MACHINE LEARNING (T-MACHLE)

Practical work 09 - Artificial Neural Networks (ANN)

STUDENT: ***ROMAIN CLARET***
PROFESSOR: ***A. PEREZ-URIBE & J. HENNEBERT***
ASSISTANT: ***H. SATIZABAL***
DUE-DATE: ***MONDAY 26 NOVEMBER 2018***

---

1. THE PERCEPTRON AND THE DELTA RULE

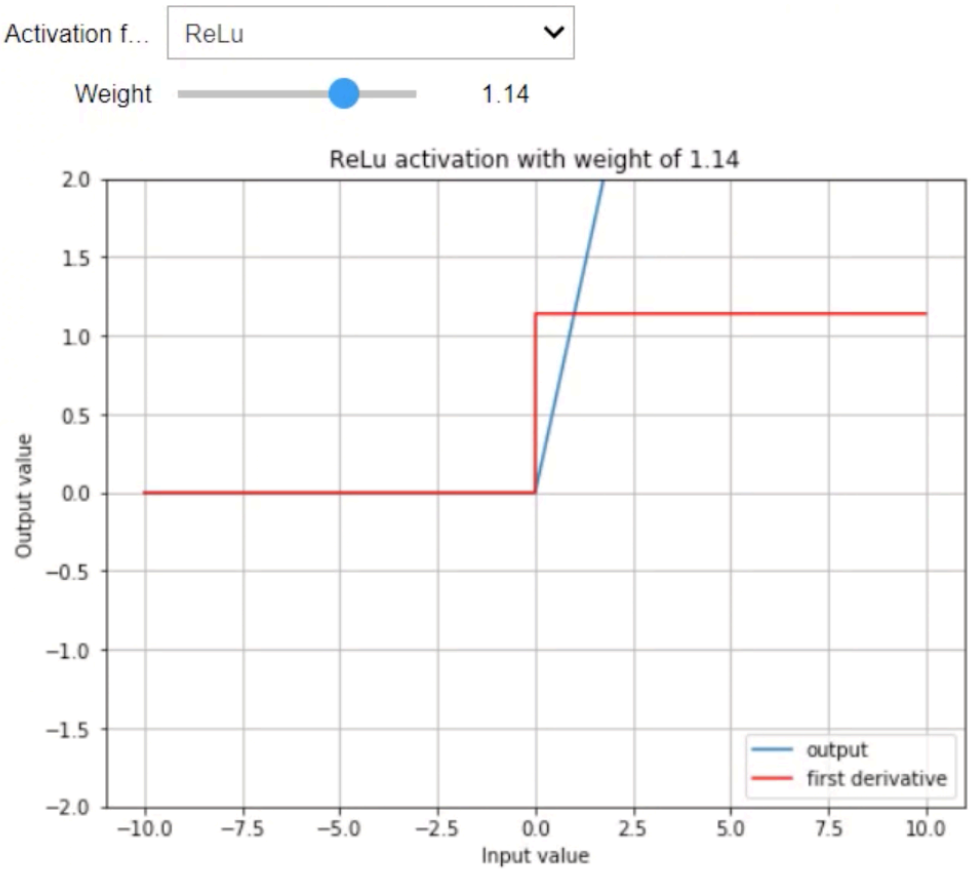
1_activation_function.ipynb

- Observe the shape of the different activation functions proposed.
- Observe the effects of modifying the weight. How the shape of the function changes? How the first derivative changes?
  - **Linear**: the weight changes the output slope and keeps the first derivative on a constant y.
  - **Sigmoid**: the weight increment/decrement affect the sigmoid formation until a weight of –1/1 and defines the strength of the first derivative at x=0.
  - **Hyperbolic tangent**: similar effect than the sigmoid, expect the weight has a stronger effect on the first derivative at x=0.
  - **ReLu**: The weight defines the constant activation threshold of the first derivate

- Implement the activation function of a rectified Linear Unit (ReLu)

```
def relu(neta):
    output = np.copy(neta)
    output[neta<0] = 0
    d_output = np.ones(len(neta))
    d_output[neta<0] = 0
    return (output, d_output)
```

- Visualize the ReLu activation function using the tools given in this notebook



2_perceptron.ipynb

- Use the sliders to change the weights of the perceptron and observe the effects on its output
- Select different activation functions and observe the output of the perceptron for different weight configurations
  - I am interpreting this phenomenon as a “flattering”(if that’s the right word). Indeed, starting with the **linear** activation, I can see hard (pixelized) breaks, then moving to the **Hyperbolic tangent** is trying to harmonize the break, then the **Sigmoid** is showing very fine and harmonized continuous breaks.

3_MLP.ipynb

- Use the sliders to change the values of the connection weights and biases, and

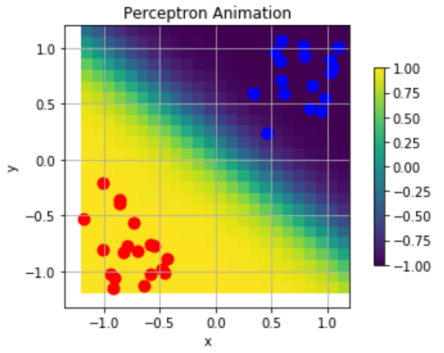
observe the resulting changes in the MLP output

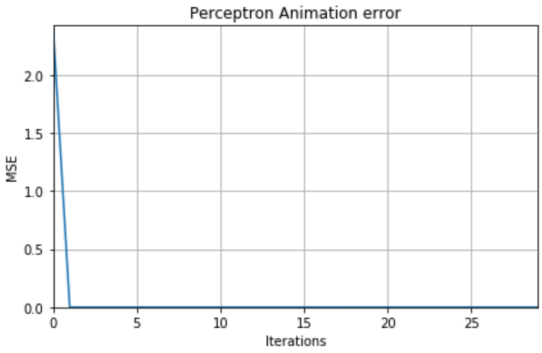
- Change the activation function and observe the changes
  - I am pretty confused about the outputs here. I found how to create rotations of the activation thresholds, but I am not sure that is what I was supposed to see.

4_delta_rule.ipynb

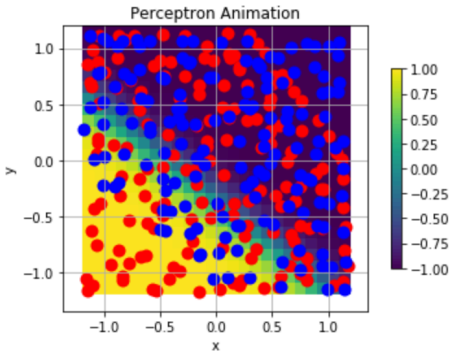
Create some datasets as it is shown, and perform the following tests:

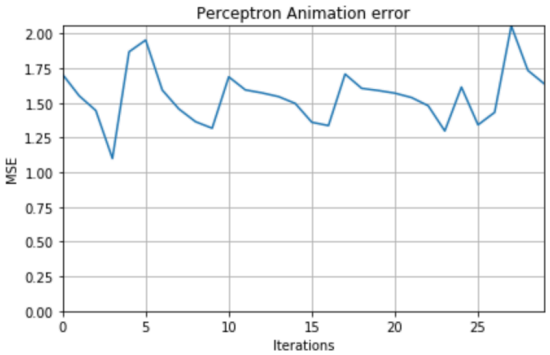
- What happens if the boundaries between both classes are well defined?
  - The error is flooring at zero very fast, and the frontier is well defined.





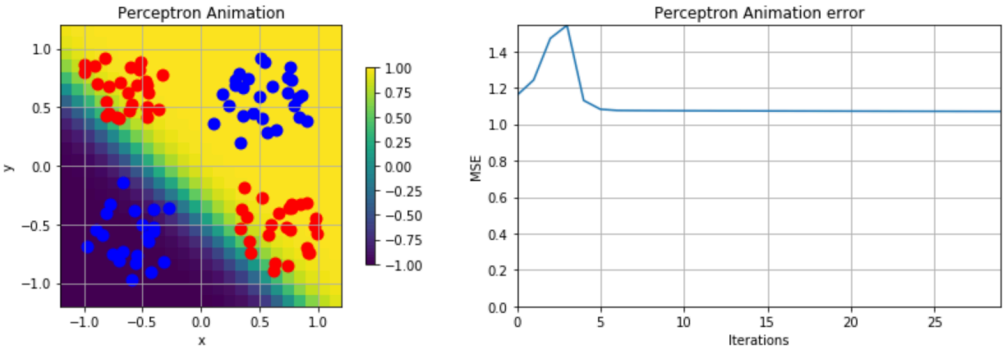
- What happens if the classes overlap? What could you say about oscillations in the error signal?
  - The error is oscillating, and the frontier is that well defined. The oscillation is due to the constant conflict between the two classes.





- What happens if it is not possible to separate the classes with a single line? What could you say about local minima?

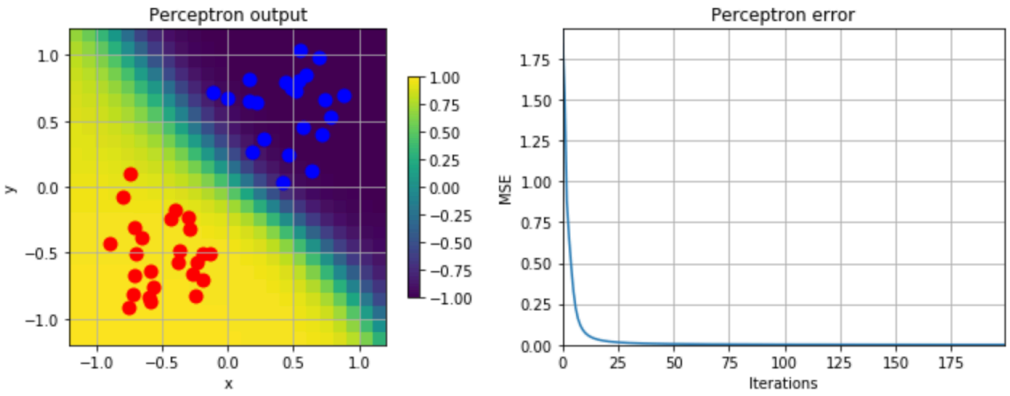
- The error is getting constant quickly. The frontier line is defined by the two clusters of the same class, and one of the clusters of the other class.



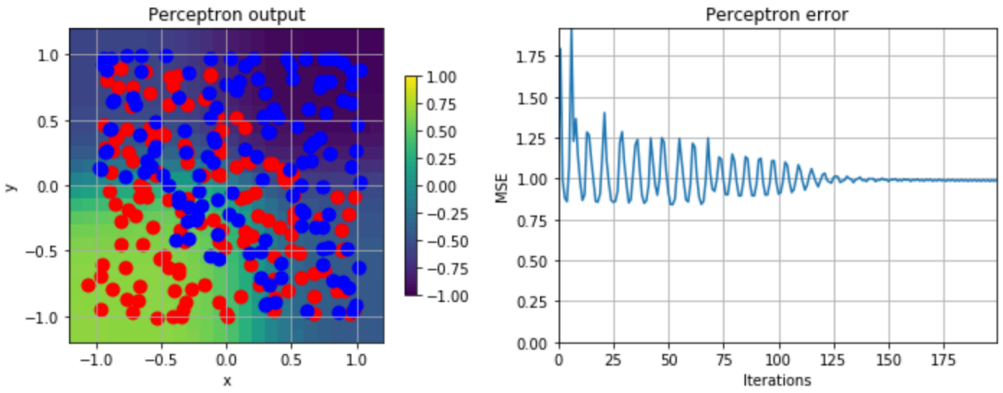
2. BACKPROPAGATION

5_backpropagation.ipynb

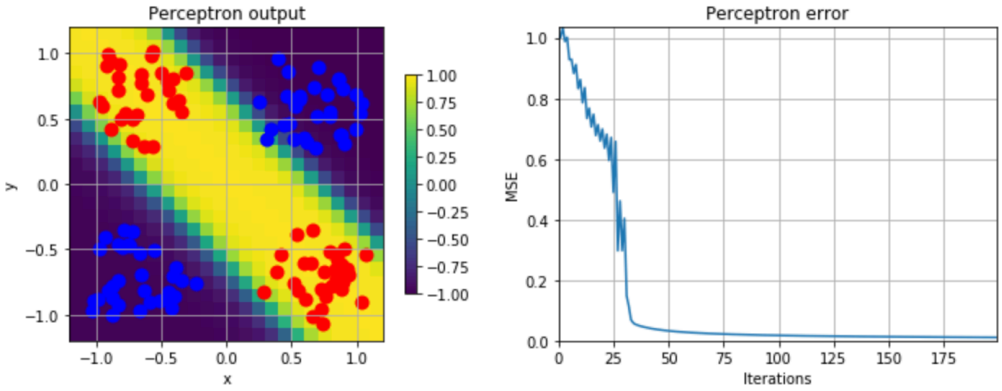
- What happens if the boundaries between both classes are well defined?
  - The error is flooring at zero very fast, and the frontier is well defined.



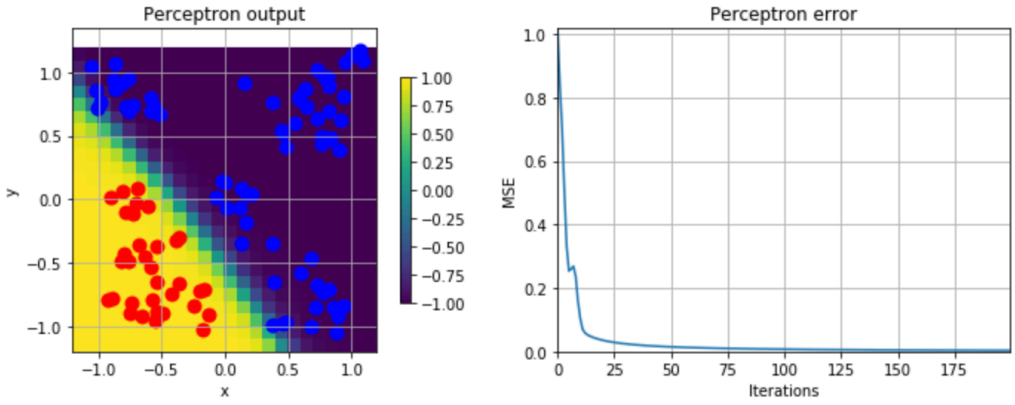
- What happens if the classes overlap? What could you say about oscillations in the error signal?
  - The error is oscillating and getting constant not that quickly. The frontier is not well defined. It's still hard for the algorithm to find a way to separate merged points.



- What happens if it is not possible to separate the classes with a single line? What could you say about local minima?
  - The error is flooring at zero very fast, and the frontiers are well defined. The frontiers are put around the 2 classes in the middle, allowing a nice operation of the 2 independent clusters of the second class.



- What happens if the points of one of the classes are separated in subgroups (blobs)?
  - The error is flooring at zero very fast, and the frontier is well defined. I even see that the frontier is curving.

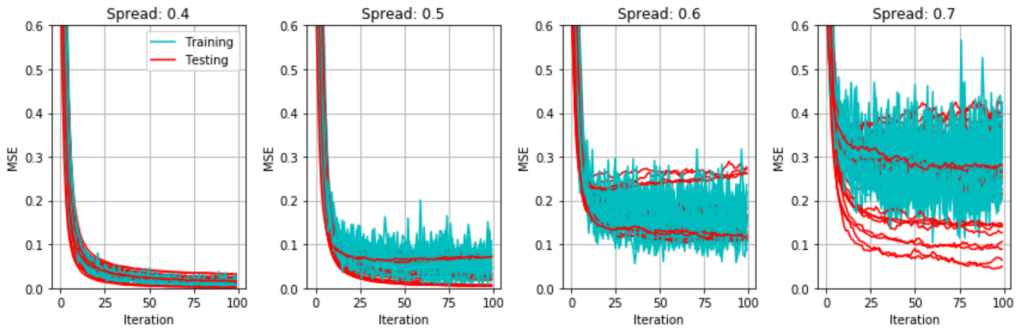


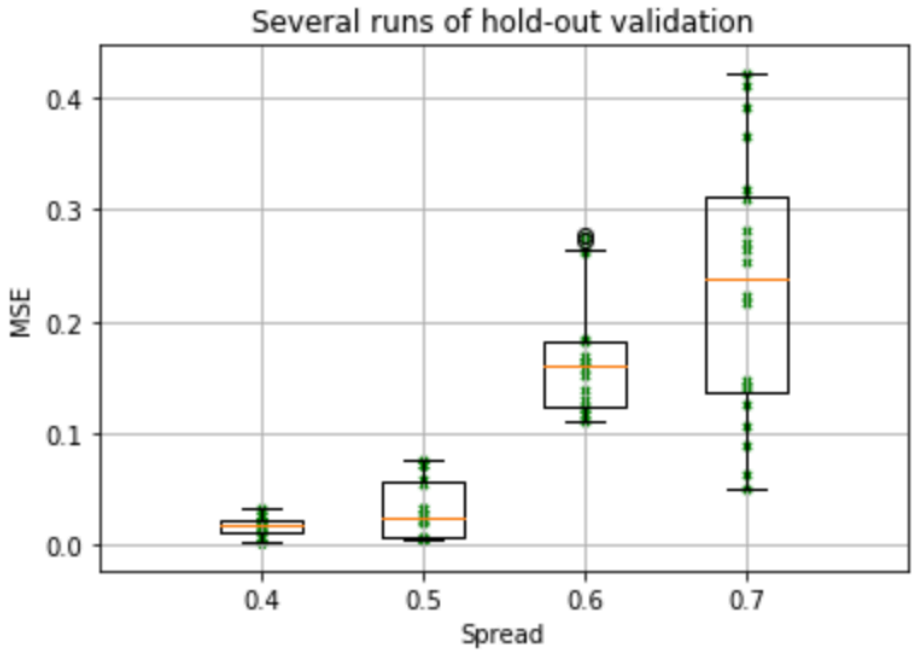
6_backpropagation_MLP.ipynb

```
47: self.delta_weights = []

55: self.delta_weights.append(np.zeros((self.layers[i - 1] + 1,
self.layers[i] + 1)))

57: self.delta_weights.append(np.zeros((self.layers[i] + 1,
self.layers[i + 1])))

108: self.delta_weights[i] = (-learning_rate * layer.T.dot(delta))
+ (momentum * self.delta_weights[i])
```

4. CROSSVALIDATION

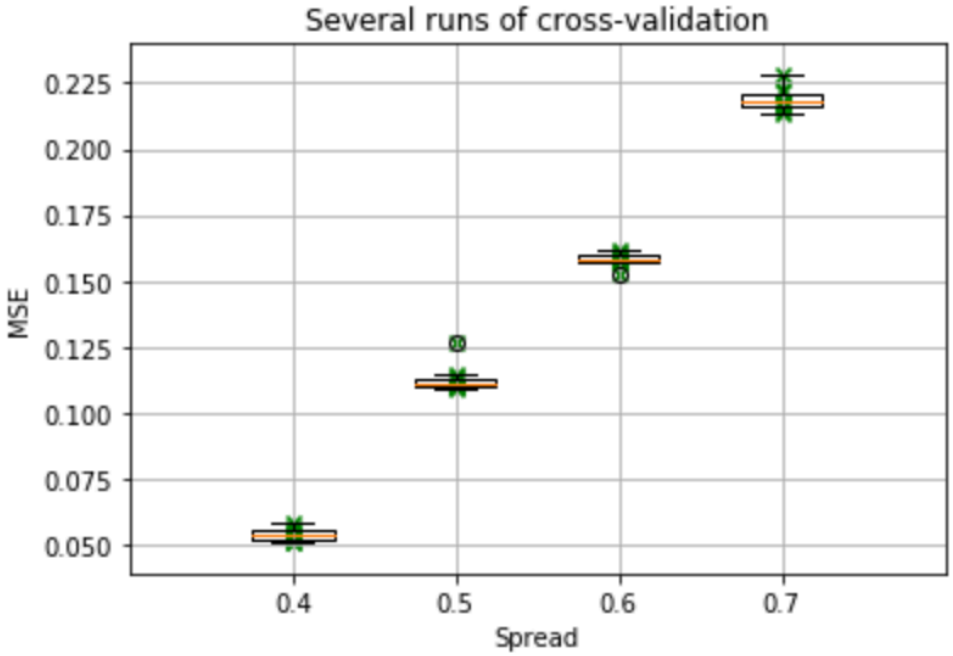
7_hold_out_validation.ipynb





- The MSE is oscillating a lot more when the spread is getting larger.
- The MSE increases with the spread size

8_cross_validation.ipynb



- We can see that the MSE oscillation is very small compared to the hold_out

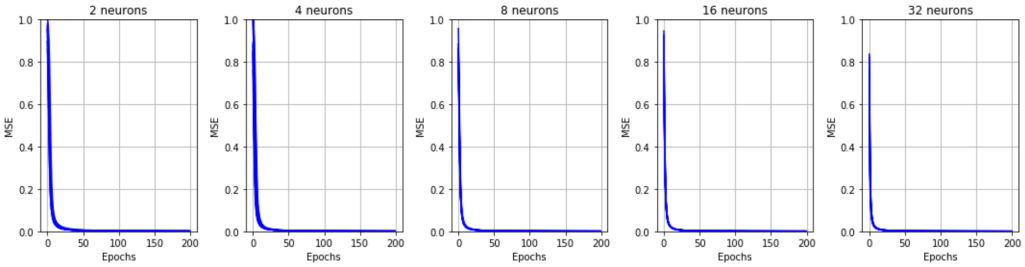
validation.

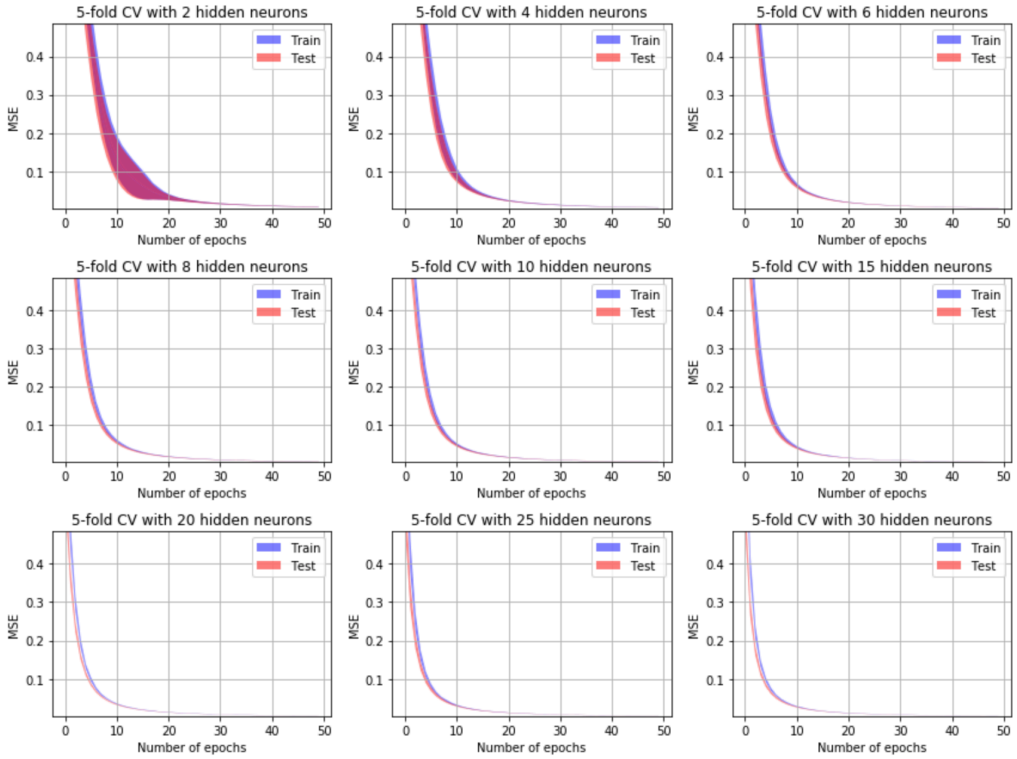
- The MSE also increases somewhat proportional with the spread size

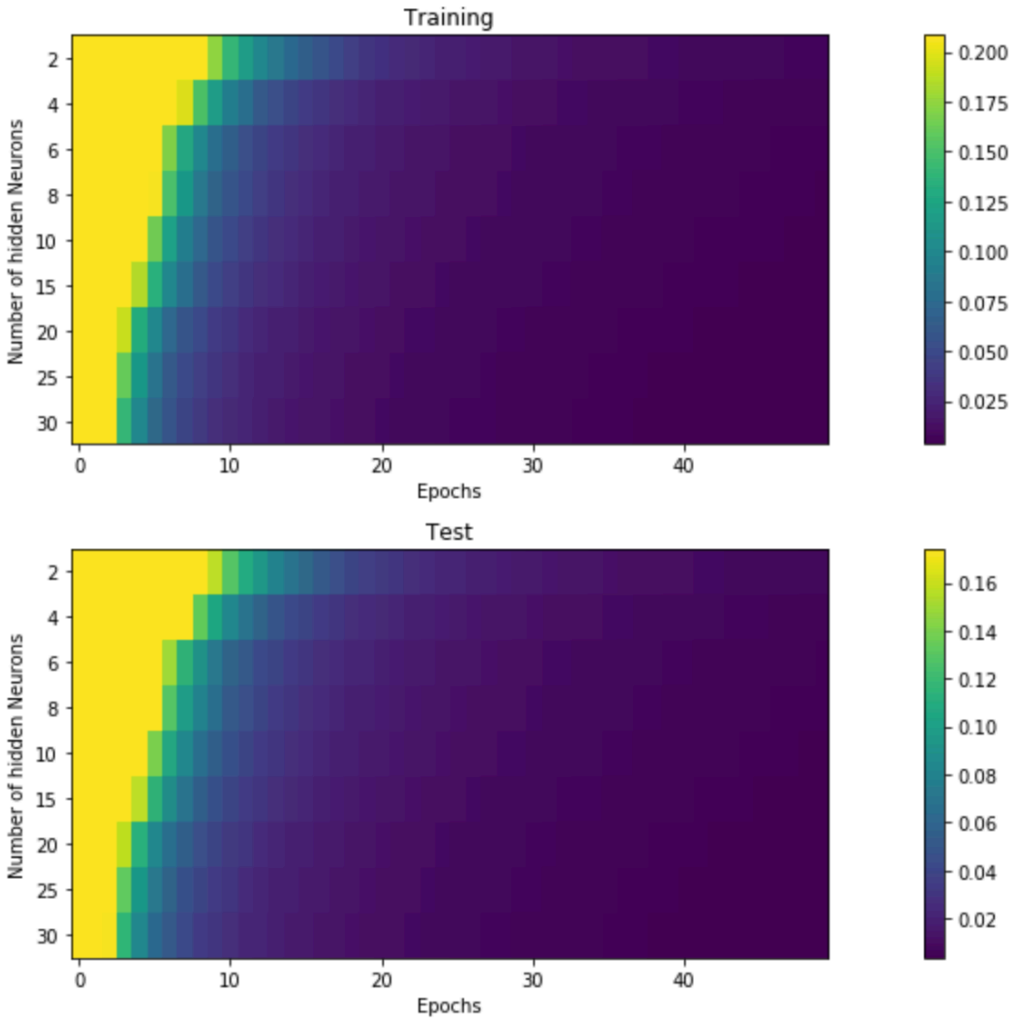
5. MODEL BUILDING

9_model_selection.ipynb

Spread = 0.1







In [60]:
```
nn = mlp.MLP([2,20,1], 'tanh')
```

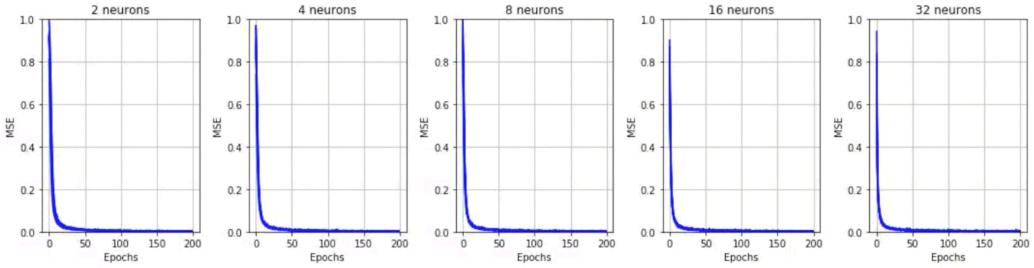
In [67]:
```
MSE_train, MSE_test, conf_mat = cv.k_fold_cross_validation(nn,
                                                           dataset,
                                                           k=K,
                                                           learning_rate=LEARNING_RATE,
                                                           momentum=MOMENTUM,
                                                           epochs=10,
                                                           threshold=0.0)
```

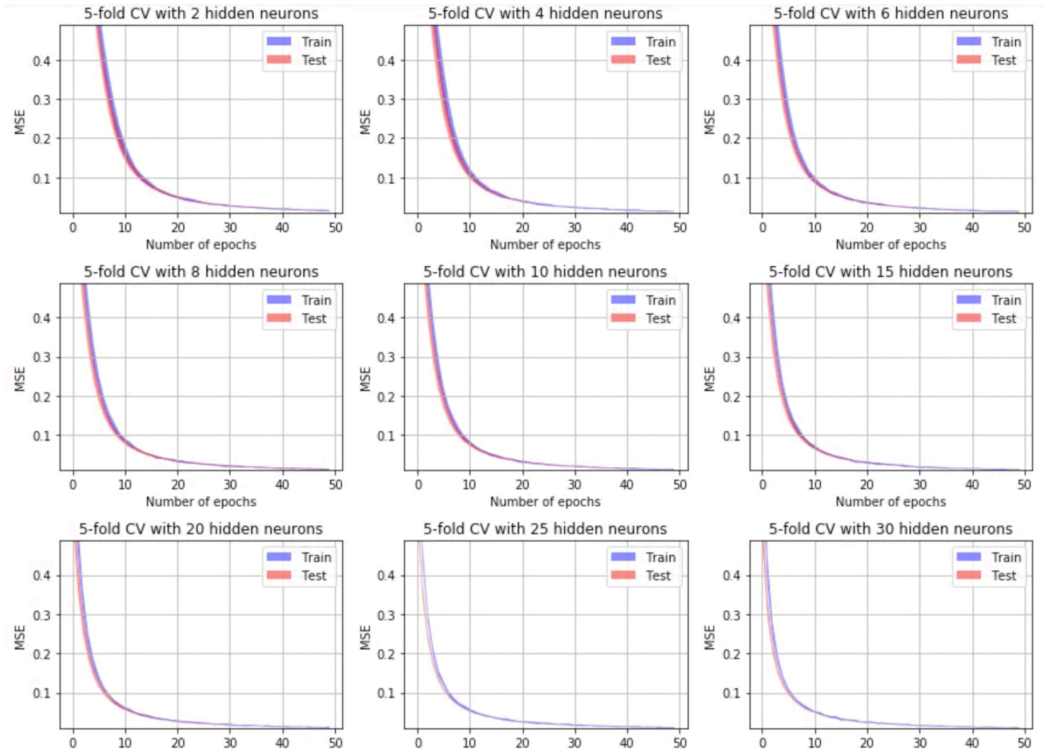
In [68]:
```
print('MSE training: ', MSE_train)
print('MSE test: ', MSE_test)
print('Confusion matrix:')
print(conf_mat)
```

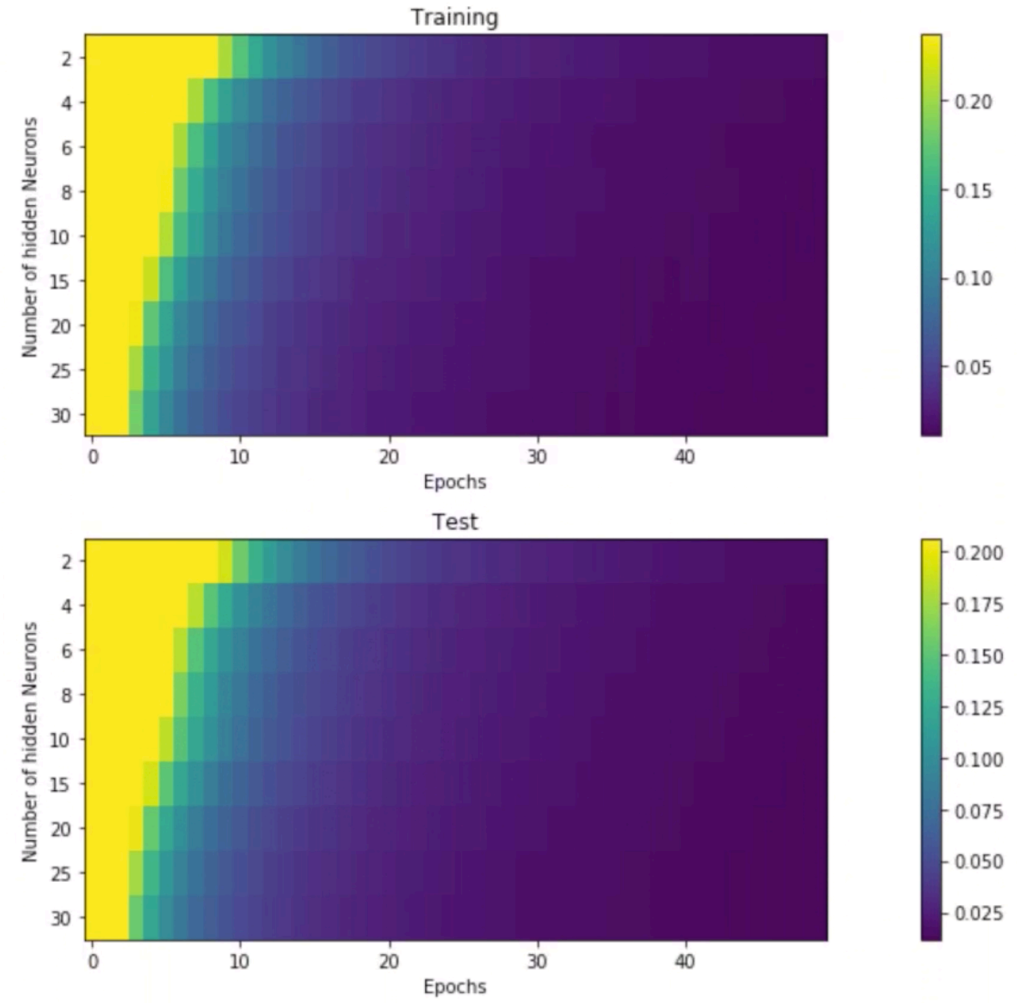
```
MSE training:  0.03634107477210422
MSE test:  0.03614249201889551
Confusion matrix:
[[100.   0.]
 [  0. 100.]]
```

- Based on the epochs exploration I am using 50 iterations.
- Based on the neurons exploration I found that 20 hidden neurons for 10 epochs should be a good match.
- The model has a perfect prediction rate.

Spread = 0.3


2 neurons
4 neurons
8 neurons
16 neurons
32 neurons
MSE
Epochs

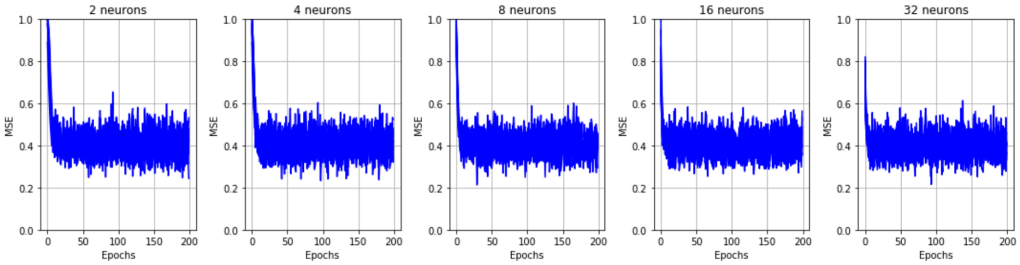

5-fold CV with 2 hidden neurons
5-fold CV with 4 hidden neurons
5-fold CV with 6 hidden neurons
5-fold CV with 8 hidden neurons
5-fold CV with 10 hidden neurons
5-fold CV with 15 hidden neurons
5-fold CV with 20 hidden neurons
5-fold CV with 25 hidden neurons
5-fold CV with 30 hidden neurons
Train
Test
MSE
Number of epochs

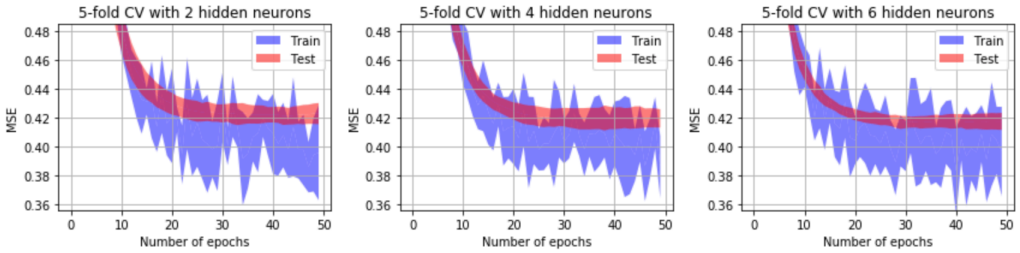


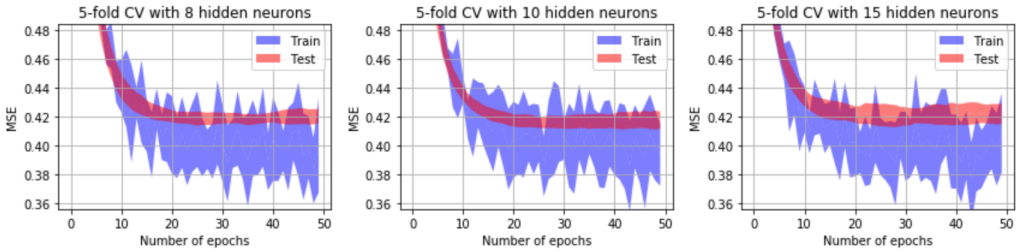

```
Entrée [15]: nn = mlp.MLP([2,25,1], 'tanh')

Entrée [20]: MSE_train, MSE_test, conf_mat = cv.k_fold_cross_validation(nn,
                                                dataset,
                                                k=K,
                                                learning_rate=LEARNING_RATE,
                                                momentum=MOMENTUM,
                                                epochs=15,
                                                threshold=0.0)

Entrée [21]: print('MSE training: ', MSE_train)
             print('MSE test: ', MSE_test)
             print('Confusion matrix:')
             print(conf_mat)

             MSE training:  0.03743239702875615
             MSE test:  0.038927467394902646
             Confusion matrix:
             [[100.   0.]
              [  0. 100.]]
```

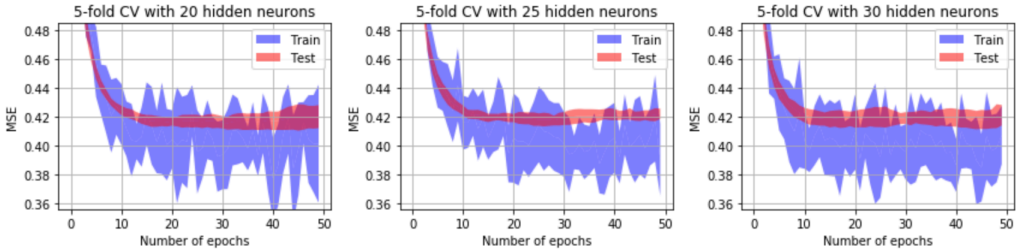
- Based on the epochs exploration I am using 50 iterations.
- Based on the neurons exploration I found that 25 hidden neurons for 15 epochs should be a good match.
- The model has a perfect prediction rate.

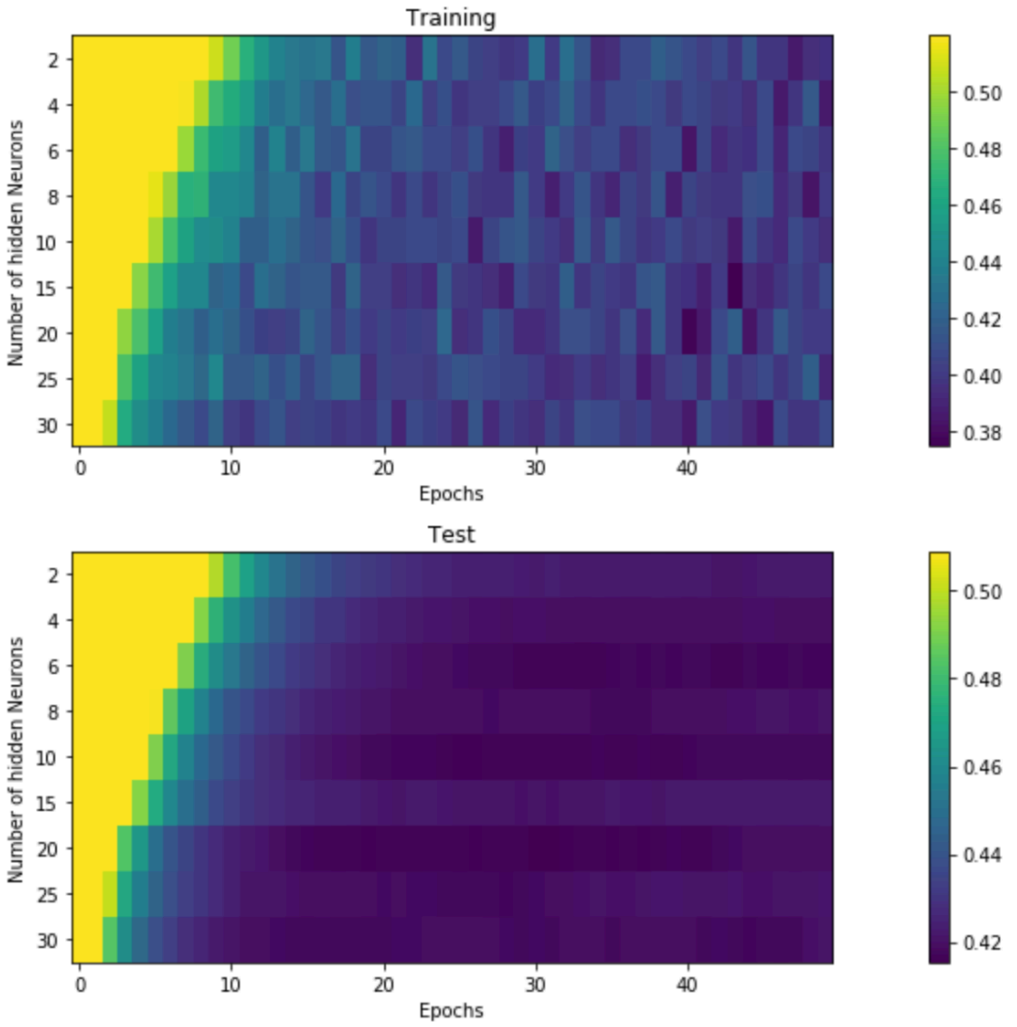
Spread = 0.9


2 neurons
4 neurons
8 neurons
16 neurons
32 neurons
MSE
Epochs


5-fold CV with 2 hidden neurons
5-fold CV with 4 hidden neurons
5-fold CV with 6 hidden neurons
Train
Test
MSE
Number of epochs


5-fold CV with 8 hidden neurons
5-fold CV with 10 hidden neurons
5-fold CV with 15 hidden neurons
Train
Test
MSE
Number of epochs


5-fold CV with 20 hidden neurons
5-fold CV with 25 hidden neurons
5-fold CV with 30 hidden neurons
Train
Test
MSE
Number of epochs



In [112]:
```
nn = mlp.MLP([2,20,1], 'tanh')
```

In [113]:
```
MSE_train, MSE_test, conf_mat = cv.k_fold_cross_validation(nn,
                                                           dataset,
                                                           k=K,
                                                           learning_rate=LEARNING_RATE,
                                                           momentum=MOMENTUM,
                                                           epochs=20,
                                                           threshold=0.0)
```

In [114]:
```
print('MSE training: ', MSE_train)
print('MSE test: ', MSE_test)
print('Confusion matrix:')
print(conf_mat)
```

```
MSE training:  0.4051705782544851
MSE test:  0.416468519028039
Confusion matrix:
[[86. 14.]
 [14. 86.]]
```

- Based on the epochs exploration I am using 50 iterations.
- Based on the neurons exploration I found that 20 hidden neurons for 20 epochs should be a good match.
- The model has a flaws in the predictions.