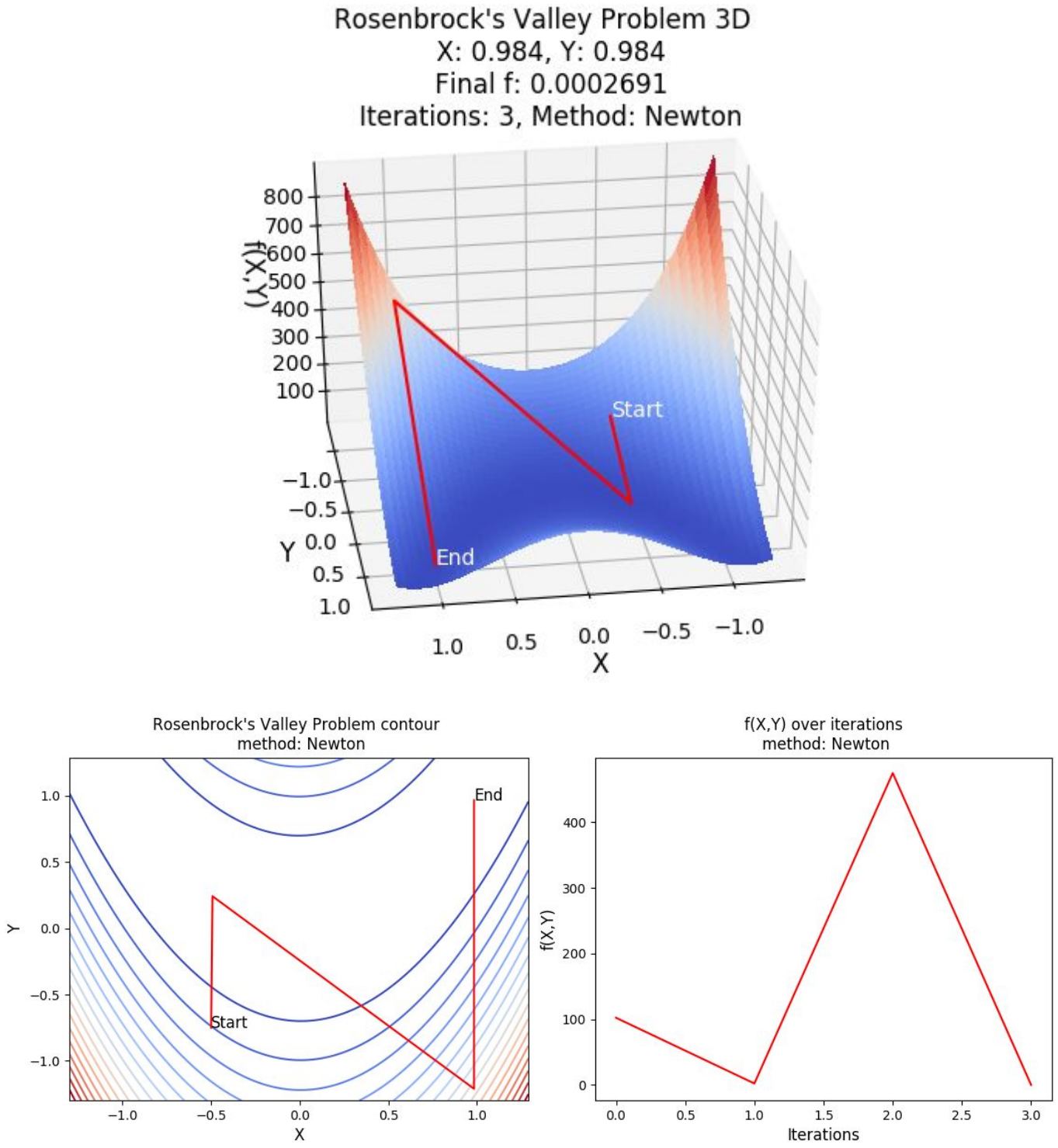
$$w(k+1) = w(k) - H^{-1}(k)g(k)$$

We need a Hessian matrix, which is composed of the second order derivatives of f

$$\frac{d^2f}{dx^2} = 1200x^2 - 400y + 2 \quad \frac{d^2f}{dxy} = -400x$$

$$\frac{d^2f}{dyx} = -400x \quad \frac{d^2f}{dy^2} = 200$$

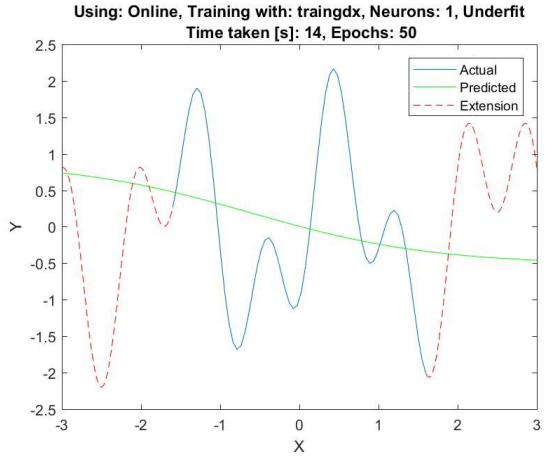
Using the Hessian matrix and g the gradient vector which holds the x and y derivatives the update response can be calculated. This is shown in the Newton Function in appendix 2.

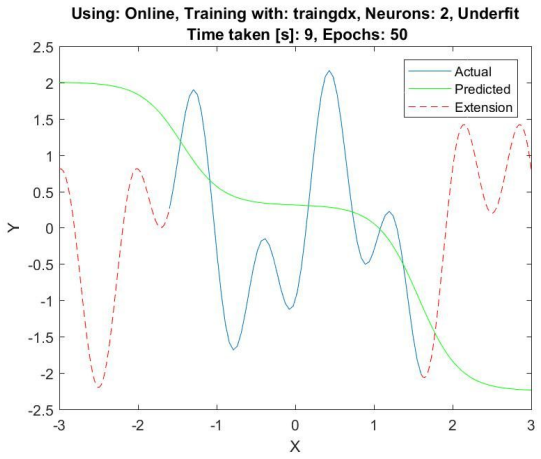


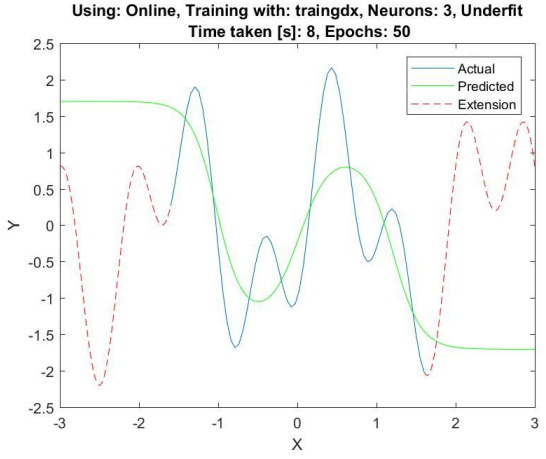
Function Approximation

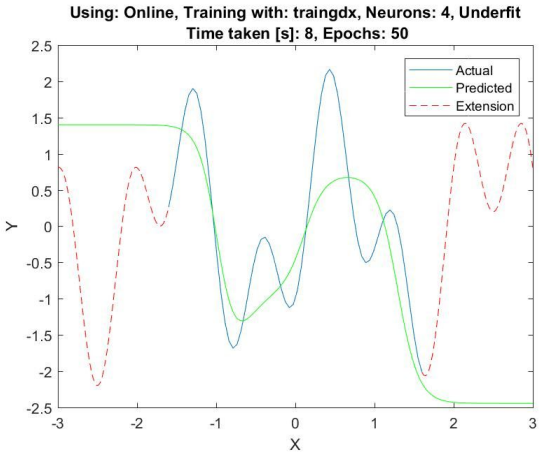
a) Using Matlab appendix 3A

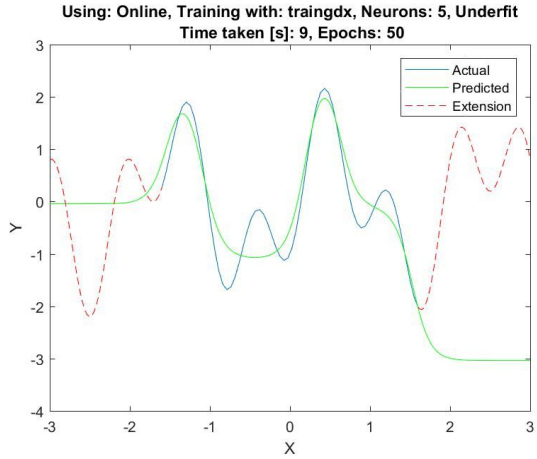
For sequential training the cell datatype has to be used. To simulate the function between [-1.6, 1.6] with a training interval of 0.05 between points 64 points are needed. While to simulate with and testing interval of 0.01 320 points are needed. For testing, to extend this between [-3,3] 600 points are needed. It has been found that 20 or 30 output the best result, afterwhich the output quickly degrades. 20 neurons has been therefore labeled as proper fit.

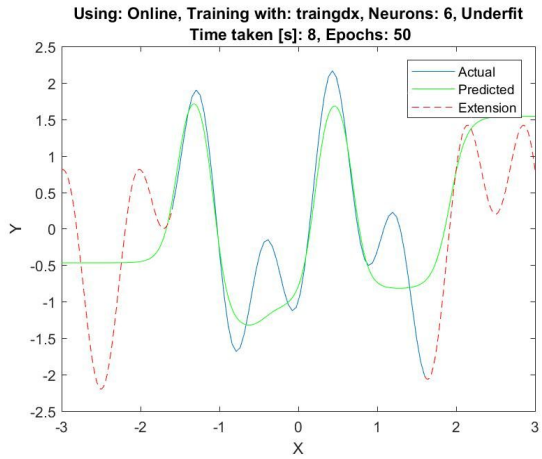

Using: Online, Training with: traingdx, Neurons: 1, Underfit
Time taken [s]: 14, Epochs: 50
Actual
Predicted
Extension
X
Y

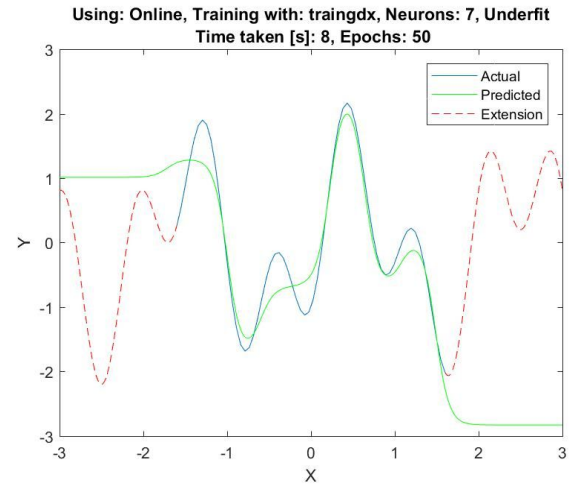

Using: Online, Training with: traingdx, Neurons: 2, Underfit
Time taken [s]: 9, Epochs: 50
Actual
Predicted
Extension
X
Y

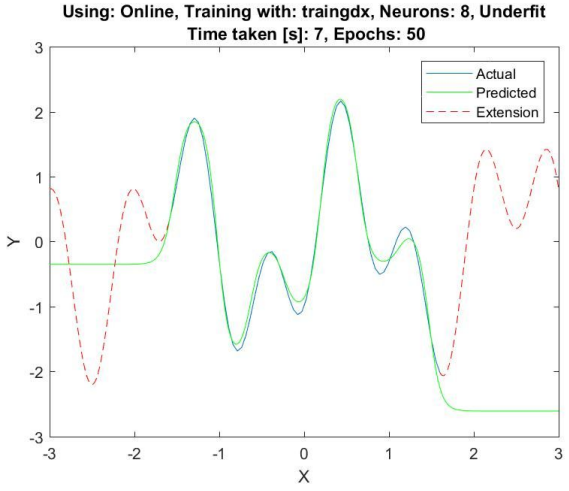

Using: Online, Training with: traingdx, Neurons: 3, Underfit
Time taken [s]: 8, Epochs: 50
Actual
Predicted
Extension
X
Y

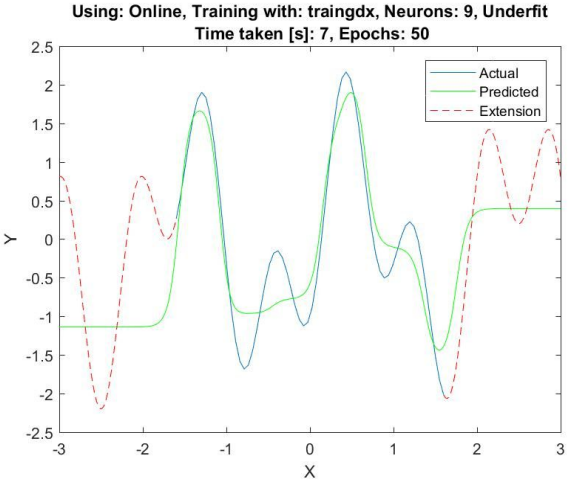

Using: Online, Training with: traingdx, Neurons: 4, Underfit
Time taken [s]: 8, Epochs: 50
Actual
Predicted
Extension
X
Y

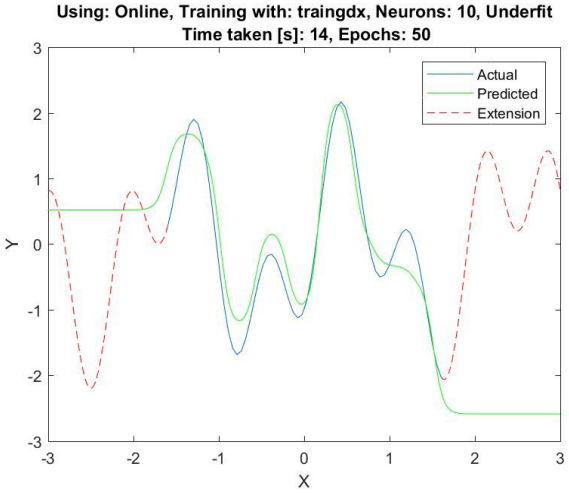

Using: Online, Training with: traingdx, Neurons: 5, Underfit
Time taken [s]: 9, Epochs: 50
Actual
Predicted
Extension
X
Y

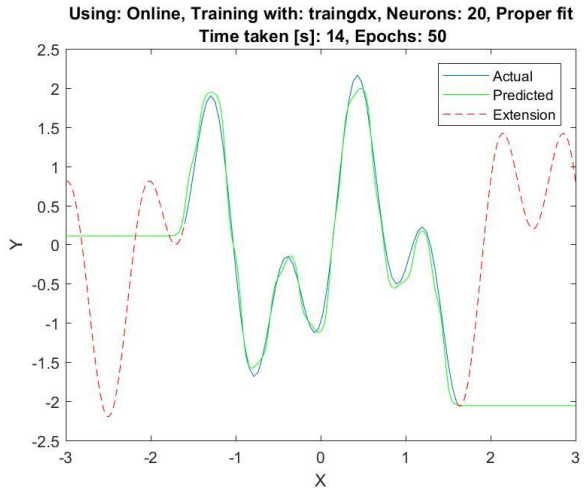

Using: Online, Training with: traingdx, Neurons: 6, Underfit
Time taken [s]: 8, Epochs: 50
Actual
Predicted
Extension
X
Y

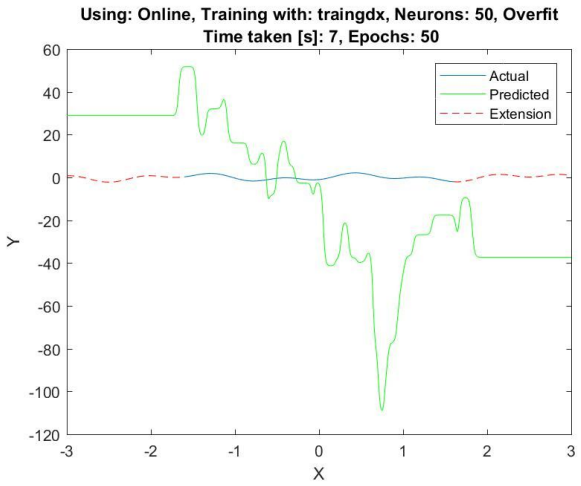

Using: Online, Training with: traingdx, Neurons: 7, Underfit
Time taken [s]: 8, Epochs: 50
Actual
Predicted
Extension
X
Y

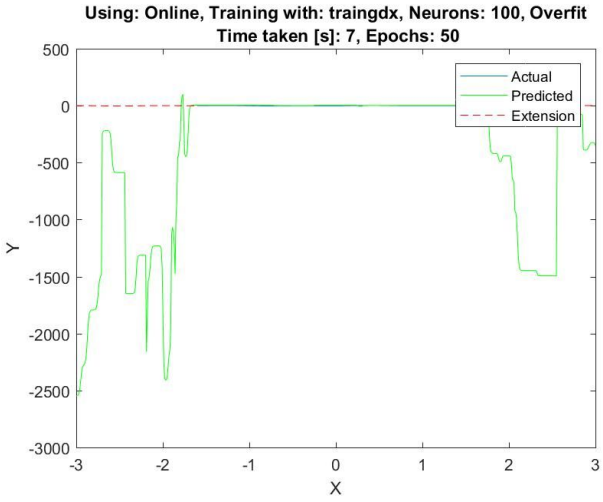

Using: Online, Training with: traingdx, Neurons: 8, Underfit
Time taken [s]: 7, Epochs: 50
Actual
Predicted
Extension
X
Y


Using: Online, Training with: traingdx, Neurons: 9, Underfit
Time taken [s]: 7, Epochs: 50
Actual
Predicted
Extension
X
Y


Using: Online, Training with: traingdx, Neurons: 10, Underfit
Time taken [s]: 14, Epochs: 50
Actual
Predicted
Extension
X
Y

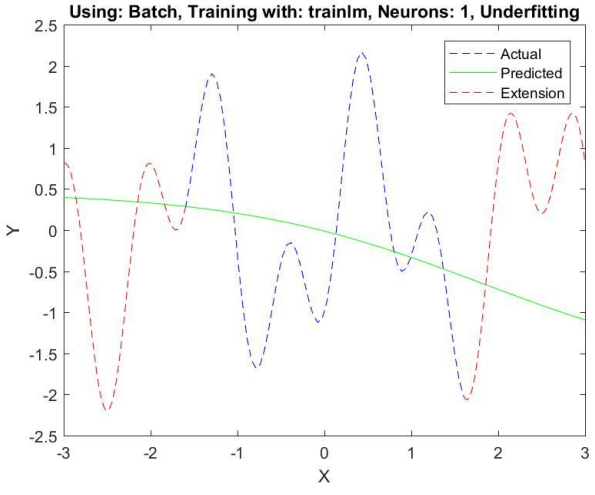


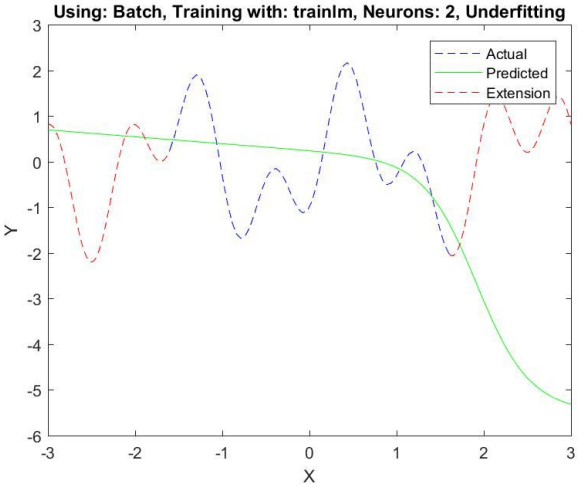


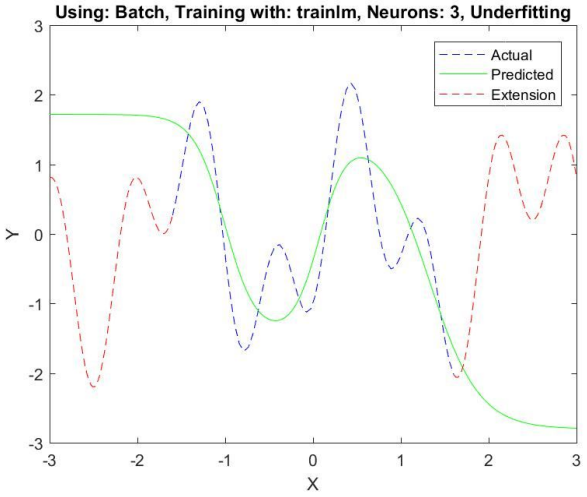


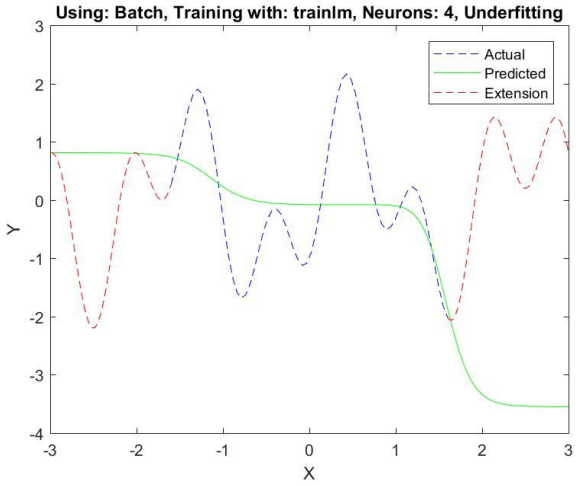
b) Appendix 3B

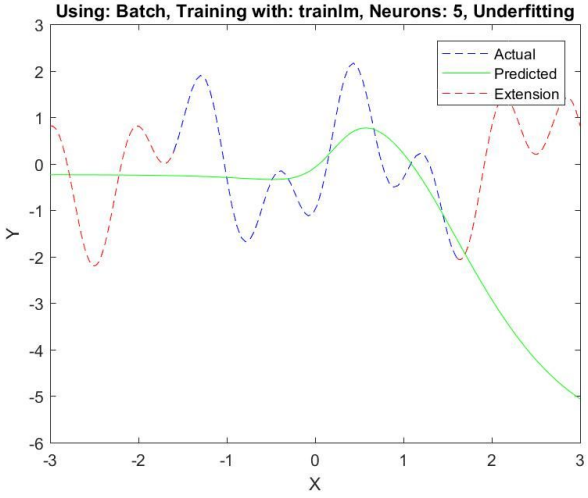
Using trainlm: Again, just as with using the sequential method not perfect solution exists, however with 20 neutrons it gives a very good representation and it has been therefore been labeled as proper fit, however it is more stable when using above 20 neurons.

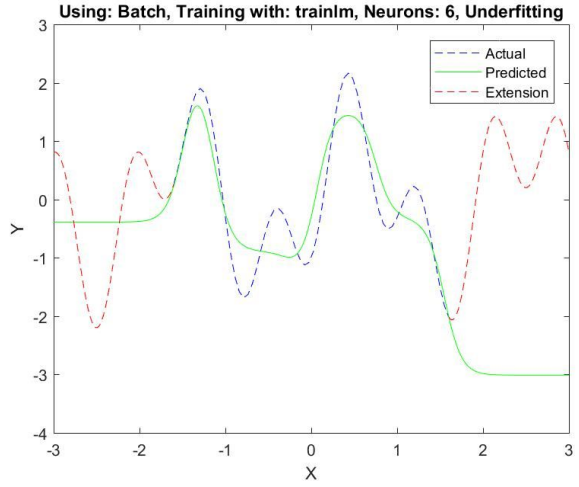


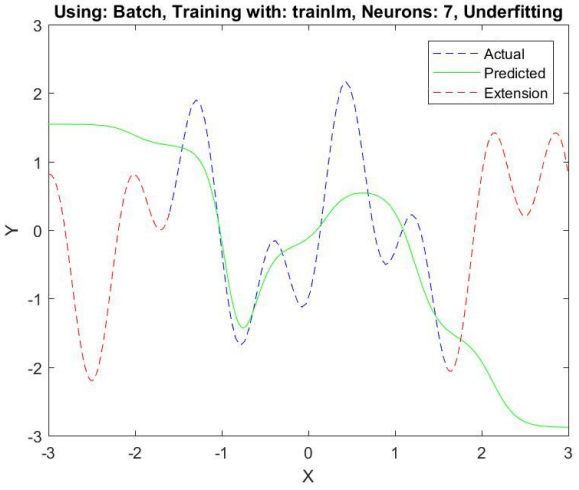


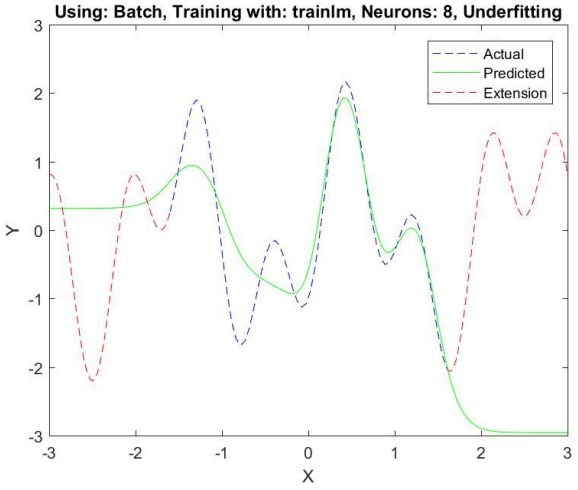

Using: Batch, Training with: trainlm, Neurons: 3, Underfitting
Actual
Predicted
Extension
X
Y

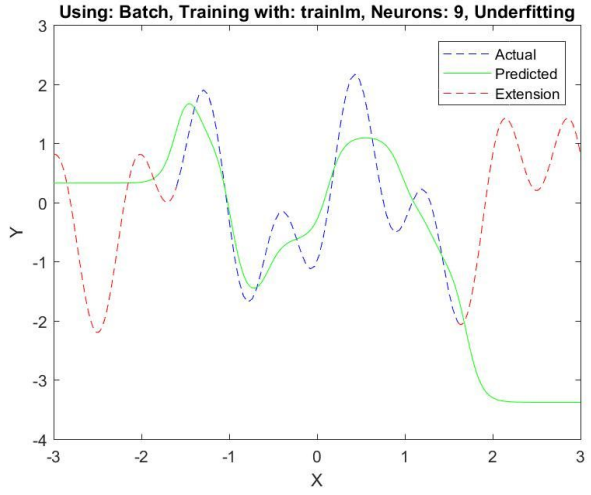

Using: Batch, Training with: trainlm, Neurons: 4, Underfitting
Actual
Predicted
Extension
X
Y

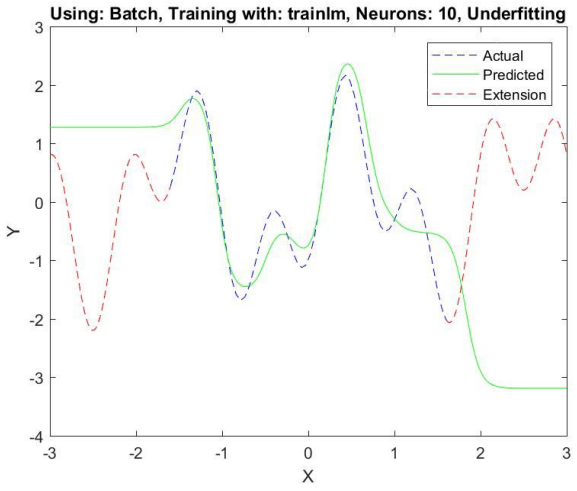

Using: Batch, Training with: trainlm, Neurons: 5, Underfitting
Actual
Predicted
Extension
X
Y

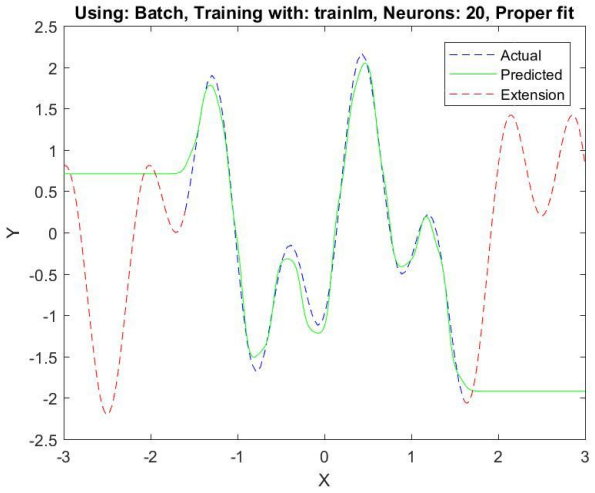

Using: Batch, Training with: trainlm, Neurons: 6, Underfitting
Actual
Predicted
Extension
X
Y

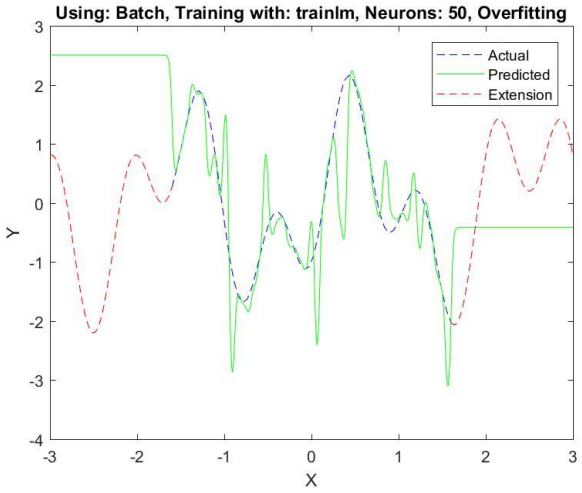

Using: Batch, Training with: trainlm, Neurons: 7, Underfitting
Actual
Predicted
Extension
X
Y

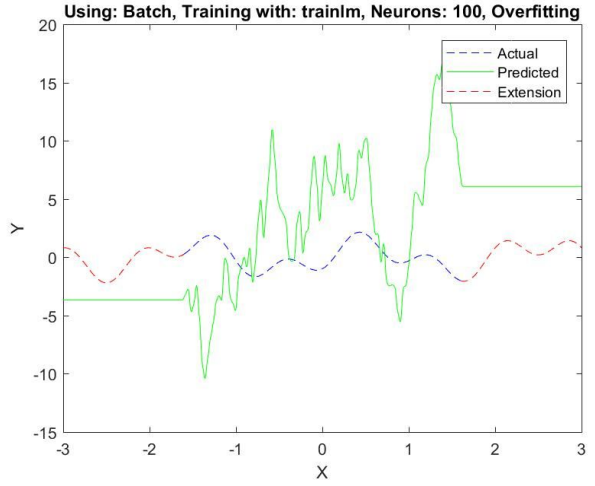

Using: Batch, Training with: trainlm, Neurons: 8, Underfitting
Actual
Predicted
Extension
X
Y





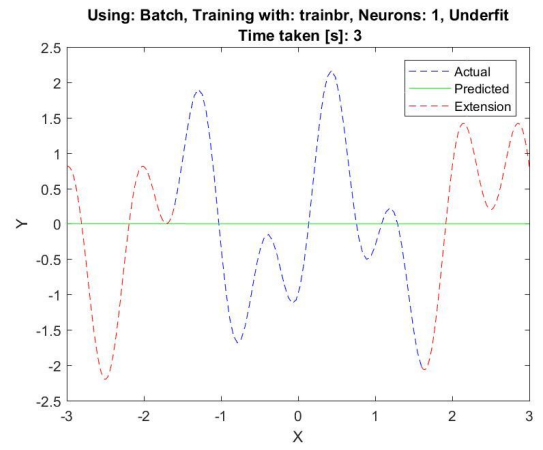


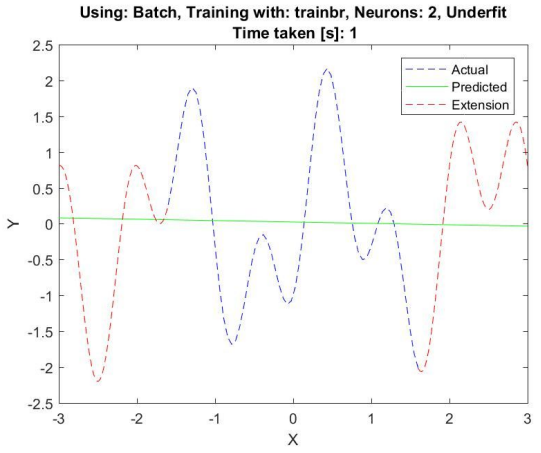


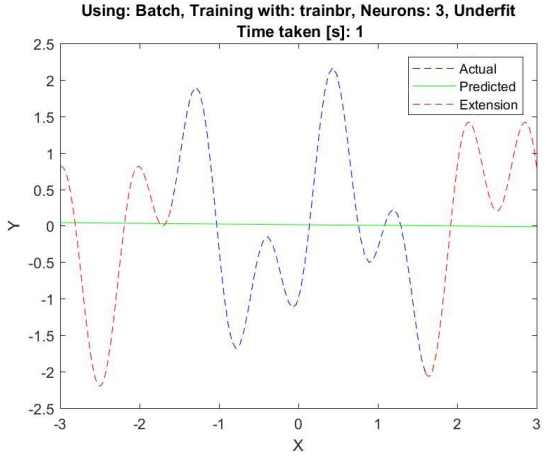


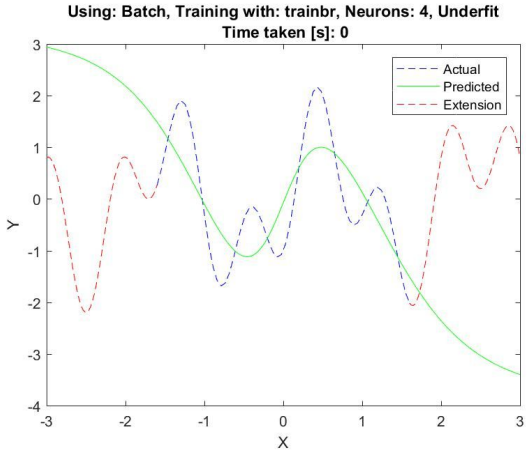
c) Appendix 3B

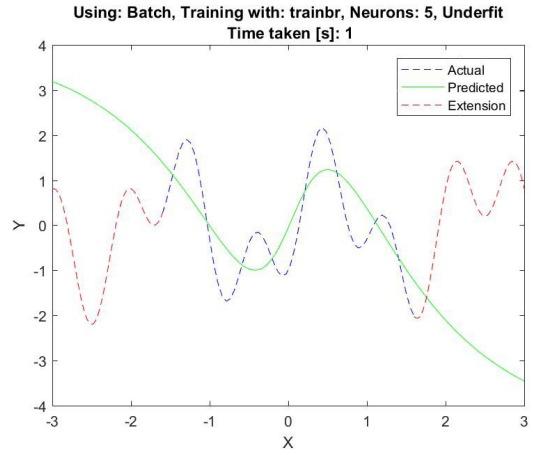
Using trainbr, no best solution has been found. It has also been found that with only a few neurons the networks struggles to generalise, Furthermore it is the most stable network when overfitting.

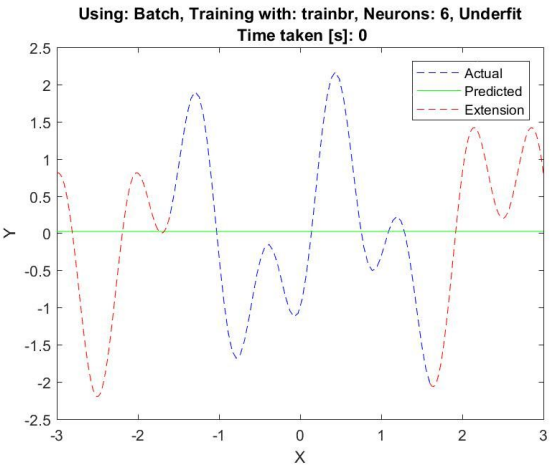

Using: Batch, Training with: trainbr, Neurons: 1, Underfit
Time taken [s]: 3
Actual
Predicted
Extension
X
Y

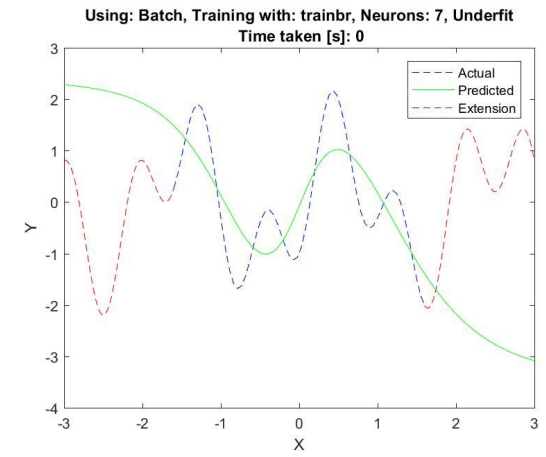

Using: Batch, Training with: trainbr, Neurons: 2, Underfit
Time taken [s]: 1
Actual
Predicted
Extension
X
Y

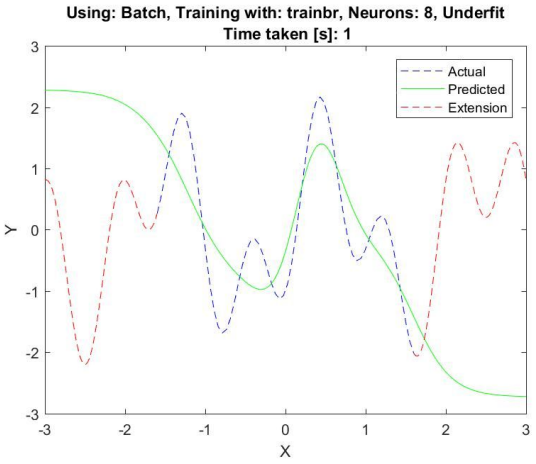

Using: Batch, Training with: trainbr, Neurons: 3, Underfit
Time taken [s]: 1
Actual
Predicted
Extension
X
Y

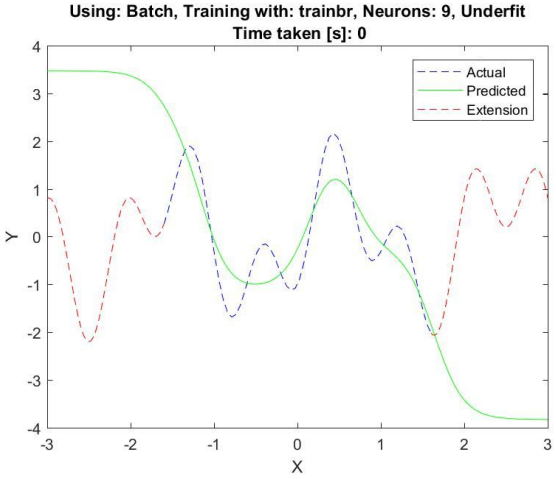

Using: Batch, Training with: trainbr, Neurons: 4, Underfit
Time taken [s]: 0
Actual
Predicted
Extension
X
Y

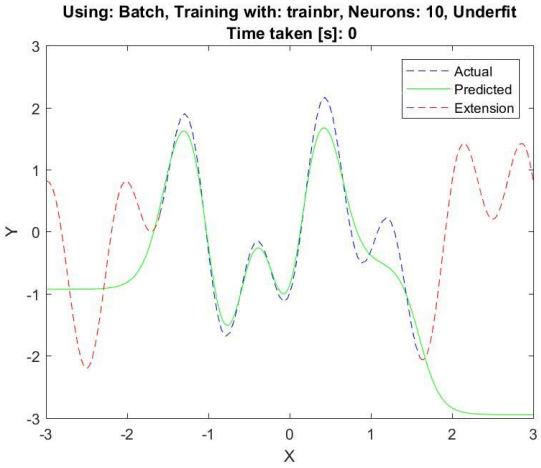

Using: Batch, Training with: trainbr, Neurons: 5, Underfit
Time taken [s]: 1
Actual
Predicted
Extension
X
Y

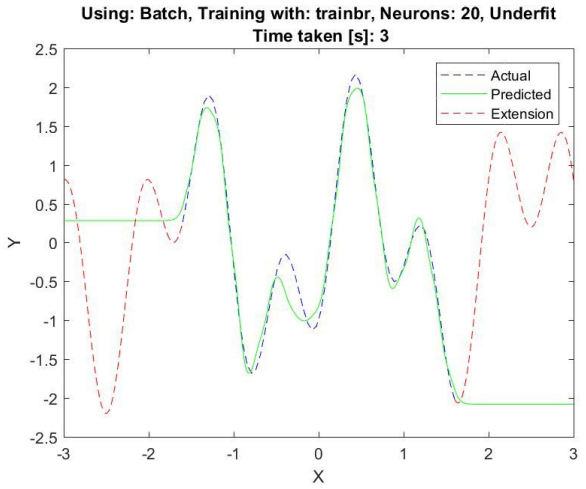

Using: Batch, Training with: trainbr, Neurons: 6, Underfit
Time taken [s]: 0
Actual
Predicted
Extension
X
Y

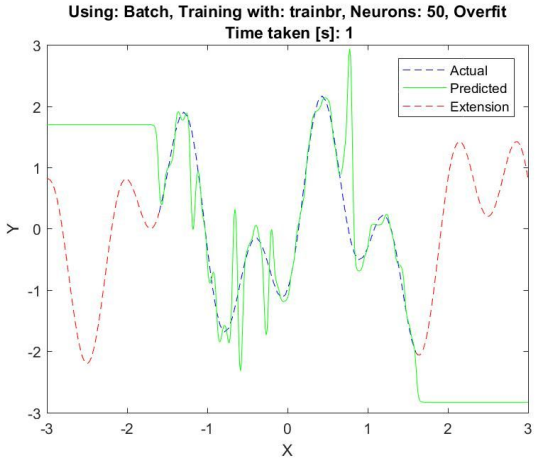

Using: Batch, Training with: trainbr, Neurons: 7, Underfit
Time taken [s]: 0
Actual
Predicted
Extension
X
Y


Using: Batch, Training with: trainbr, Neurons: 8, Underfit
Time taken [s]: 1
Actual
Predicted
Extension
X
Y


Using: Batch, Training with: trainbr, Neurons: 9, Underfit
Time taken [s]: 0
Actual
Predicted
Extension
X
Y


Using: Batch, Training with: trainbr, Neurons: 10, Underfit
Time taken [s]: 0
Actual
Predicted
Extension
X
Y





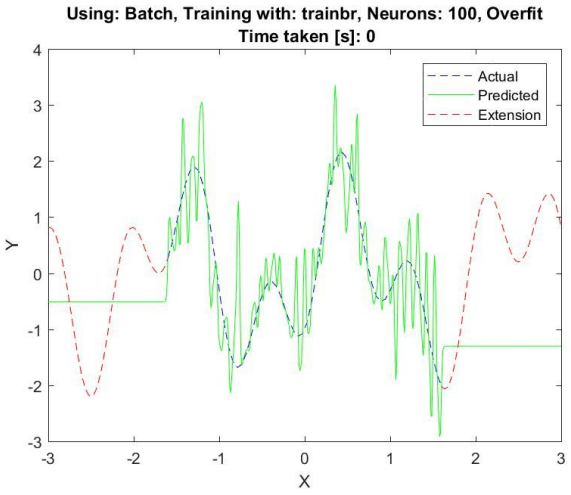


Image Classification

a) Code in appendix 4, written in python. ID: A0196190Y. mod(90, 3) = 0. Classified between airplanes and cats.

For the single layer perceptron use the Perceptron call from sklearn. This call is shown in the Single_perceptron function. The Perceptron uses the fit function call to train the network.

When using a stopping criterion of $tol = 0.003$ where $loss > previous\ loss - tol$
Train score: 0.653, Validation score: 0.62

With 8 epochs:
Train score: 0.557, Validation score: 0.5 (Underfit)

30 epochs:
Train score: 0.721, Validation score: 0.63 (Overfit)

100 epochs
Train score: 0.863, Validation score: 0.6 (Overfit)

Overfitting occurs when the difference between the validation score and the train score becomes substantial. In this case 30 epochs is still good but it's starting to overfit. Fewer epochs would be better.

b)
By taking the global mean and variance more generalization can be achieved quicker with better results. This time a good result is achieved with only 8 epochs. Above this the model starts to overfit.

8 epochs
Train score: 0.804444444444, Validation score: 0.66

30 epochs
Train score: 0.798888888889, Validation score: 0.61

100 epochs
Train score: 0.934444444444, Validation score: 0.6

The results are better than a) for less epochs while being worst for more epochs and the same when its overfitting. This is because subtracting the global mean and variance is helping the network generalise.

c)
Using the whole dataset to train one epoch with a network that has 100 hidden neurons.

30 epoch
Train score: 0.714444444444 Validation score: 0.69

100 epoch
Train score: 0.763, Validation score: 0.67

200 epoch
Train score: 0.808888888889, Validation score: 0.7

d)
When using $loss > previous\ loss - tol$ the best amount of epochs is automatically used before overfitting can be expected as a problem. With the MLPClassifier function this happens at the 31st epoch.

31 epoch
Train score: 0.717, Validation score: 0.7

Using l2 regularisation at $l2 = 0.1$:

200 epoch
Train score: 0.791111111111, Validation score: 0.71

Clearly using regularisation outputted a slightly better result, however more importantly it also reduced the difference between the train and validation score, from 0.108 in c (200) to 0.081 in f (200). This means it's helping the model not overfit.

e)
Using the sequential the training is slower, however the results are slightly better

30 epochs
Train score: 0.727, Validation score: 0.71

100 epochs
Train score: 0.824, Validation score: 0.71

200 epochs
Train score: 0.88, Validation score: 0.7

The best result has been achieved using $l2 = 0.1$ and 30 epochs with sequential learning
Train score: 0.721111111111, Validation score: 0.73.

Recommendations
Seeing the two above approaches it seems sequential is superior and therefore should be used over batch mode. This can be attributed to the nature of the training process as using sequential means updating the weights of the model at every iteration instead of over the whole data set. This means that the process is more noisy and more prone to jumping over local minimas. However, this comes at a disadvantage, as when the global minima is found sequential will never find the 0 energy point. Using batch on the other hand will.

f)
Using mini-batches - This takes the noisy nature of sequential and merges it to the speed of batch mode. Overall its less noisy but still serves the purpose well. It's now one of the most popular methods to train a network. It just divide the dataset up in bins of 2^n to optimise for memory.

Better optimisers - The used method in d and c was stochastic gradient descent, alternatives (with momentum) exist. These methods, such as adam, try and calculate an optimal learning rate for the job.

Data augmentation -This means taking the original dataset and extending it by doing simple changes to each image, such as shear, chopping, rescaling and flipping.

Other regularization techniques - L2 is not the only regularizer, L1 also exists and might be more suitable for this task. These methods have one job and it’s to penalise complex modes.