TBMI26 Neural Networks and Learning Systems
Assignment 1 - Supervised Methods

Linus Ahlenius - linah499@student.liu.se
Johan Lindström - johli160@student.liu.se

February 10, 2017

1 Introduction

This document is the first of three lab reports in the course Neural Networks and Learning Systems (TBMI26), held at Linköping University. The purpose of this lab is to implement and evaluate the performence of a few different classifiers on four sets of data. The classifiers implemented are kNN(k Nearest Neighbours) algorithm and two neural networks, both trained with backpropagarion. One of the neural networks is a single layer network and one is a network with one hidden layer.

2 Data set

Four data sets are given for the first lab. In this section they are described and a discussion about what kind of classifier would be required to separate the different classes are held.

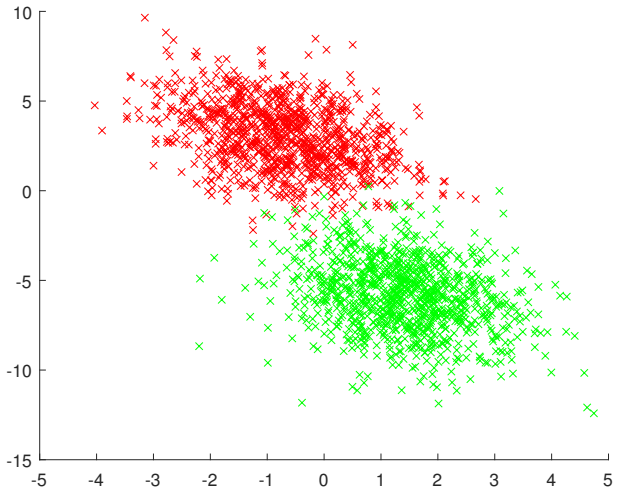

Figure 1: Data set 1

Data set 1, shown in figure 1, consists of two classes with two features. This problem is linearly separable since a straight line can be drawn between the bulks of the two classes.

Figure 2 shows data set 2. Like data set 1 this set consists of two classes with two features. However, unlike the first set sample points no longer shows two bulks with data. It shows one bulk of one class and the sample points from the other class surrounding it. There are no way to draw a single straight line to separate the two classes. With two straight lines and a decision tree it could be possible, but those classifiers are neither linear, nor for this lab. A non-linear

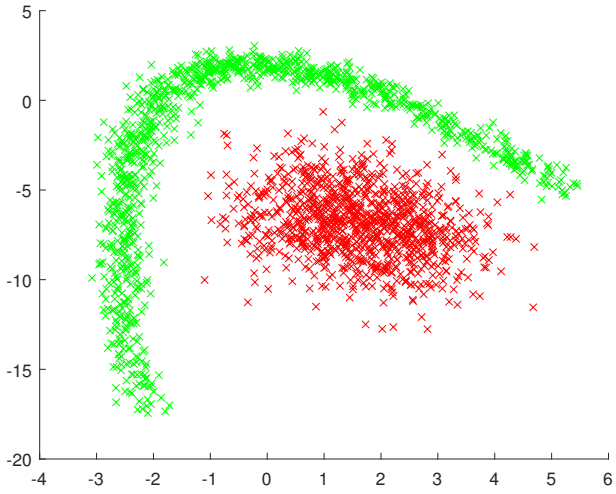

Figure 2: Data set 2

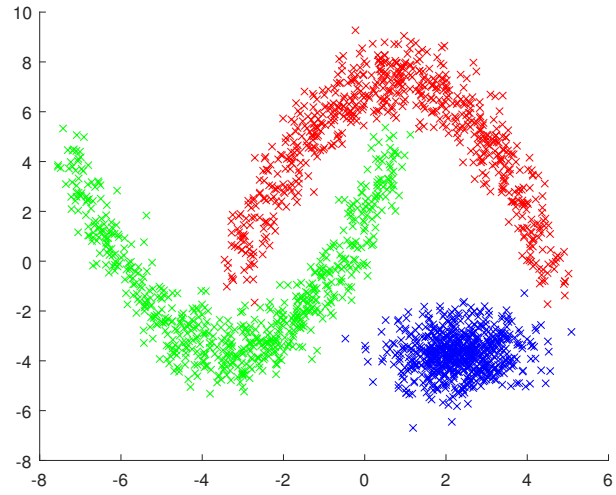

Figure 3: Data set 3

classifier is thereby needed.

Data set 3 consists of data with two features and three classes. That set is shown ni figure 3. Like data set 2 it needs non-linear classifiers to distinguish all three data sets. Although, if the class depicted in green weren't present a

linear classifier would be enough to separate the other two.

The fourth set, data set 4, is an OCR set consisting of 10 classes with 64 features. OCR stands for Optical Character Recognition and the characters to classify are the numbers 0-9. The features are extracted from pictures of 32x32 pixles. The feature exctraction is done by pre-processing. On the 32x32 pictures a grid of non-overlaping 4x4 squares are layed out. In each 4x4 squares the number of pixles filled in are counted and from that a value between 0 and 16 is returned. This reduces the amount of features and also "blurs out" the picture. One might think this is bad, but it actually makes the problem more general and in the end the classifiers more robust to variations. If a blurring would not be done the classifiers might get very specific that a few pixles has to be filled, when what is really interesting is to recognize a general pattern for how each number is written.

3 kNN algorithm implementation

The algorithm looks at k nearest neighbours of the data point we want to classify. Between these neighbours a majority voting is calculated. The class that most of the neighbours is classified as will be the class we classify the data point of interest. We handle ties by classify the data point as the nearest neighbour. A tie can occur for instance when k = 2 and number of classes that exists is 2.

By using cross validation we can select the best k for each data set. In the cross validation algorithm the data gets divided into three folds. We calculate the accuracy for a specific k three times. For each of the calculation a new fold is used as the test data and the other two as training data. The average accuracy is the measurement that we use to compare different k. In figure 4 for each data set the average accuracy for different k is shown. From the graph we can see a general pattern of lower k being better than higher k. We believe that models with low k works so well because the data is very distinctly divided, there are no outliers. Should the data points overlap more a higher k might be better. In general a higher k should create more smooth zones while lower k could induce overfitting, for example for outliers.

4 Backpropagation implementation

The initial weights of the classifiers are randomized. The network then classifies a batch of training data. The resulting classification is compared to the correct classification and the error is propagated back through the system. The new weights are calculated from a gradient decent calculation. For a single layer network the formula is shown in equation 1 where W are the weights, η is the learning rate, ϵ the error, X are the feature inputs, Y are the correct classifications and N is the number of samples.

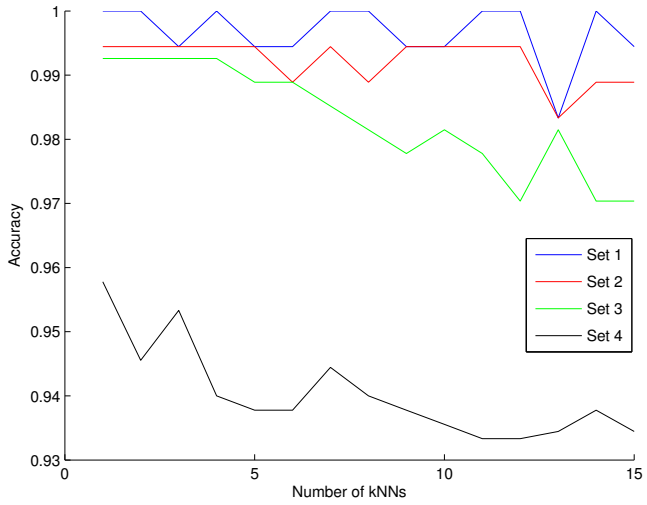

Figure 4: Cross validation

$$W_{t+1} = W_t - \eta \frac{d\epsilon}{dW}, \qquad \frac{d\epsilon}{dW} = \frac{2}{N}(W^T X - Y)X \tag{1}$$

The equations for two layer networks are presented in equation 2 - 4. W are the weights for the output layer and V are the weights for the hidden layer, Z is the classification from the classifiers, H is the input to the output layer/output from the activation function, σ is the activation function, in our case $tanh$ and S is the output from the hidden layer before the activation function.

$$W_{t+1} = W_t - \eta \frac{d\epsilon}{dW} \qquad V_{t+1} = V_t - \eta \frac{d\epsilon}{dV} \tag{2}$$

$$\frac{d\epsilon}{dW} = \frac{2}{N}(Z - Y)H \tag{3}$$

$$\frac{d\epsilon}{dV} = \frac{2}{N}W(Z - Y)\sigma'(S)X \tag{4}$$

For our implementation we also used a momentum term for calculating the new weights. This is just an additional term with a weight which adds to the gradient. This made us able to reduce the step size and still take large steps where we didn't require high precision. By implementing this we got the speed of a large step, with the precision of a small step close to the optimum.

Since we knew from observing the first three data sets that the first set was linearly separable we used the single layer network to classify data there. By observing data set two and three it was obvious that something non-linear was needed so we used the multi-layerer network. The single layer network never converged for the fourth data set which made the choice there easy.

Data set	hidden neurons	iterations	learning rate	momentum	accuracy	classifyer
1	-	20	0.01	0.75	99.4%	single layer
2	4	400	0.005	0.6	99%	multi-layer
3	6	1000	0.03	0.6	99.1%	multi-layer
4	50	1000	0.005	0.7	96.7%	multi-layer

Table 1: Multilayer accuracy for each data set

In general the choice of parameters were an iterative process. Since we implemented the momentum term we could get away with quite small steps, but we also got a new parameter to configure. We noticed pecks in the error graphs if the momentum term was too large. That meant that we often jumped over the optima and had to "turn around" and go down again. By reducing the momentum term these spikes were reduced in both height and intervals, but more iterations were then, understandably, required.

If the classifiers never converged (the error got larger and larger) we had to reduce the step size, since that meant that we were jumping over the optima each time to a place with a larger gradient each time, we reduced the step size. If we never found an optima, but the error kept getting smaller, we would increase the step size, the momentum term or number of iterations to be able to find it.

As a starting point we used as many neurons as there were features and iterated until we met the criterias. For data set 4, however, we played around a little and found that significantly less hidden neurons were needed. We believe this is due to that many features, for examples the pixels in the corners, are irrelevant to classify these kinds of problems.

In figure 5 and 6 we compare how the neural networks solutions changes with different amount of training data. Figure 5 is an example of an non-generalisable backpropagation solution, for this solution few data samples were used for training. We can see that this amount of data samples is to low to understand the generalized pattern of the data. This network will predict wrong many more times than the solution found in figure 6 were we used all data we were given to train the network.

In general it is important to have a lot of training data or rather, it is important to have data that gives a lot of information about the classification problem. Samples close to another class gives more information on where the boundary between the classes should be, compared to a sample in the middle of a bulk.

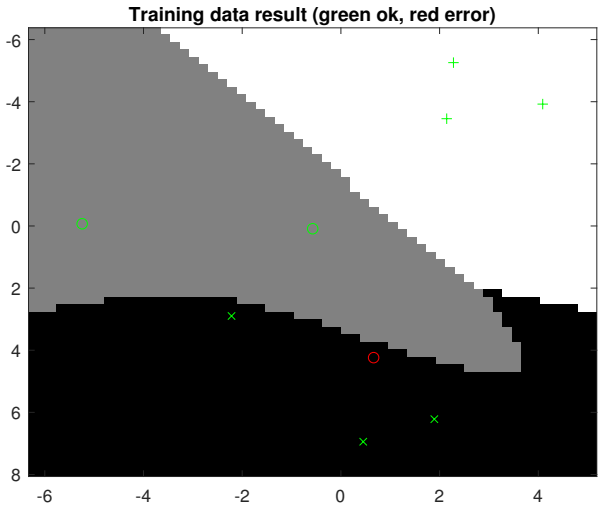

Figure 5: None generalizable

Figure 6: Generalizable

5 Discussion and conclusion

The kNN algorithm performed remarkably well for all four data sets. We have a theory that this has a lot to do because of how well separated the different classes are. Had the classes been closer to each other, with some overlap kNN,

at least for low k's, would probably be more inaccurate.

A significant pro for the kNN algorithm is that it requires no training of a model and is easy to implement. However, the algorithm itself is not very scalable. It requires you to save all the data points which makes it cumbersome to classify for large data sets. A possible improvement for kNN, that would make it more scalable, could be to remove some data points that carry less information, data points further away from the decision boundary.

Neural networks can be both linear and non-linear depending on if activation functions are used. They are a powerful too that comes with a cost. A neural network has to be trained which can take a lot of time and computer power. There also seems a bit of a mystery how many layers, and nodes in each layer that should be used. This makes experience of the user quite important.

The single layer neural network only works for linearly separable problems, which made us have limited use of it. The multi-layer network was of greater use. In reality a lot of problems are of greater complexity, like the OCR problem. A multi-layer neural network can often deal with those, given that good features are selected.

The results for the three first data sets using neural network was good, as expected since the data was very separated. The fourth data set was a lot less intuitive regarding its solvability. Here the networks shows its potential. To describe the reliationships that forms a specific number feels quite hard to produce, but the neural network are able to figure out these patterns. The fourth data set were preprocessed to make the problem more robust (and less data intensive). This feels reasonable since everyone doesn't write the same. We are out to identify the general pattern, rather than what is in specific pixels.

Another way of improving the results could be to remove potential outliers. Although, for the first three data sets there does not seem to be many, but for the fourth set there could be. A way of reducing the computational complexity would be to reduce the dimensionality, the number of features. Again, for the first three sets it's not possible, but we would be surprised if that can not be done for the fourth set. There are likely pixels around the edges that are barely touched by anyone writing the numbers.

We found two good ways of improving the convergence time and accuracy. The first one was to introduce a momentum term. This made us able to take larger steps before we get close to the optima. Around the optima the gradient flattened out and we could take smaller steps.

The second thing we found out was how significant of a role the initiation of the weights played. We started out with the weights being randomized between $\{-1, 1\}$. The convergence speed for these initial weights were orders of magnitude larger, compared to initiate the weights between $\{-0.05, 0.05\}$. We found out that this has to do with the gradient far out on the edges of the tanh-function is more or less flat.