Neural Networks
Final assignment

Anna Katrine Jørgensen and Indra den Bakker
GROUP 26

October 26th, 2014

1 Introduction

During the May 4th National Remembrance Day in 2010 a person started shouting, resulting in a large panic (http://www.youtube.com/watch?v=VfrDGjS3fOo). In this assignment we will try to predict the movement of people after the event, i.e. shouting of a person, occurred. We were given a data set with all relevant points (coordinates of the source, fences and building corners) and of the movement of 35 people during 47 time steps. The first location, i.e. at time step 1, and the last given location, i.e. at time step 47, is plotted in Figure 1.

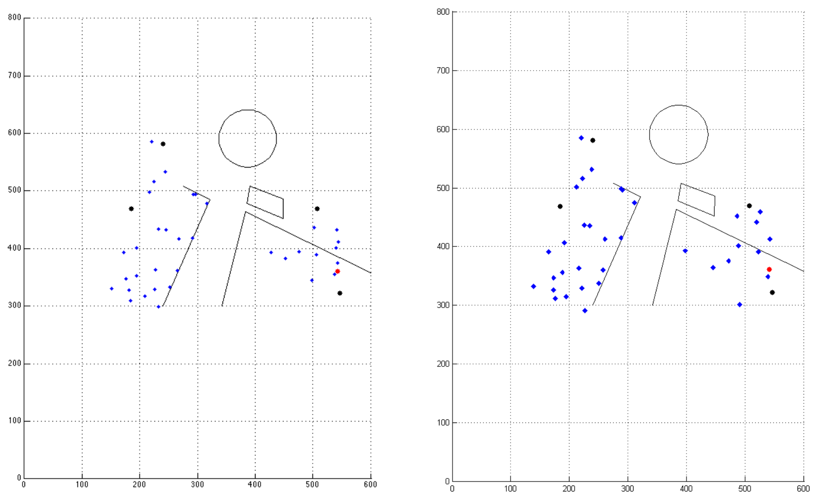

Figure 1: Left: Location on time step 1; Right location on time step 47

2 Neural Network - Multilayer Perceptron

The single-layer perceptron is incapable of solving tasks that are not linearly separable, since it is restricted to linear calculations. However, by combining several layers in one perceptron, the network becomes capable of solving complex problems. It is this combinatory effect we explore in the following for predicting the movements of individuals on the Dam Square. We use MATLAB R2014a for this purpose.

3 Multilayer Perceptron Background

Since it is expensive to design and create networks by hand, we want a generic network that learns from the training data it is given. A multilayer perceptron is an example of such a feed-forward network, which maps inputs to outputs by the use of weights. The MLP consists of one or more hidden layers, and is therefore considered a deep neural network. This means that the network can model complex non-linear relationships and the network is capable of modelling more abstractions and connections in the data. Besides the hidden layers, the neural net contains an input layer and an output layer.

Each of the layers contain multiple nodes, and the layers are fully connected to each other. This resembles the processes in the human brain, in which information from one neuron is fed to other neurons via synaptic processes.

Each node in the hidden layers and the output layer is to be though of as a neuron with an activation function, which defines the output of that node given the input it receives. It provides a representation of the activation potential of that neuron and is in its simplest form binary; either the neurone fires, or it does not. Most often, however, the activation function with be a normalisable sigmoid function or another from which a slope can be derived. This is necessary for the process of readjusting the weights to make the model fit the data, as will be explained in the following.

In the MLP network, each connection is assigned a weight. These weights are tweaked and adjusted so the network produces outputs that correspond to the desired targets. Since the network is given the desired outputs, the network learns in a supervised manner; it does not only learn from the data, but also from the targets it receives, which it uses to optimise its parameters. This process of learning the parameter optimisations is called backpropagation.

Backpropagation is a measure for optimising the weights connected to each node in each layer and therefore relies on the error produced by the model with respect to the input and the desired output (target).

The error measure for the output layer is defined as: $e_j(n)=d_j(n)-y_j(n)$, where *d is* the desired output and *y* is the output of the network with respect to the *n*th datapoint.

We apply several different techniques to minimise the error of the model with respect to the labels we give as targets. The techniques we apply are: the Levenberg-Marquardt technique, resilient back propagation and three different versions of gradient descent, namely regular gradient descent, gradient descent with momentum and scaled gradient descent.

All methods attempt to find the minimum of the curve of the error function. Even though they find a local minimum, they may not find the global minimum of the curve, which is the lowest possible error. There are various ways to overcome this issue; random initialisation is often employed to enhance the chance of finding the global minimum, the speed of learning is also important since it is essentially the rate of convergence with the global minimum from the current location, and momentum can enable the network to surpass local minima and instead move toward and end in or at the global minimum. Our results for these approaches are reported below.

4 Neural Network configuration

4.1 Input and output

For predicting the movements of the individuals on Dam Square after the shouting, we train our MLP network to predict the location of a individual at the following time stamp. We try different inputs for the network to see what works better. First we try as input the distance to the shouter and the angle. Next we will try the same two inputs plus the time label. As output we use the location of the next time step.

4.2 Training and testing

As an error measurement we use the mean square error and we perform 10-fold cross-validation to prevent overfitting and to ensure that the minimum is not a local minimum. This is done via randomised initialisations and 6 validation checks.

4.3 Training algorithm

For this assignment we will use different training algorithms that has been discussed during the lectures and that are implemented in MATLAB. We use the following algorithms: Levenberg-Marquardt,Gradient descent, Gradient Descent with Momentum, Resilient Backpropagation and Scaled Conjugate Gradient.

5 Preprocessing data

5.1 Extending data

The data set of 35 people is relatively small to train on for a neural network. That is why we need to extend the data set given. We choose to extend it by adding 4 data points for every given data point. We do this by creating a point 2 pixels left from the original, one 2 pixels to the right, one 2 pixels above and one 2 pixels below (see Figure 2). The movement of these new data points are exactly the same as for the copied data point. Herewith we create 140 new data points per time steps, so in total we now have 8050 data points.

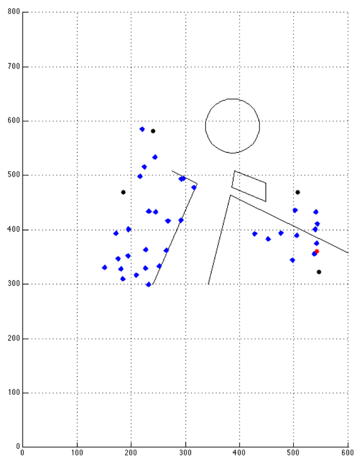

Figure 2: Extended data set of 175 data points at time step 1

5.2 Analysis

If we look at the given data we see that the people that are closest to the source are moving in the beginning and people that are further away start moving later (something that has been explained in [1] by the characteristics of people and how they react on each other). We visualised the path of the 35 persons in Figure 3. At first glance it looks like most people are moving straight away from the source. When we have a closer look we see that most people move away with a certain angle, for example away from the Dam Monument or buildings.

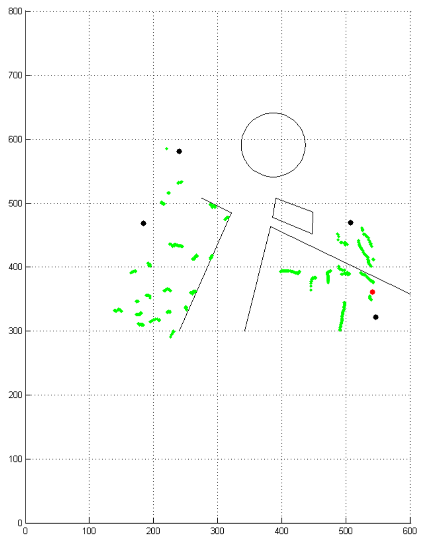

Figure 3: Path over time of original data

5.3 Transforming data and normalise

The determine the distance to the source and the angle we transform the data from Cartesian coordinates to polar coordinates. First we subtract the coordinates from the source from all other data points, this results in having the source as origin. Now we can transform the coordinates to polar coordinates (theta and rho) and we now what the distance from each data point to the source is (rho) and which angle the vector has (theta).

The performance of neural networks can be increased by normalising the data. We do this by calculating the standard deviation and mean for the theta and rho. We multiply the values with the standard deviation and we subtract the mean from the result.

We use the transformed data as input and output (for training and test data). For visualisation we transform the results back in Cartesian coordinates.

6 Experiments

First we only use the theta and rho as input values for our neural network. We experiment with different number of hidden layers (from 1 to 3) and with different number of hidden neurons per hidden layer (from 2 to 100). We start with the Levenberg-Marquardt algorithm. The results are given in Table 1.

Algorithm	Input	Architecture (#hidden layers [#hidden neurons per layer])	MSE (10-fold cross validation)
Levenberg-Marquardt	theta, rho	1 [2]	0.0019
Levenberg-Marquardt	theta, rho	1 [5]	0.0019
Levenberg-Marquardt	theta, rho	1 [10]	0.0019
Levenberg-Marquardt	theta, rho	1 [20]	0.0019
Levenberg-Marquardt	theta, rho	1 [50]	0.0021
Levenberg-Marquardt	theta, rho	1 [100]	0.0021
Levenberg-Marquardt	theta, rho	2 [2 2]	0.0022
Levenberg-Marquardt	theta, rho	2 [5 5]	0.0020
Levenberg-Marquardt	theta, rho	2 [10 10]	0.0019
Levenberg-Marquardt	theta, rho	2 [20 20]	0.0019
Levenberg-Marquardt	theta, rho	2 [30 30]	0.0020
Levenberg-Marquardt	theta, rho	3 [2 2 2]	0.0176
Levenberg-Marquardt	theta, rho	3 [5 5 5]	0.0019
Levenberg-Marquardt	theta, rho	3 [5 10 5]	0.0021
Levenberg-Marquardt	theta, rho	3 [10 10 10]	0.0019
Levenberg-Marquardt	theta, rho	3 [10 20 10]	0.0019
Levenberg-Marquardt	theta, rho	3 [20 30 20]	0.0019

Table 1: Results for Levenberg-Marquard algorithm with theta and rho as input

The performance and gradient of the neural network for the Levenberg-Marquardt algorithm with 1 hidden layer with 10 hidden neurons is given in Figure 4. The gradient does not converge.

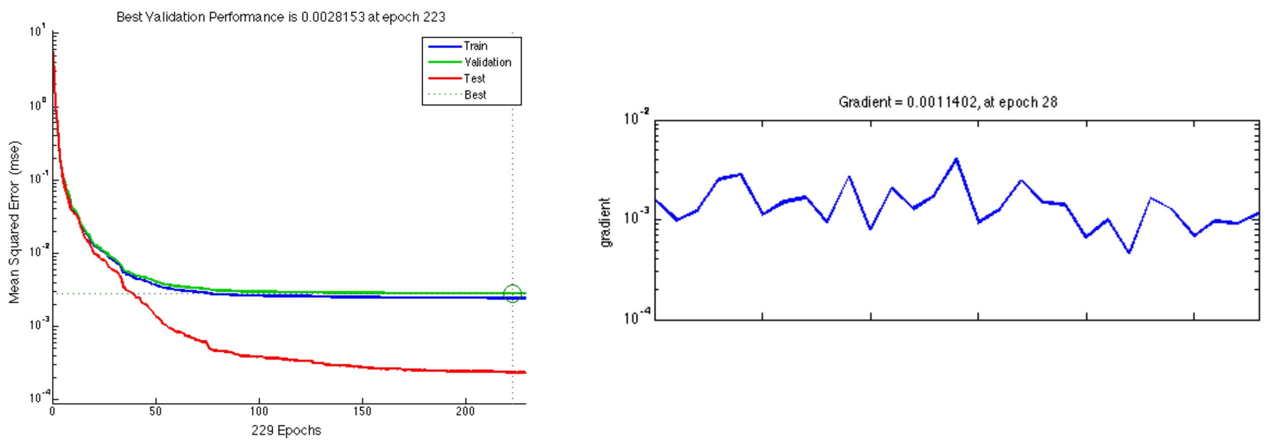

Figure 4: Performance and gradient for Levenberg-Marquardt algorithm with 1 hidden layer with 10 hidden neurons (theta, rho)

For these settings the predicted locations (blue dots) given the locations at time step 46 are plotted in Figure 5 next to the actual locations at time step 47 (green cross) for all 41 random positions as input. As an comparison we plotted the results of the network with 2 hidden layers with both 2 hidden neurons (Figure 6) and our worst result (3 hidden layers with 2, 2 and 2 hidden neurons respectively) in Figure 7.

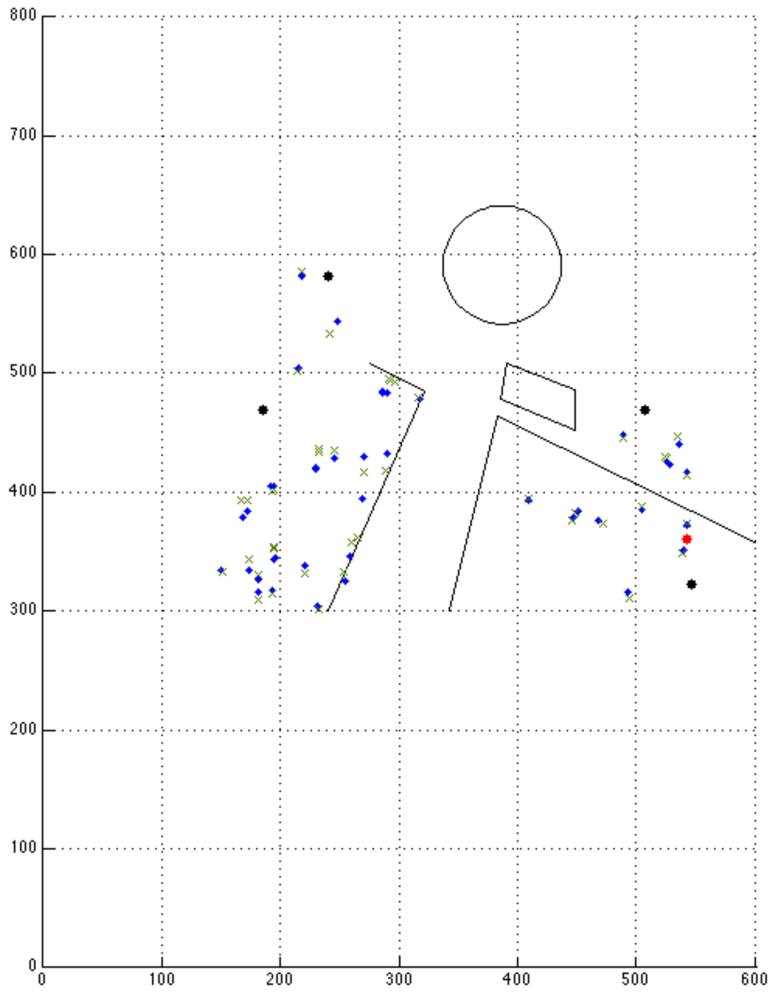

Figure 5: Predicted locations (blue) vs actual locations (green) for 41 random inputs (Levenberg-Marquardt with 1 hidden layer with 10 hidden neurons (theta,

The results between the Levenberg-Marquardt algorithm with 1 hidden layer with 10 hidden neurons and with 3 hidden layers with 2, 2 and 2 hidden neurons respectively are big as can easily seen visually. We see that performance of the neural networks with different parameters, i.e. different number of hidden layers and hidden neurons, are close to each other. It looks like increasing the number of hidden layers or number of hidden neurons can result in overfitting. We now will experiment with other algorithms. Some results are given in Table 2.

Algorithm	Input	Architecture (#hidden layers [#hidden neurons per layer])	MSE (10-fold cross validation)
Levenberg-Marquardt	theta, rho	1 [10]	0.0019
Gradient Descent	theta, rho	1 [10]	0.0020
Gradient Descent with Momentum	theta, rho	1 [10]	0.0021
Resilient Backpropagation	theta, rho	1 [10]	0.0022
Scaled Conjugate Gradient	theta, rho	1 [10]	0.0019

Table 2: Results other algorithms with theta and rho as input

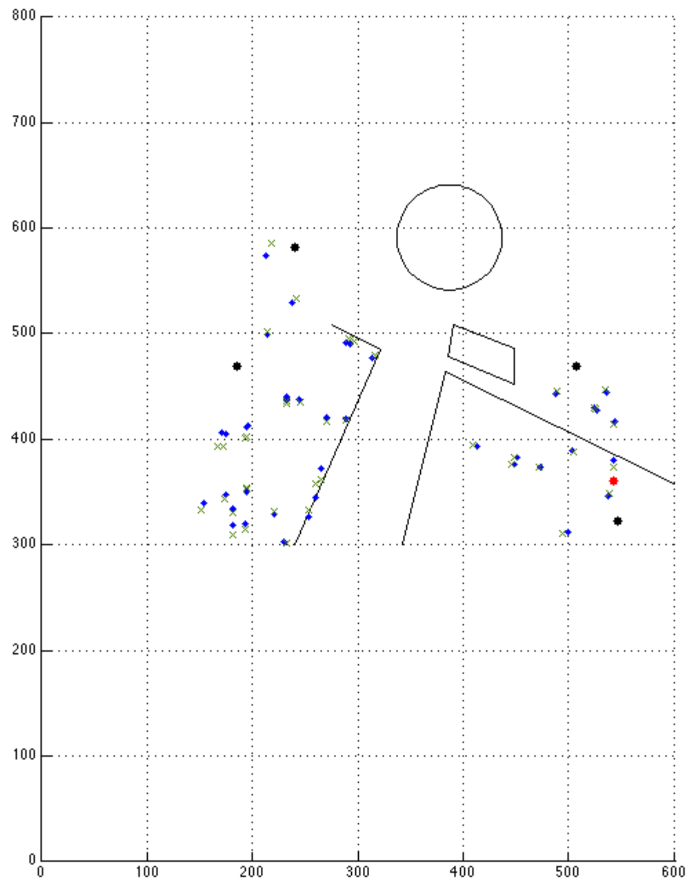

Figure 6: Predicted locations (blue) vs actual locations (green) for 41 random inputs (Levenberg-Marquardt with 2 hidden layer with 2 and 2 hidden neurons (theta, rho))

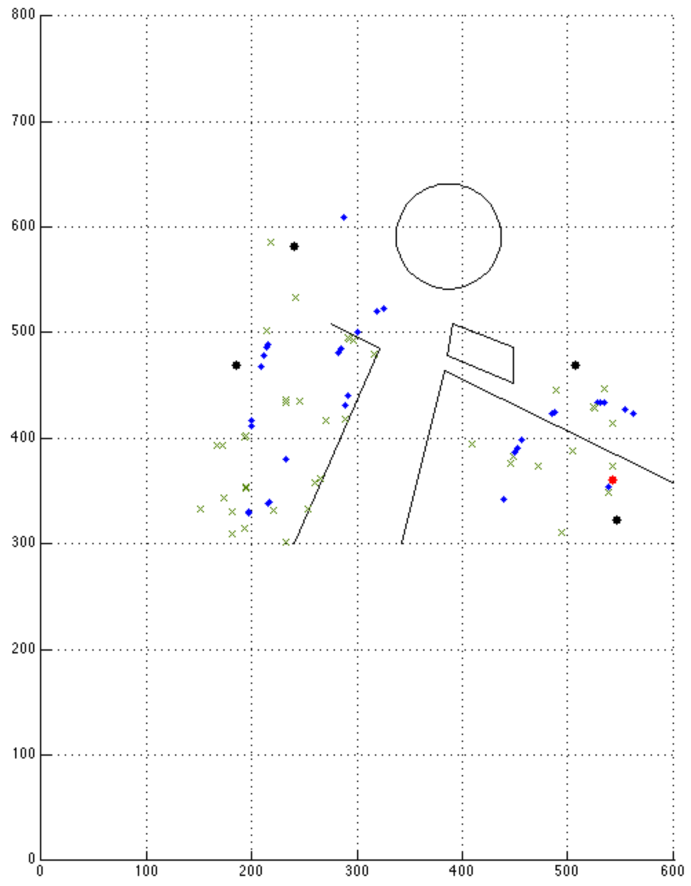

Figure 7: Predicted locations (blue) vs actual locations (green) for 41 random inputs (Levenberg-Marquardt with 3 hidden layer with 2, 2 and 2 hidden neurons (theta, rho))

The results with other algorithms do not show significant differences. That is why we will use the Levenberg-Marquardt algorithm for our next experiment for ease of comparison. We now will include the tilmestep in the input next to distance and angle. The results for this experiment are given in Table 3.

Algorithm	Input	Architecture (#hidden layers [#hidden neurons per layer])	MSE (10-fold cross validation)
Levenberg-Marquardt	theta, rho, time	1 [2]	0.0019
Levenberg-Marquardt	theta, rho, time	1 [5]	0.0019
Levenberg-Marquardt	theta, rho, time	1 [10]	0.0019
Levenberg-Marquardt	theta, rho, time	1 [20]	0.0020
Levenberg-Marquardt	theta, rho, time	1 [50]	0.0020
Levenberg-Marquardt	theta, rho, time	1 [100]	0.0022
Levenberg-Marquardt	theta, rho, time	2 [2 2]	0.1066
Levenberg-Marquardt	theta, rho, time	2 [5 5]	0.0019
Levenberg-Marquardt	theta, rho, time	2 [10 10]	0.0020
Levenberg-Marquardt	theta, rho, time	2 [20 20]	0.0019
Levenberg-Marquardt	theta, rho, time	2 [30 30]	0.0018
Levenberg-Marquardt	theta, rho, time	2 [40 30]	0.0019
Levenberg-Marquardt	theta, rho, time	3 [2 2 2]	0.3059
Levenberg-Marquardt	theta, rho, time	3 [5 5 5]	0.0019
Levenberg-Marquardt	theta, rho, time	3 [5 10 5]	0.0020
Levenberg-Marquardt	theta, rho, time	3 [10 10 10]	0.0019
Levenberg-Marquardt	theta, rho, time	3 [10 20 10]	0.0018
Levenberg-Marquardt	theta, rho, time	3 [20 30 20]	0.0018
Levenberg-Marquardt	theta, rho, time	3 [30 30 30]	0.0021

Table 3: Results for Levenberg-Marquard algorithm with theta, rho and time as input

These results show that the performance of the neural network increases slightly when the time steps are included in the input. Especially when more hidden neurons and hidden layers are added to the network (we also tried 4 hidden layers, but this did not result in lower MSE). The best results (a MSE of 0.0018) were obtained with a network with 2 hidden layers with 30 and 30 hidden neurons; a network with 3 hidden layers with 10, 20 and 10 hidden layers; a network with 3 hidden layers with 20, 30 and 20 hidden neurons.

The performance and gradient for the network with 2 hidden layers and 30 and 30 hidden neurons are plotted in Figure 8.

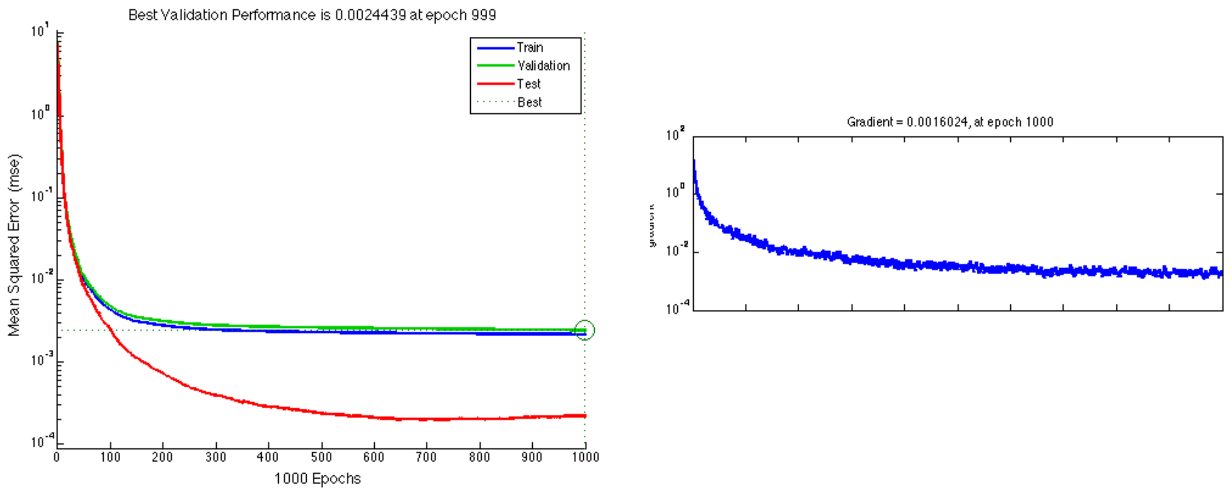

Figure 8: Performance and gradient for Levenberg-Marquardt algorithm with 2 hidden layers with 30 and 30 hidden neurons (theta, rho, time)

In Figure 9 we plotted the predicted values from the neural network with 2 hidden layers with both 30 hidden neurons and as input theta, rho and time.

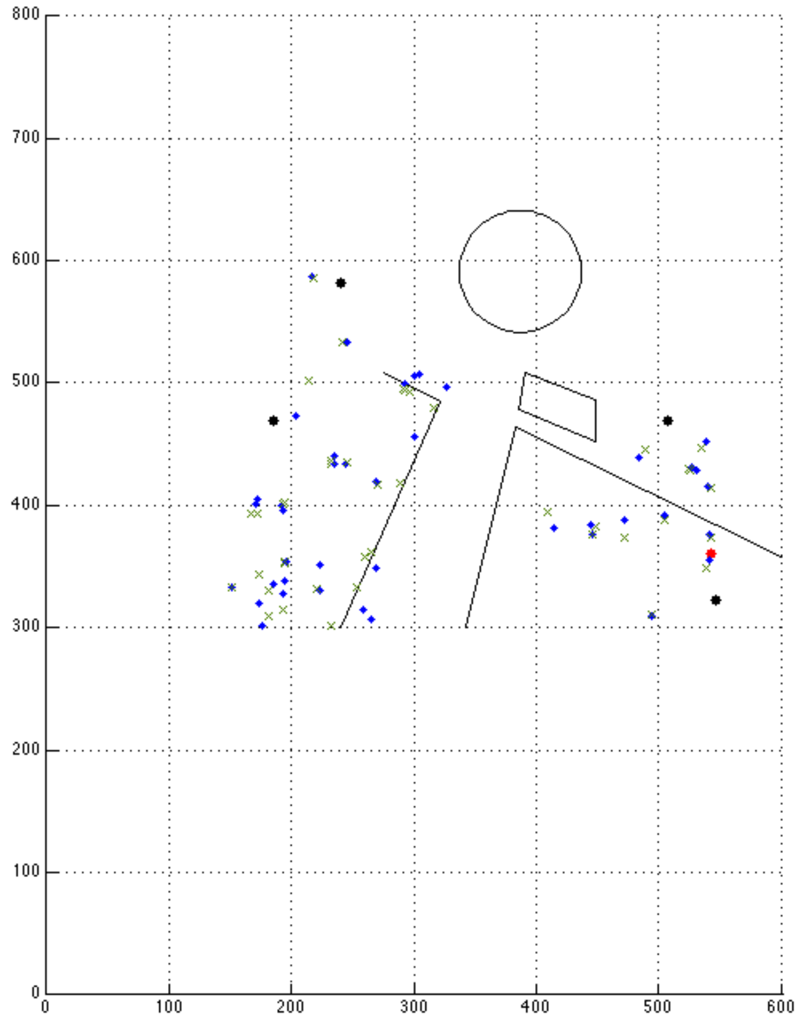

Figure 9: Predicted locations (blue) vs actual locations (green) for 41 random inputs (Levenberg-Marquardt with 2 hidden layers with 30 and 30 hidden neurons (theta, rho))

7 Conclusion

From our experiments we can conclude that none of the different algorithms performed significantly better. Furthermore, increasing the number of hidden layers or hidden neurons does not improve the predictions significantly. Even more, it looks like increasing the number of hidden neurons too much results in overfitting. When adding time as a input variable, we got the best result when more hidden neurons are used and multiple (i.e. two or three) hidden layers.

We got our best result with 2 hidden layers with 30 and 30 hidden neurons. This network gave a 10-fold cross validation MSE of 0.0018 on the transformed and normalised data. The predicted values of this network versus the actual locations of time step 47 (with time step 46 as input) are plotted in Figure 9.

8 Discussion

Why adding time did not not improve the results significantly?

Adding time as variable in the input did improve the network slightly. We think the results are not improved a lot because the time step and distance to source are not independent measures. People that are close run away from the source, resulting in a larger distance at a latter time step. How farther away from the source how smaller the movement will be, in this case implicating a later time step will result in a smaller movement. So including the time step can overfit the neural network faster.

9 Future work

During our experiments we came up with ideas to improve our neural network, but we did not have enough time to implement these possible improvements. Below we list the ideas for extending this research:

- *Try other methods:* We chose to implement the multilayer perceptron because it is a useful method for supervised learning in pattern recognition problems. A similar, and perhaps simpler, alternative would have been to implement a Support Vector Machine, however, we wanted to investigate the properties of the MLP in this assignment and have therefore experimented with the optimisation process and the number of hidden neurons and hidden layers to investigate the effects of these in this problem.
- *Take into consideration obstacles (buildings and fences)*: A important variable for predicting the movement of people could be other obstacles. This could for example limit the movement towards a direction. From the data we see two persons that are already very close to a fence, after 47 time steps these persons are further away from the source, but their direction is limited by the fence.
- *Create more data points*: To increase the accuracy of the predictions one could increase the number of data points. Maybe with data of people that are further away from the given 35 people.
- *Take into account people reacting to people close to each other*: It might be an idea to take into account the (number of) people around a person and how they are moving/reacting.
- *Add more hidden layers and neurons*: We tried to improve our neural network by increasing the number of hidden neurons and hidden layers with steps. Another combination that has not been tried could have preformed better.
- *Extend research and try to predict time step 47 from time step 1*: In this research we tried to predict the movement of people in the next time step. It might also be interesting to predict the final position of people over time (for example given the current position - and time - what will the position of the person be over a minute).

References

1. Bosse, T., Hoogendoorn, M., Klein, M. C. A., Treur, J., Wal, C. N. van der, and Wissen, A. van. Modelling Collective Decision Making in Groups and Crowds: Integrating Social Contagion and Interacting Emotions, Beliefs and Intentions. Autonomous Agents and Multi-Agent Systems Journal (JAAMAS), volume 27 (1), 2013, pp. 52-84.