

# TBMI26

## Assignment 1

Martin Estgren <mares480>

8 februari 2017

## 1 Assignment 1

### 1.1 Overview of the data

Looking at the data from a machine learning perspective we can observe how the data is compressed from 32x32 bitmap images to 64 features with the integer range 0..16. This means that we can create a hypercube feature space with 64 dimensions. This is a significantly dimensionally reduced feature space compared to 32x32 binary dimensions.

The pre-processing of the data doesn't only reduce the number of dimensions, it also reduces the variance in small distortions.

### 1.2 Implementation of the kNN algorithm

#### 1.2.1 kNN without cross-validation

The implementation of kNN is fairly straight forward.

We iterate through all the cases in the test set and calculate the distance to each point in the training set. The result is then sorted in ascending order and the 'k' first elements are counted in regards to what label they are assigned. The label with the most values are then picked to be assigned for the testing case we are currently processing. When no counted label is higher than another we pick the label with the data point which have the closest euclidean distance to the test case.

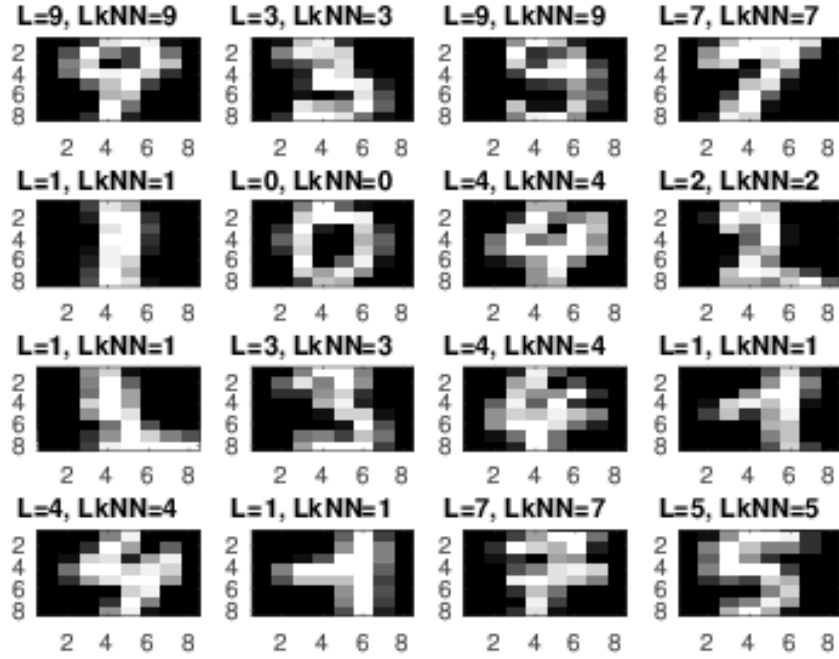
The result of our implementation of the kNN algorithm with k arbitrarily choose as 4 can be seen below.

Parameters:  $k = 4$  Accuracy = 0.9710

cM =

99	0	0	0	0	0	0	0	0	0
0	97	1	0	0	0	0	0	2	3
0	0	97	0	0	0	0	0	0	1
0	1	0	98	0	0	0	0	0	3
0	1	0	0	96	0	0	0	0	2
0	0	0	1	0	100	0	0	0	1
1	0	0	0	0	0	100	0	0	0
0	0	2	0	0	0	0	100	0	0
0	1	0	0	1	0	0	0	96	2
0	0	0	1	3	0	0	0	2	88

Figur 1: Result from kNN



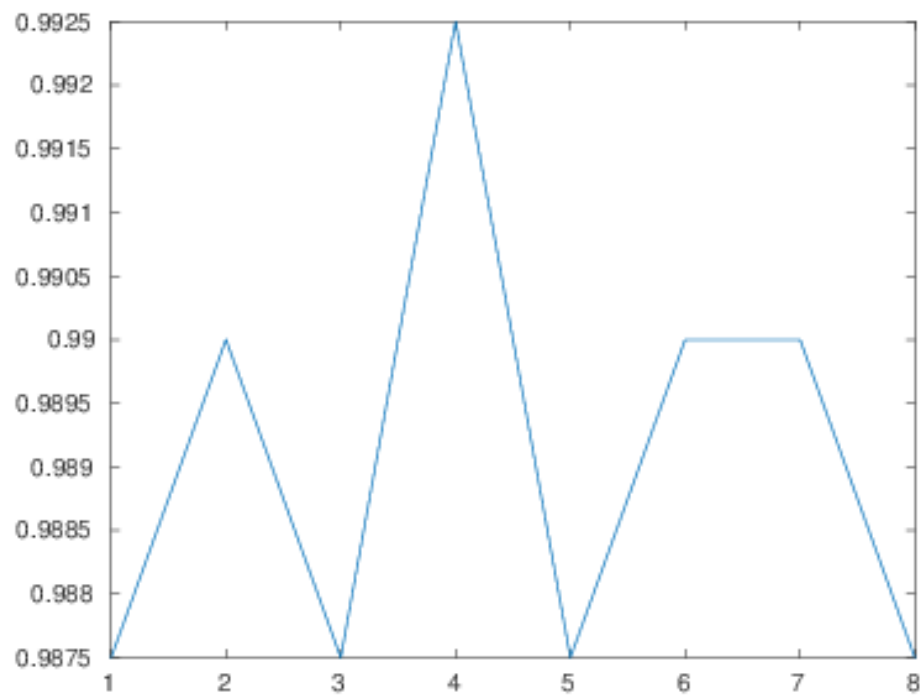
### 1.2.2 kNN with n-fold cross validation

For the cross validation version the n-fold cross validation algorithm was used to determine the best value for k. For all of the following results  $n = 2$  was used but the algorithm implementation allow for any value of n as long as it

can evenly distribute the data set. Accuracy is used as the validation score in order to find the best value for  $k$ .

### Dataset 1

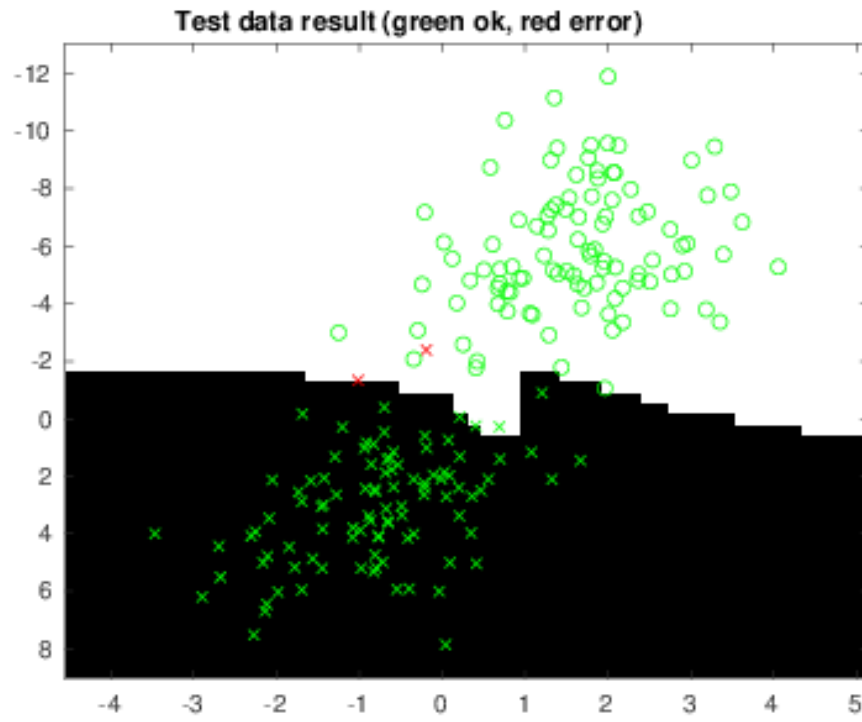
Figur 2: Cross validation score



### Best parameters:

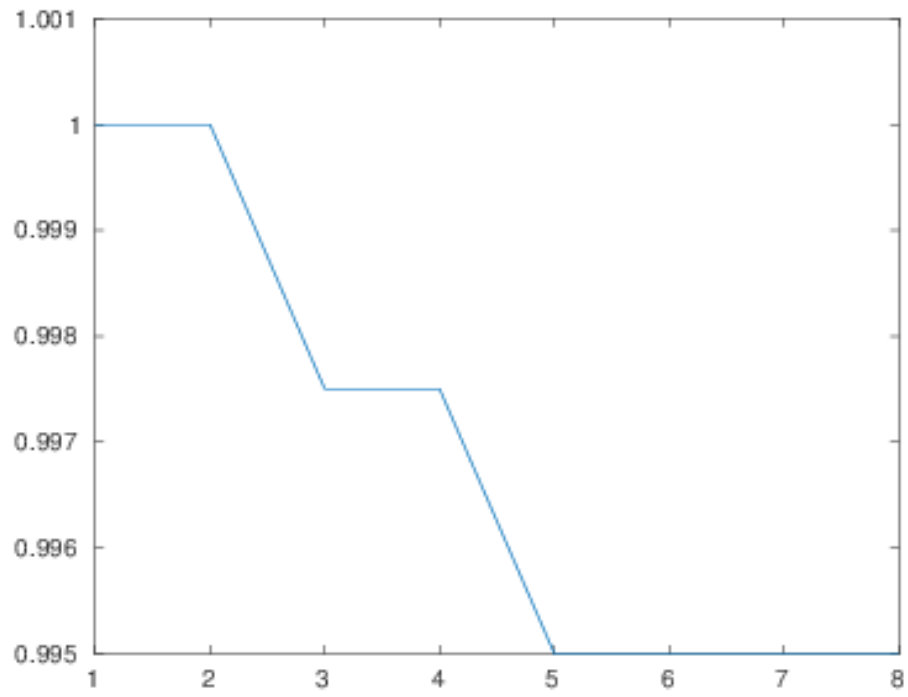
- $k = 1$
- $Accuracy = 0.9900$

Figur 3: Cross validation result



Dataset 2

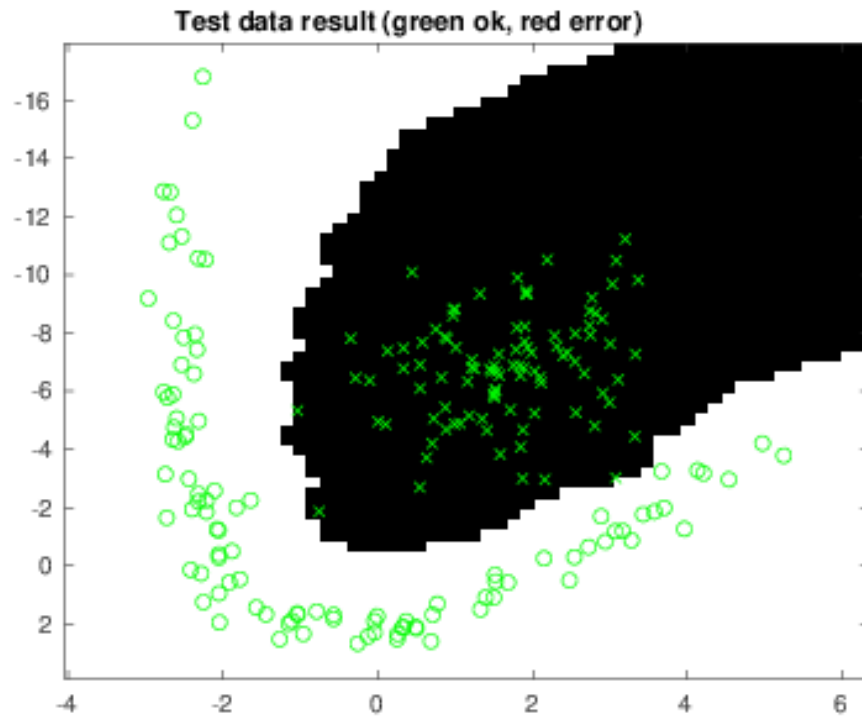
Figur 4: Cross validation score



**Best parameters:**

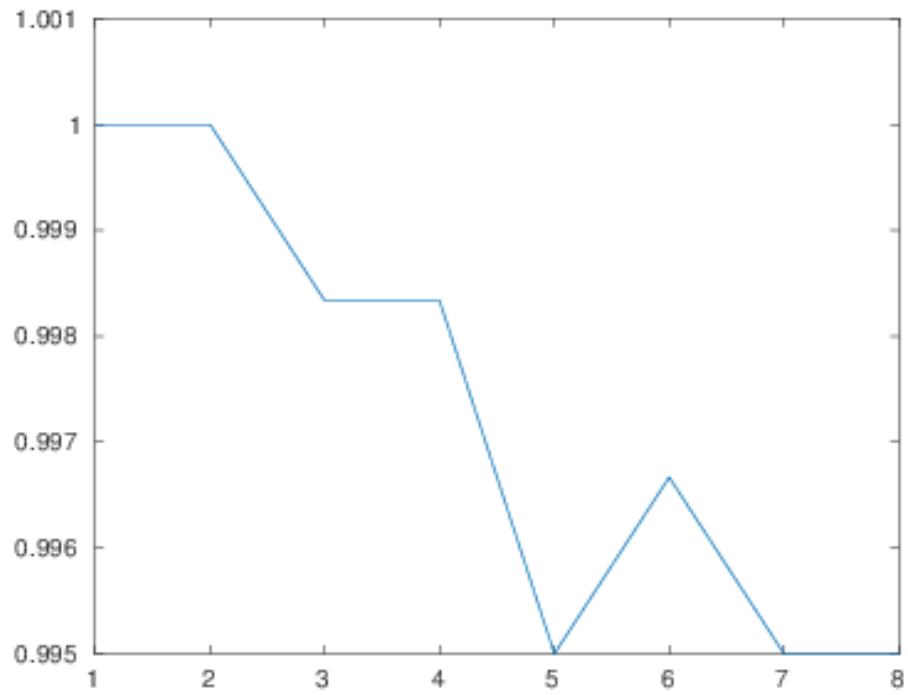
- $k = 1$
- $Accuracy = 1$

Figure 5: Cross validation result



Dataset 3

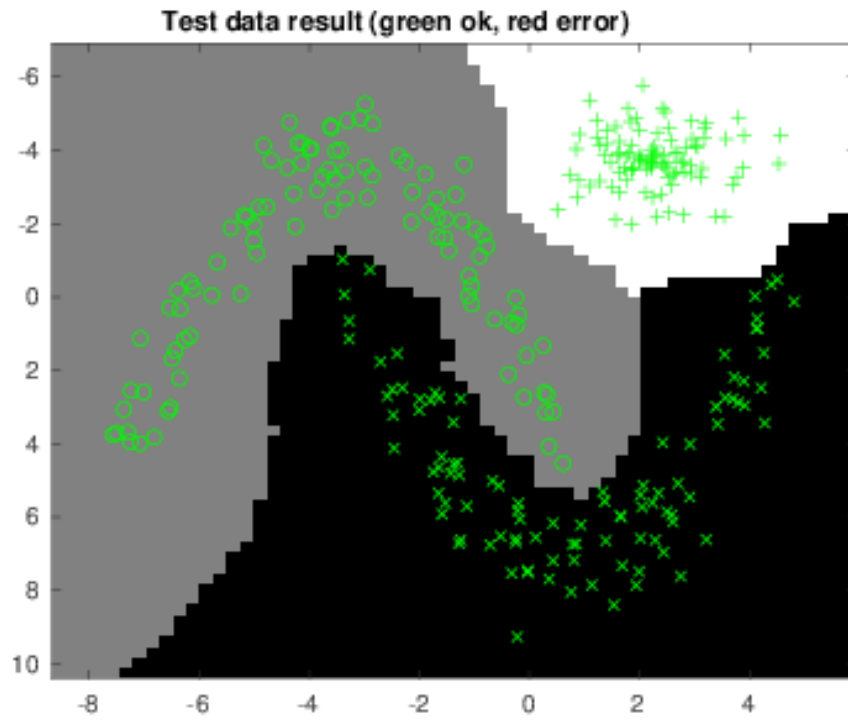
Figur 6: Cross validation score



**Best parameters:**

- $k = 1$
- $Accuracy = 1$

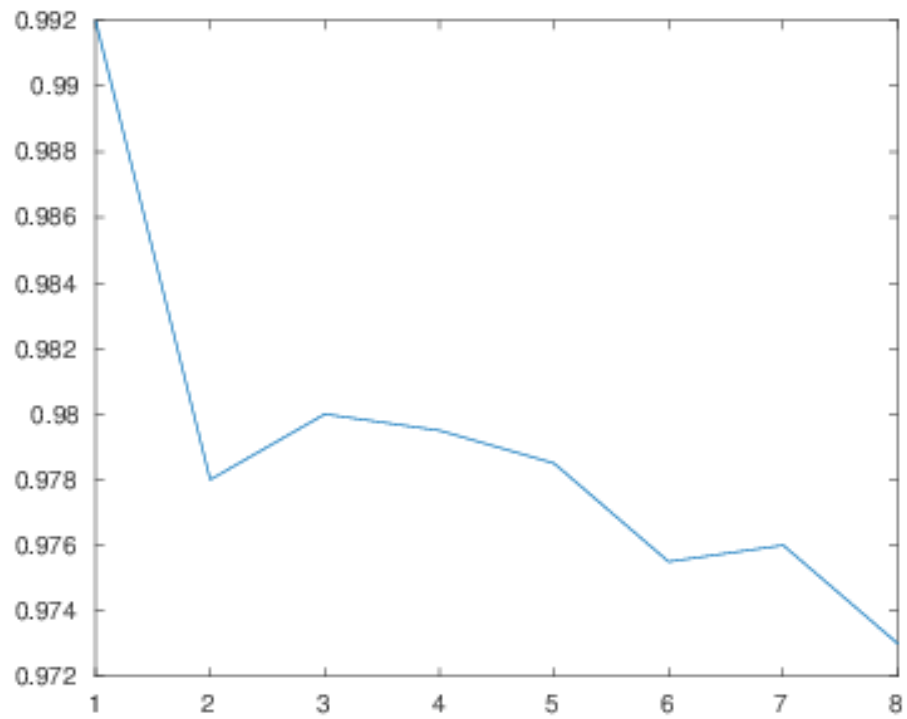
Figure 7: Cross validation result



Dataset 4



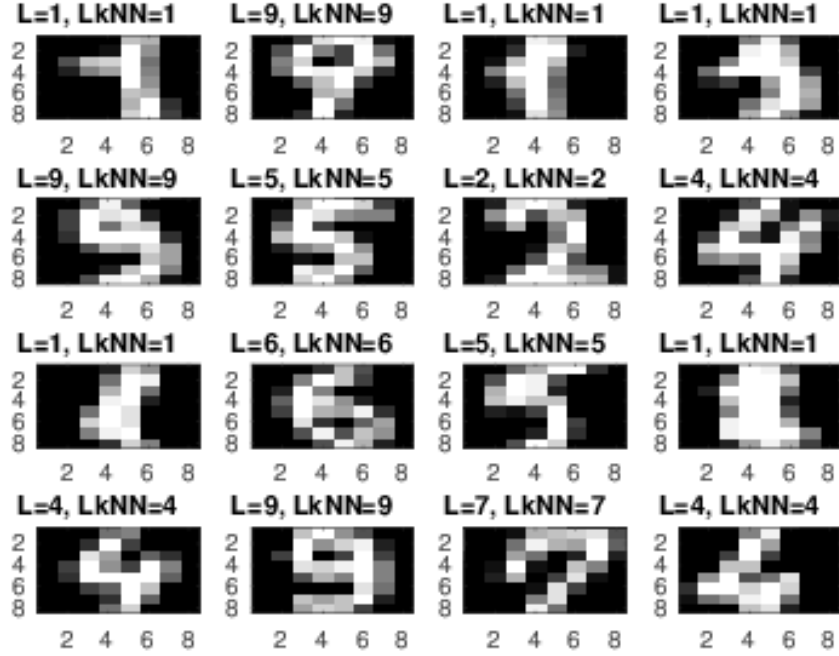
Figur 8: Cross validation score



**Best parameters:**

- $k = 1$
- $Accuracy = 0.9840$

Figure 9: Cross validation result



### 1.3 Single/Multi-layer neural network

#### 1.3.1 Single layer backpropagation

The implementation of the single layer backpropagation algorithm we use a linear activation function and the gradient  $\frac{n}{s}(Y - Dt) * Xt^t$  where  $n$  is the number of case,  $Y$  is the result from the forward propagation,  $Dt$  is the expected value for  $Y$  and  $Xt$  is a matrix with all training features.

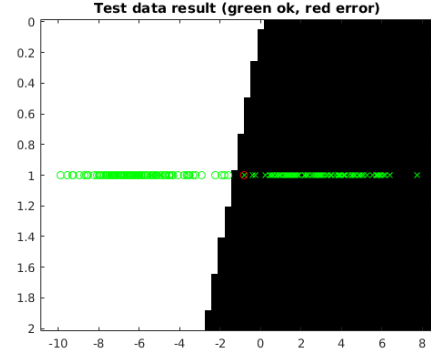
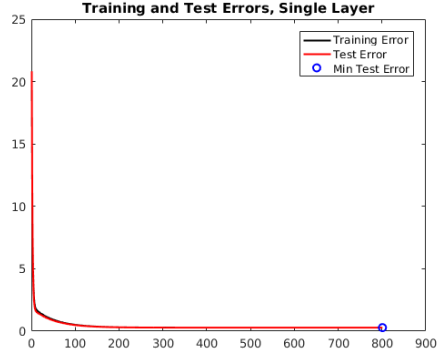
##### Dataset 1

This is not a large feature space meaning we don't need huge precision in our weights.

##### Parameters:

- *Iterations* = 800
- *learningrate* = 0.005
- *Accuracy* = 0.9950

Figure 10: Single-layer network error      Figure 11: Single-layer network result



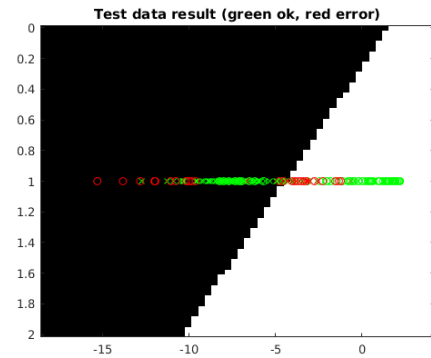
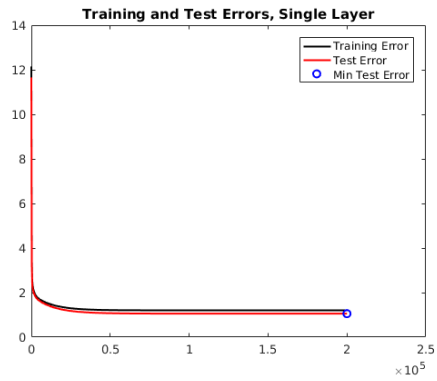
## Dataset 2

For this data set we require much more precision which is why we have bumped up the number of iterations and reduced the learning rate.

### Parameters:

- $Iterations = 200000$
- $learningrate = 0.00005$
- $Accuracy = 0.8200$

Figure 12: Single-layer network error      Figure 13: Single-layer network result



## Dataset 3

These parameters should work just fine for this data set as well since we are classifying something that is similar to the last data set.

### Parameters:

- $Iterations = 200000$

- $learningrate = 0.00005$
- $Accuracy = 0.8533$

Figure 14: Single-layer network error

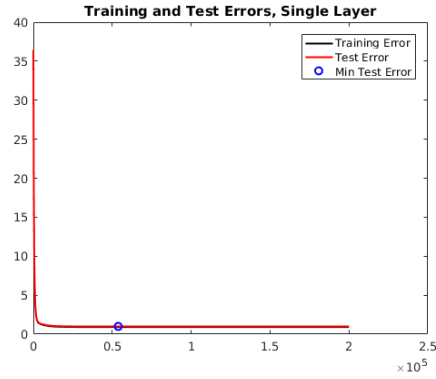
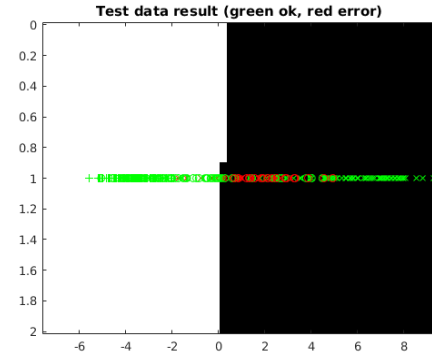


Figure 15: Single-layer network result



#### Dataset 4

This data set is much more complex compared to the other three but we are still dealing with the same number of neurons.

##### Parameters:

- $Iterations = 200000$
- $learningrate = 0.00005$
- $Accuracy = 0.9160$

Figur 16: Single-layer network error

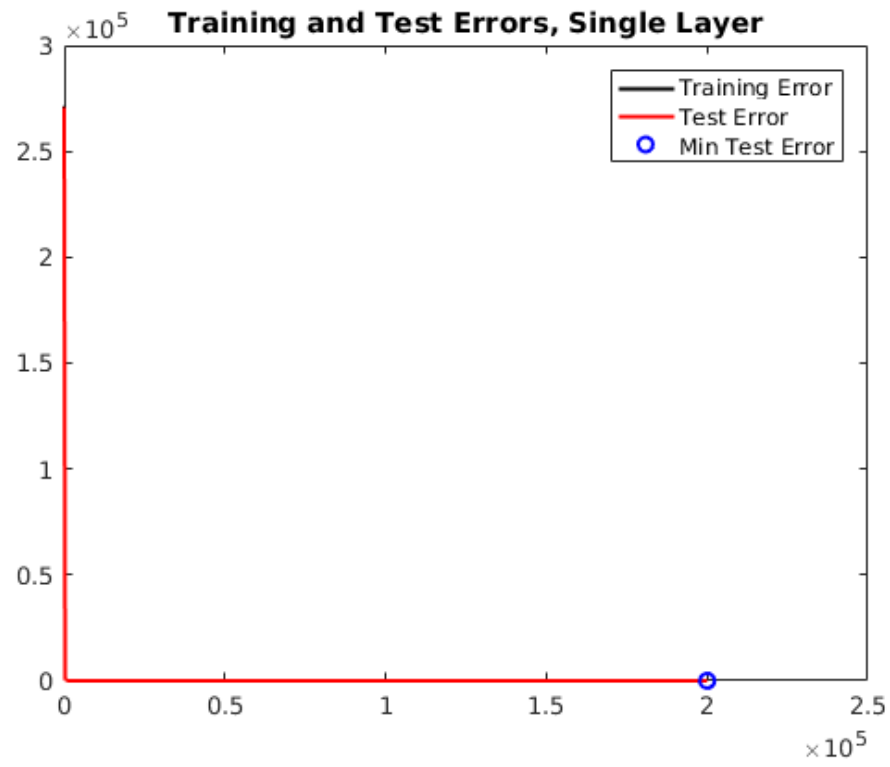
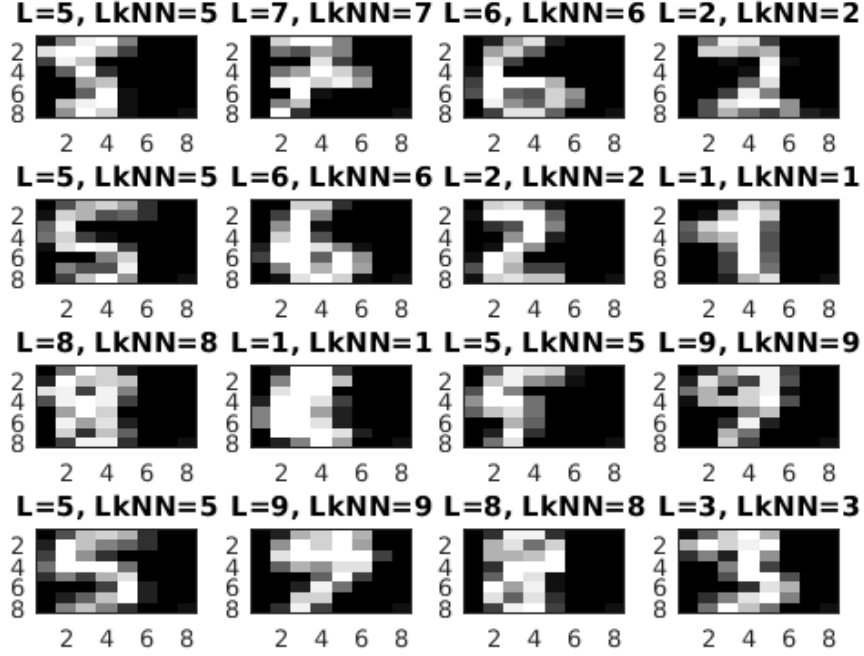


Figure 17: Single-layer network result



### 1.3.2 Multi-layer backpropagation

When dealing with multi layer networks we have to take care of order-of-magnitudes more number of weights. Because of this we want to limit the number of hidden neurons as much as we can. We are also using a non-linear activation function for the hidden layer which means we get transformed information at the linear output layer.

#### Dataset 1

For this dataset we progressively increased the number of hidden neurons, iterations, and lowered the learning rate until we got within our goal *Accuracy*

##### Parameters:

- *Hidden* = 5
- *Iterations* = 2000
- *learningrate* = 0.01
- *Accuracy* = 0.9950

Figure 18: Multi-layer network error

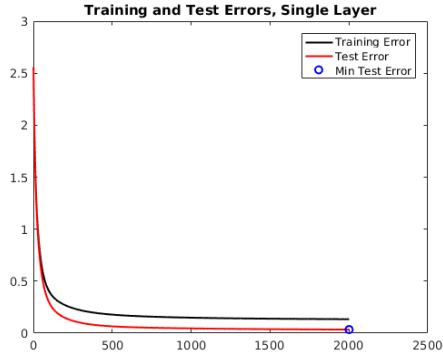
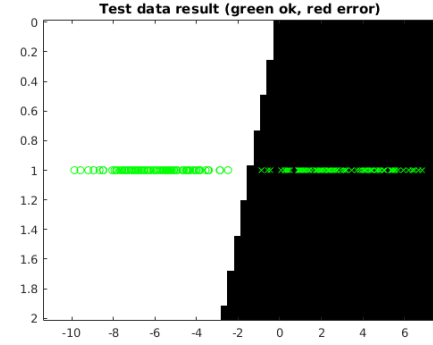


Figure 19: Multi-layer network result



## Dataset 2

For this dataset we progressively increased the number of hidden neurons, iterations, and lowered the learning rate until we got within our goal *Accuracy*

### Parameters:

- *Hidden* = 2
- *Iterations* = 30000
- *learningrate* = 0.005
- *Accuracy* = 1

Figure 20: Multi-layer network error

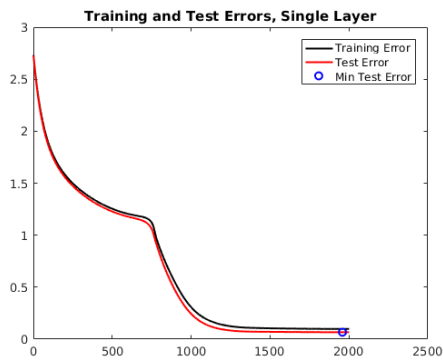
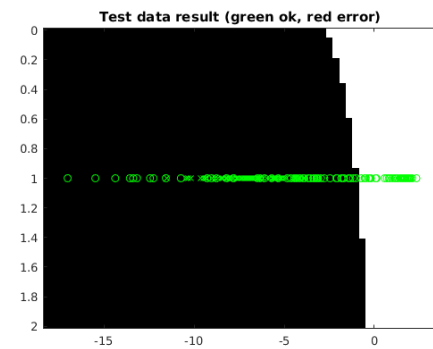


Figure 21: Multi-layer network result



## Dataset 3

For this dataset we progressively increased the number of hidden neurons, iterations, and lowered the learning rate until we got within our goal *Accuracy*

**Parameters:**

- *Hidden* = 10
- *Iterations* = 4000
- *learningrate* = 0.005
- *Accuracy* = 0.9967

Figure 22: Multi-layer network error

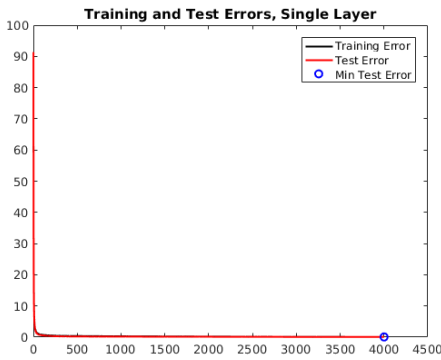
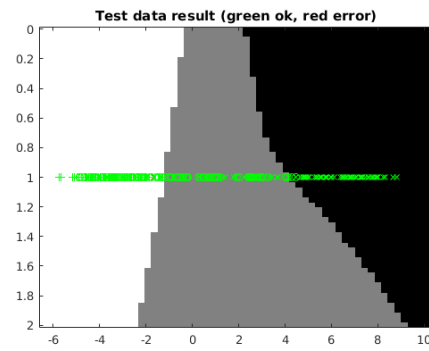


Figure 23: Multi-layer network result



**Dataset 4**

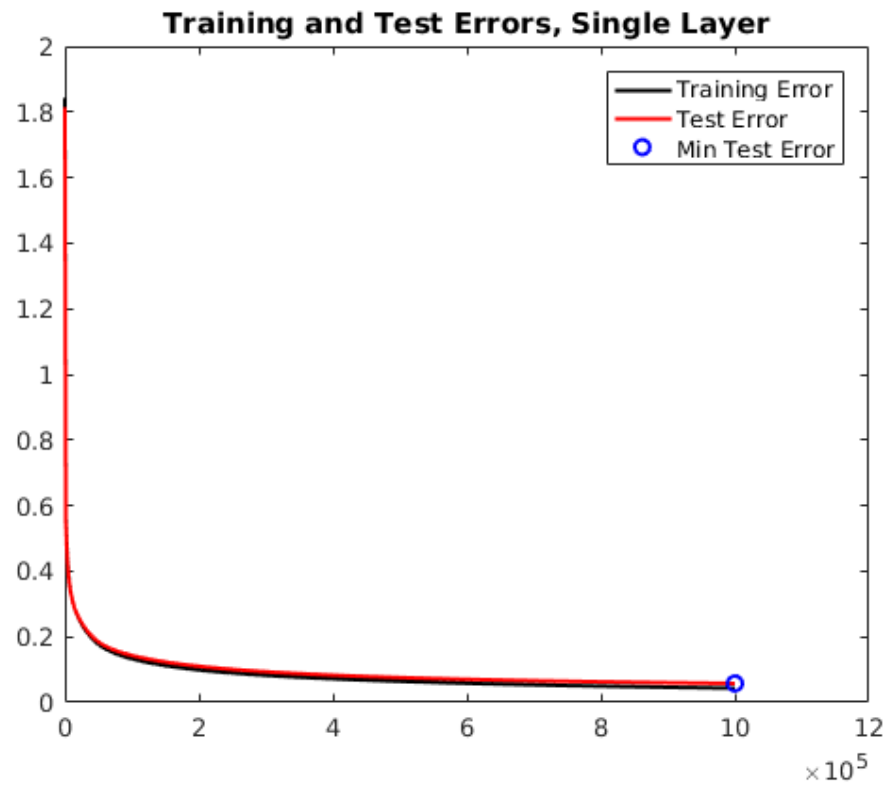
This dataset is a bit different, we are not dealing with only two or three dimension but 64. Because of the added complexity we can't find good parameters through arbitrary guessing. Instead we set the number of hidden to ten times the dimension, ensuring we get a model complex enough to find most approximate polynomials in 64 dimensions.

**Parameters:**

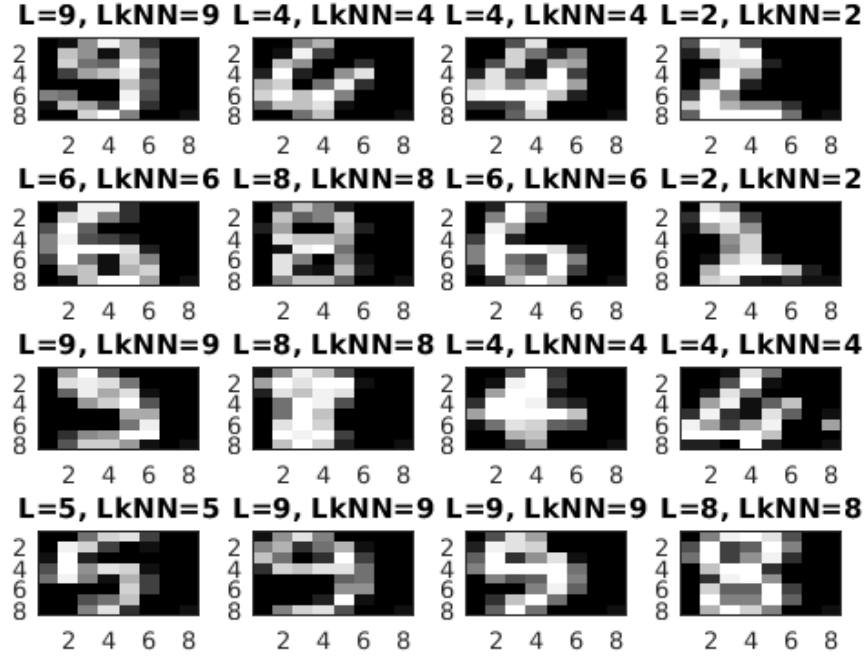
- *Hidden* = 640
- *Iterations* = 400000
- *learningrate* = 0.001
- *Accuracy* = 0.9150



Figur 24: Multi-layer network error



Figur 25: Multi-layer network result



Non-generalizable dataset

If there's enough data points in order to find a reliable model for a unknown function we say that the model we get is non-generalizable.