TBMI26 – Computer Assignment Report Supervised Learning

Deadline – March 14 2021

Author/-s: Mohammed Ali & Simon Alsén

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. Please upload the document in PDF format. **You will also need to upload all code in .m-file format**. We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the lab part of the course reported in LADOK together with the exam. If not, you'll get the lab part reported during the re-exam period.

1. **Give an overview of the four datasets from a machine learning perspective. Consider if you need linear or non-linear classifiers etc.**
   Dataset 1, 2 and 3 consists of 2 features, while dataset 4 has 64 features. Dataset 1 can almost certainly be used with a linear classifier. The classes in dataset 2, 3 and 4 however, are not linearly separable.

2. **Explain why the down sampling of the OCR data (done as pre-processing) result in a more robust feature representation. See http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits**
   The down sampling removes some of the noise, and thereby the reduces the risk for overfitting and instead results in a more generalized model.

3. **Give a short summary of how you implemented the kNN algorithm.**
   1. The Euclidean distance is measured between each point in the test data and each point in the training data.
   2. For each point in the test data, the k points in the training data with the shortest distance to the test point is selected.
   3. A point is assigned the class that is most frequent among its k nearest neighbors.

4. **Explain how you handle draws in kNN, e.g. with two classes (k = 2)?**
   If a draw occurs, we randomly select the one of the classes involved in the draw.

5. **Explain how you selected the best k for each dataset using cross validation. Include the accuracy and images of your results for each dataset.**
   1. The training data is split into n folds.
   2. The n:th fold is used as test data to be classified, while the remaining $n-1$ folds are used as training data.
   3. Step 2 is repeated for every fold. For each fold the accuracy is evaluated.
   4. Step 2 and 3 is repeated for each k from 1 to $k_{\max}$.
   5. The average accuracy is calculated for each k. The k with the highest average accuracy is selected as the best k.

Dataset 1:

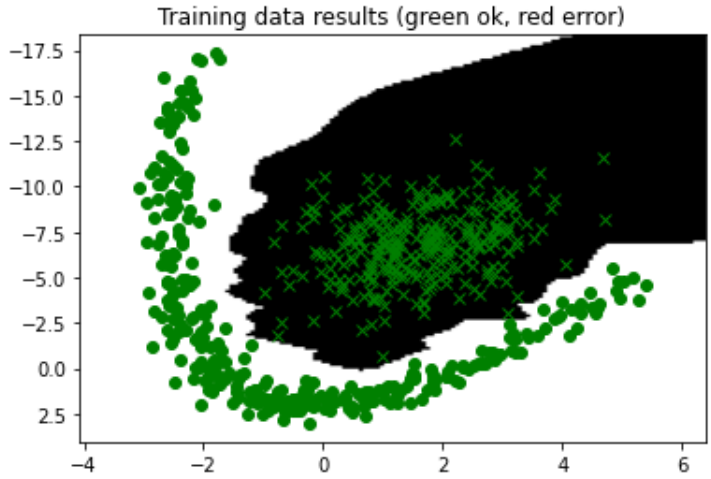
Best k: 3

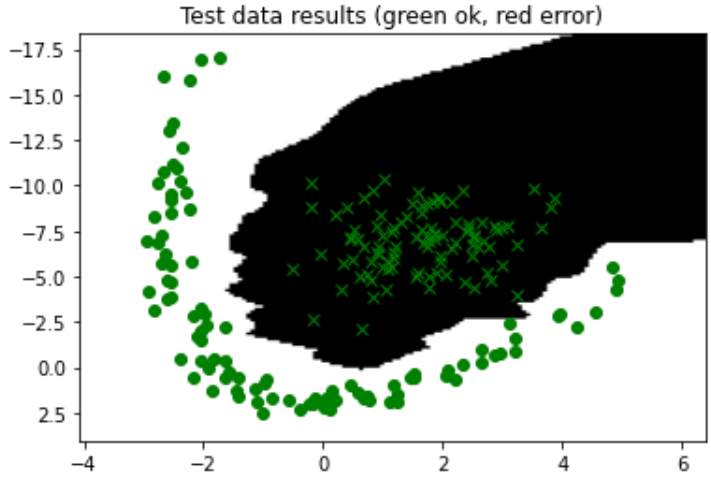
```
Confusion matrix:
[[249.   1.]
 [  2. 248.]]
Accuracy: 0.9940
```

Dataset 2:

Best k: 1

```
Confusion matrix:
[[100.   0.]
 [  0. 100.]]
Accuracy: 1.0000
```

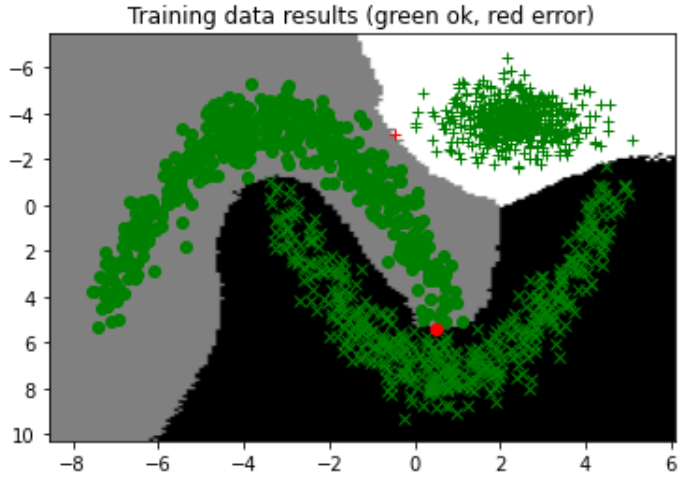


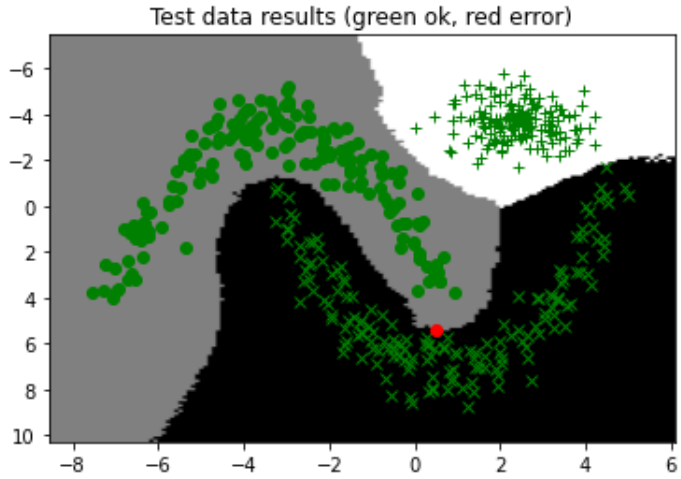


Dataset 3:

Best k: 10

```
Confusion matrix:
[[166.   0.   0.]
 [  1. 165.   0.]
 [  0.   0. 166.]]
Accuracy: 0.9980
```





Dataset 4:

Best k: 5

```
Confusion matrix:
[[138.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0. 137.   0.   0.   0.   0.   0.   1.   0.   0.]
 [  0.   0. 136.   0.   0.   0.   0.   2.   0.   0.]
 [  0.   0.   0. 136.   0.   1.   0.   0.   0.   1.]
 [  0.   0.   0.   0. 136.   0.   0.   2.   0.   0.]
 [  0.   0.   0.   1.   0. 135.   0.   0.   0.   2.]
 [  0.   1.   0.   0.   0.   0. 137.   0.   0.   0.]
 [  0.   1.   0.   0.   0.   0.   0. 137.   0.   0.]
 [  0.   8.   0.   0.   0.   0.   0.   0. 129.   1.]
 [  0.   2.   0.   1.   0.   0.   0.   0.   0. 135.]]
Accuracy: 0.9826
```

6. **Give a short summary of your backprop implementations (single + multi). You do not need to derive the update rules.**
   1. The weights w_0 are initialized, and biases is added for every input unit.
   2. The inputs are forward propagated through the network. In the single-layer implementation the output is calculated as $Y = X \cdot W$. In the multi-layer implementation, the activation of hidden neurons are calculated by using the hyperbolic tangent.
   3. The error of the desired outputs and the actual outputs is calculated.
   4. The error is backpropagated.
   5. The weights are updated.

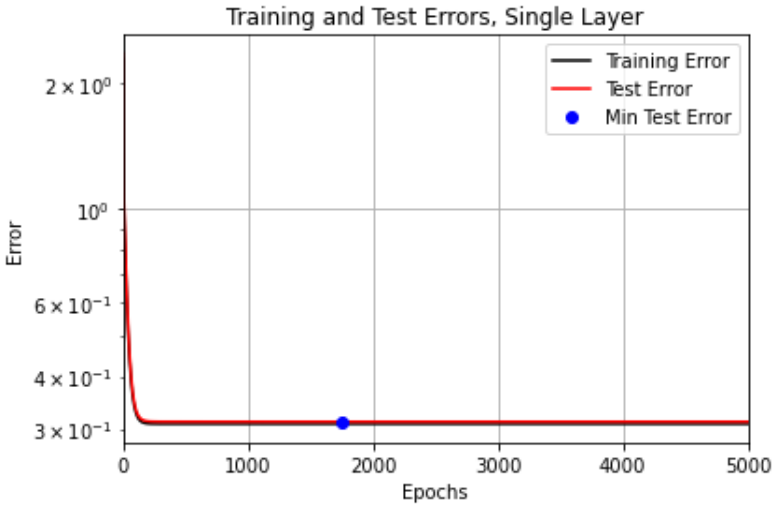
7. **Present the results from the neural network training and how you reached the accuracy criteria for each dataset. Motivate your choice of network for each dataset. Explain how you selected good values for the learning rate, iterations and number of hidden neurons. Include images of your best result for each dataset, including parameters etc.**

   Dataset 1:

   Network: Single layer
   numIterations = 5000

learningRate = 0.01

Dataset 1 is linearly separable, and because of that we have chosen to use a single-layer network. The learning rate was set to 0.01 through trial and error. We used 5000 iterations, but as can be seen in the figure below, the minimum test error is reached much earlier than that. Therefore, the number of iterations can be reduced without losing accuracy.

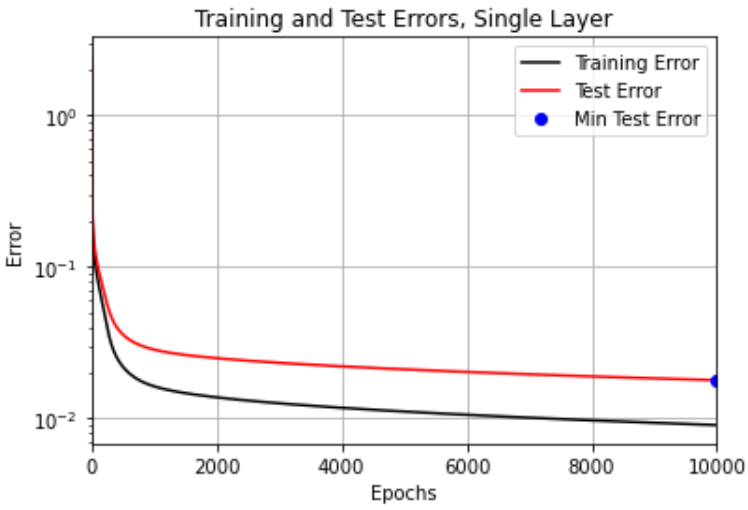



```
Confusion matrix:
[[499.   1.]
 [  8. 492.]]
Accuracy: 0.9910
```

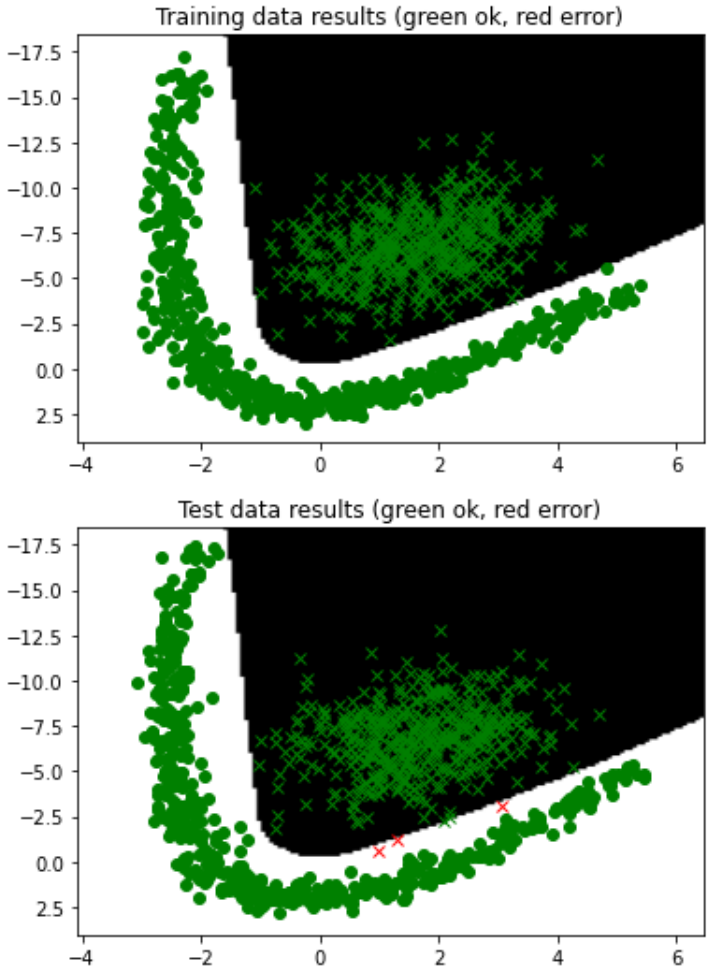
Dataset 2:

Network: Multi-layer
numHidden = 30
numIterations = 10000
learningRate = 0.01

Since dataset 2 is not linearly separable, we have chosen to use a multi-layer network. After some trial and error, we decided to set the number of hidden nodes to 30, and the learning rate to 0.01. The test error could potentially be lowered further by increasing the number of iterations, however 10 000 is enough to reach the required 99% accuracy.



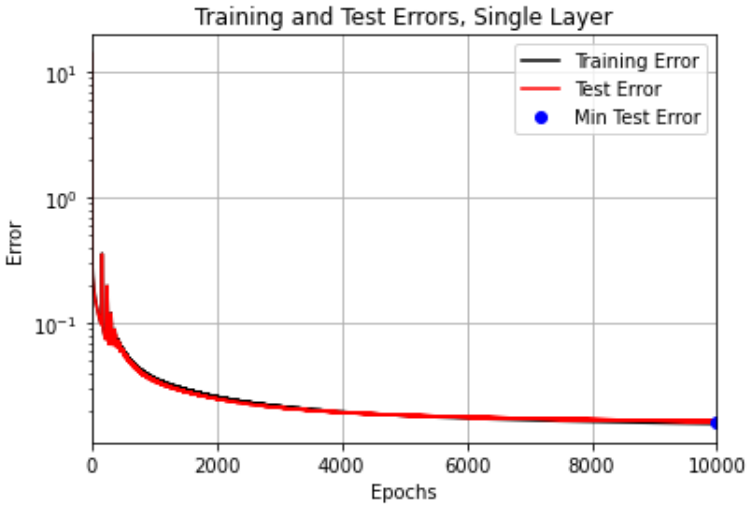

```
Confusion matrix:
[[497.   3.]
 [  0. 500.]]
Accuracy: 0.9970
```



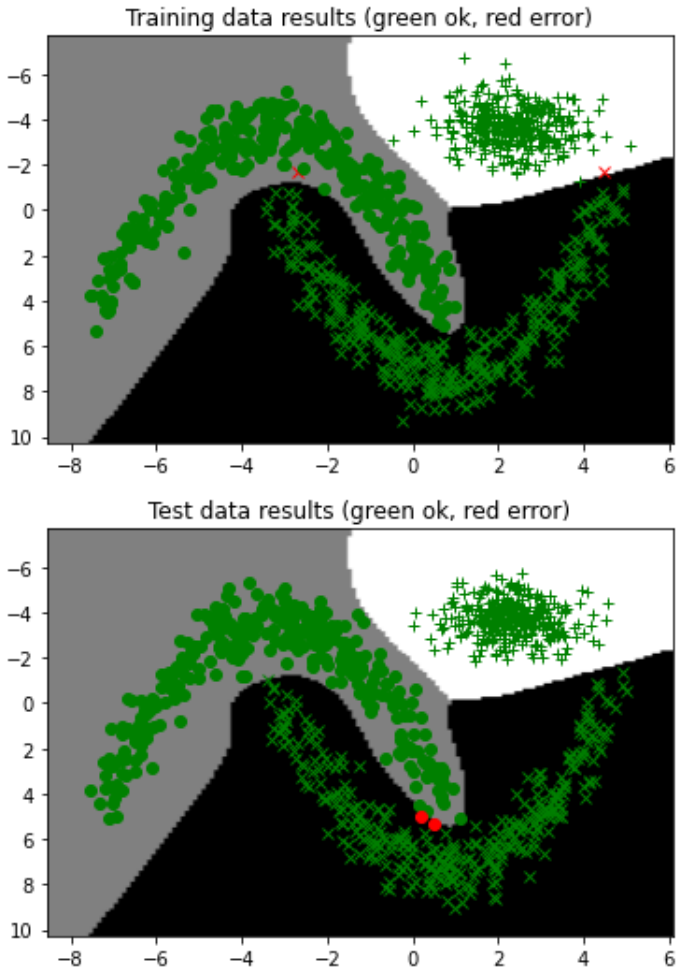
Dataset 3:

Network: Multi-layer
numHidden = 30
numIterations = 10000
learningRate = 0.05

For dataset 3 we have used a multi-layer network since the classes are not linearly separable. The learning rate was set to 0.05 and the number of hidden nodes was set to 30 after trial and error. With these setting, 10 000 iterations was enough to reach an accuracy that is greater than the required 99%.



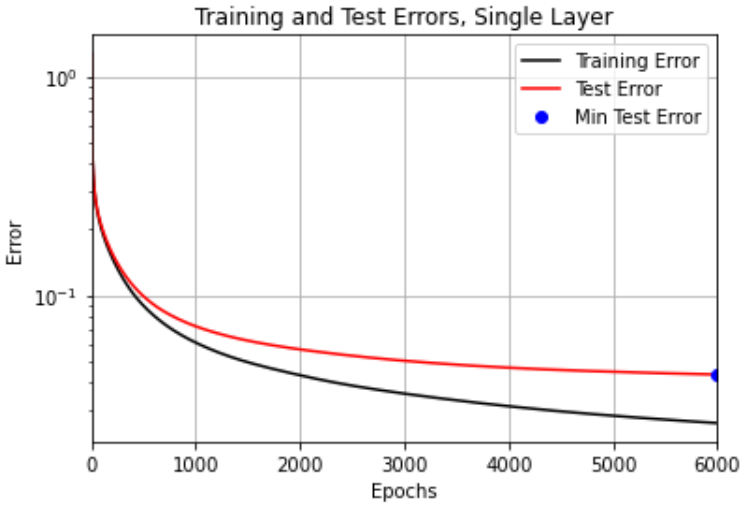

```
Confusion matrix:
[[333.   0.   0.]
 [  2. 331.   0.]
 [  0.   0. 333.]]
Accuracy: 0.9980
```



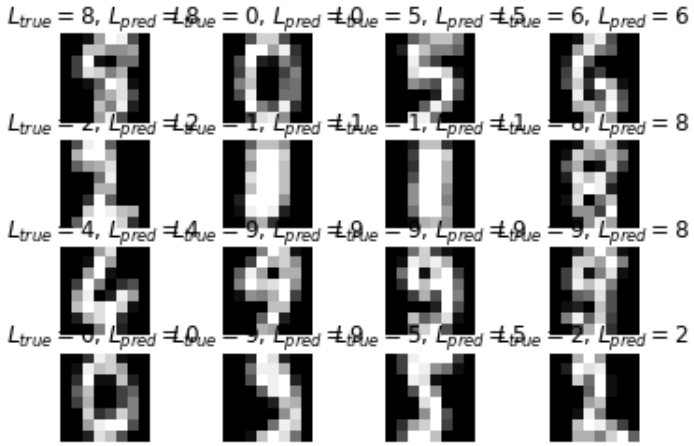
Dataset 4:
Network: Multi-layer
numHidden = 60

numIterations = 6000
learningRate = 0.005

For dataset 4, we have used a multi-layer network. Since the problem is more complex (the dataset contains both more classes and features), we have increased the number of hidden nodes to 60. The learning rate was set to 0.005 after trial and error. We used 6000 iterations, which was enough to reach the required 96% accuracy. It is possible that the accuracy could have been improved by running more iterations, however that would also have vastly increased the running time.



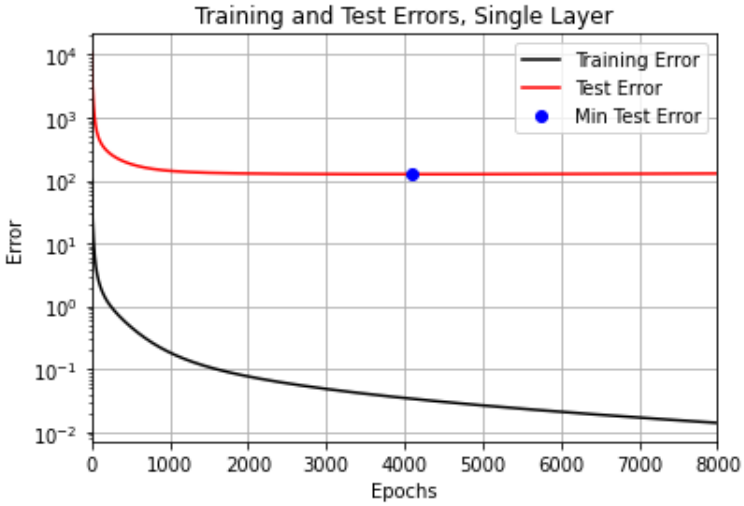

```
Confusion matrix:
[[274.   0.   1.   0.   2.   0.   0.   0.   0.   0.]
 [  0. 267.   1.   0.   0.   2.   0.   2.   3.   2.]
 [  0.   1. 268.   1.   0.   0.   1.   1.   4.   1.]
 [  0.   1.   1. 264.   0.   6.   0.   0.   3.   2.]
 [  0.   3.   0.   0. 269.   0.   3.   2.   0.   0.]
 [  0.   1.   1.   2.   0. 265.   2.   0.   2.   4.]
 [  0.   3.   0.   0.   2.   0. 270.   0.   2.   0.]
 [  0.   1.   0.   0.   0.   0.   0. 272.   4.   0.]
 [  1.   7.   1.   2.   1.   3.   0.   0. 259.   3.]
 [  1.   3.   1.   2.   3.   6.   0.   2.   4. 255.]]
Accuracy: 0.9614
```

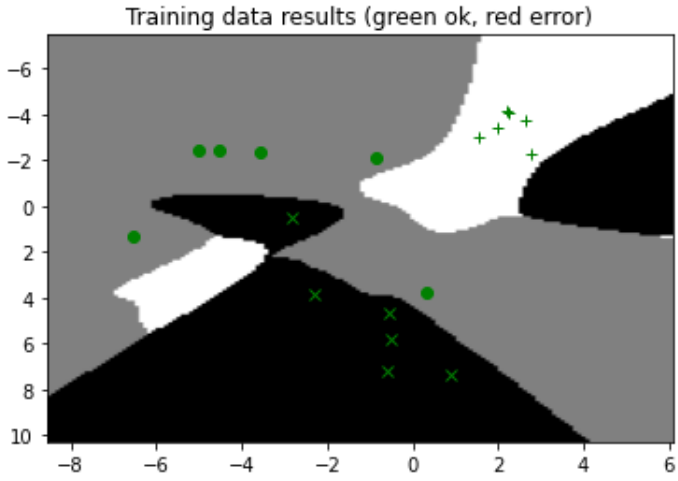


8. **Present the results, including images, of your example of a non-generalizable backprop solution. Explain why this example is non-generalizable.**

A non-generalizable solution works well on the training data, but perform worse on the test data. This can be caused by overfitting the model to the training data. If the number of samples in the training data is low, the number of hidden neurons high, and the learning rate low the risk of overfitting increases.

In the following example, we have trained the network using only 18 samples from dataset 3, using a low learning rate (0.0001), a high number of hidden neurons (400), and a high number of iterations (8000). This causes the network to overfit to the training data, and results in perfect accuracy for the samples in the training dataset. However, as can be seen in the last figure, the model does not generalize well which means that it performs poorly on previously unseen data.





9. **Give a final discussion and conclusion where you explain the differences between the performances of the different classifiers. Pros and cons etc.**
   KNN is computationally expensive, but easier to interpret than the neural networks. Another disadvantage of KNN is that all training data needs to saved.

   Neural networks often give better results, but can sometimes be prone to overfitting.

10. **Do you think there is something that can improve the results? Pre-processing, algorithm-wise etc.**
    - The inputs can be normalized.
    - Use cross-validation to find the optimal number of hidden neurons and value of learning rate.
    - Another activation function for the hidden neurons could potentially improve the results.

11. **Optional task (but very recommended). Simple gradient decent like what you have implemented can work well, but in most cases we can improve the weight update by using more sophisticated algorithms. Some of the most common optimization algorithms are summarized nicely here:**
    https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
    Implement one or a few different optimizers and compare the speed at which the training converges. A good starting point is to implement momentum gradient decent.