VU AMSTERDAM, FACULTY OF EXACT SCIENCES

Neural Networks : Final Assignment

Jorma Nieminen, Cyril Quinton, Group 24

October 29, 2014

1 INTRODUCTION

In the assignment a neural network was used to predict the movement of a person in a groud as panic occurs. The data for the task was gathered from this type of situation that occurred in Amsterdam on 4th of May 2010 during the national remembrance of the people that died during wars or peacekeeping missions. This data was then enriched and modified to input vectors. Radial basis function networks (RBF) with different amounts of hidden units were trained and evaluated with different types of input vectors and comparison of these networks was made based on the error value obtained from 10-fold crossvalidations.

2 INPUTS AND OUTPUTS

Inputs used to train and validate the network were collected from all the points in the original dataset except for the last line. This meant that $35 \times 46 = 1610$ inputs were extracted without counting the expanded points discussed in section 3. Similarly the outputs contained all the original points except for the first line. The input output mapping was thus

$$Env_x(t) \mapsto x(t+1), \tag{2.1}$$

where $Env(t)$ is the environment, i.e. the input vector, and $x(t)$ is the position of the point x at timestep t.

2.1 REDUCTION OF THE INPUTS

Four different input vectors were used to train and evaluate the networks. The first vector **L** used coordinates of the ends of the lines that represented the fences. The second vector **P** used the closest points to those lines, measured from the simulated point, which reduced the amount of inputs by 16. Third and fourth vectors, $\mathbf{L_t}$ and $\mathbf{P_t}$, used the information about the lines the same way as **L** and **P** but in addition all the points were transformed by shifting and rotating the coordinates of the scene which reduced the amount of inputs by 3. The number of different inputs are shown on table 2.1. The output vector $\mathbf{O_t}$ for the $\mathbf{L_t}$ and $\mathbf{P_t}$ was also transformed with same methods as the input vectors.

Table 2.1: Inputs by tested input vectors

Input	**Number of values**	**L**	**P**	$\mathbf{L_t}$	$\mathbf{P_t}$
Timestep from the start of the panic	1	x	x	x	x
Coordinates of the source of the panic	2	x	x		
Coordinates of the circle	2	x	x	x	x
Coordinates of the corners of the buildings	8	x	x	x	x
Coordinates of the simulated point	2	x	x		
Coordinates of the ends of the lines (fences)	32	x		x	
Coordinates of the closest points on the lines	16		x		x
Distance to the source of the panic	1			x	x
All	64	47	31	44	28

2.2 TRANSFORMATIONS

With the vectors $\mathbf{L_t}$, $\mathbf{P_t}$ and $\mathbf{O_t}$the coordinates were first shifted so that the person to be simulated was placed at the origin. This was achieved with equation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} P_x \\ P_y \end{bmatrix} \tag{2.2}$$

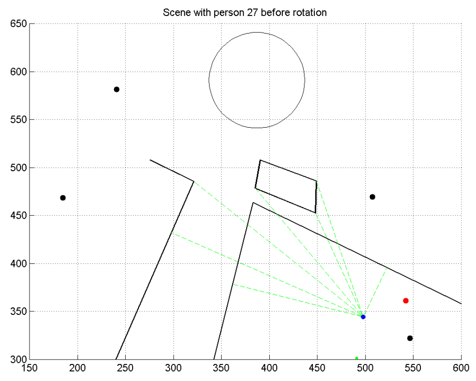
where (x, y) is a point in the vector to be transformed and (P_x, P_y) is the point representing the person to be simulated.

After the shift, the coordinates were rotated so that the source of panic was placed on the positive x-axis. Rotation was calculated with equation

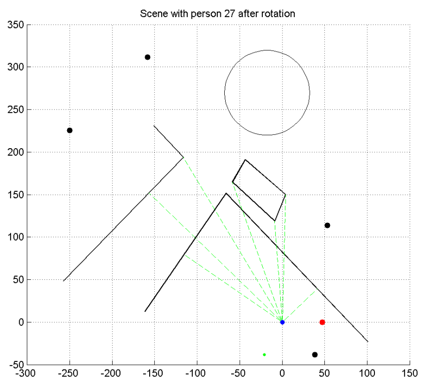
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y, \end{bmatrix} \tag{2.3}$$

where θ is the angle between the x-axis and the position vector of the source of the panic.

The shifts and rotations reduced the number of the inputs since the coordinates of the simulated person were not needed and the source of the panic could be expressed with only the distance value. Intuitively these coordinate changes could have helped the network to learn to the task more easily since all the points in the environment were relative to the person to be simulated. This way it might have been easier to determine the direction of movement by the locations of the possible pulling and pushing points. One might think that corners of the buildings acted as pulling points and the fences, represented as lines, were pushing the person away. An example of original scene and the rotated one can be seen in figures 2.1a and 2.1b respectively.



(a) Before the transformation of coordinates.



(b) After the transformation of coordinates.

Figure 2.1: Environment of the 27th person. The blue points in the pictures represents the location of the person, the red point is the source of the panic, the black points are the corners of the buildings, the circle is the circle in the Dam square, the lines are the fences and the green dashed lines are the vectors to the closest points on these fences.

2.3 Representation of the lines

Thinking about lines, i.e the fences, acting as pushing force towards the person to be simulated led to the alternative representation of the lines. Originally the lines were represented as real-valued vector of size four, i.e. as the end points of the lines. Another way to represent the information about the lines was to use only the closest points on the lines relative to the person. The closest points to the lines were first calculated with equation

$$\mathbf{a} - (\mathbf{a} - \mathbf{p}) * \mathbf{n})\mathbf{n}, \tag{2.4}$$

where **a** is the left end point of the line, **p** is the location of the person and **n** is the unit vector of line. If however this point was on the wrong side of an end point of the line, the relevant end point was selected. Unit vectors had to be calculated before the rotations discussed in section 2.2 because the directions of vectors change with rotation. Vectors from person to the closest points in lines can be seen in figures 2.1a and 2.1b.

3 Dataset

Dataset consisted originally of 35 points that represented people that moved in similar patterns during the panic. This dataset was enriched by creating at most 20 points around these original points.

3.1 DATASET ENRICHMENT METHOD

New points were generated from original points (parent points) at step 1 (t_0) with the following equation :

$$\begin{aligned} x^{'} &= N(x,3) \\ y^{'} &= N(y,3) \end{aligned} \tag{3.1}$$

where $x^{'}$ and $y^{'}$ are the coordinates of the new point, x and y are the coordinates of the parent point, $N(x,3)$ is a number picked from a Gaussian distribution with $mean = x$ and $sigma = 3$.
The new coordinates were generated with two constraints.

- cluster separability : A new point generated from a parent point could not be in the smallest square containing every offspring of another parent point. This constraint guarantied that clusters were separated, thus, each square region had his own movement pattern.
- identical relative line positions with the parent point: A new point point had to be on the same side of each line in respect with his parent. This was based on the assumption that people on the other sides of the fences would not behave similary.

Offspring had the same movement pattern as their parents because their relative distance from their parents were unchanged during every steps. For all steps :

$$\begin{aligned} x^{'}_{step+1} &= x_{step+1} + rx \\ y^{'}_{step+1} &= y_{step+1} + ry \end{aligned} \tag{3.2}$$

Where rx et ry are the relative distances of the offspring from his parent on the x axis and the y axis calculated at step 1.
A total of approximately 430 new points were created, the pseudode of the enrichment of the dataset is shown in algorithm 1.

4 MEASURING NETWORK PERFORMANCE

The network performance was evaluated for all the combinations of different types of input vectors, discussed in section 2, and the different amounts of hidden units. The range of amounts of hidden units was $[1,10]$. Root mean squared error (MSE) was used as the measurement for these evaluations. The error values were obtained by applying 10-fold cross-validation to the training and testing of the networks.

5 NETWORK ARCHITECTURE

The evaluated networks were radial basis function networks with different amounts of hidden units. A k-means clustering was used to find the centers of the network.

Algorithm 1 dataset enrichment method

```
$newPoints \leftarrow []$
for $i \leftarrow 1,20$ do                                      ▷ Expand the dataset at step 1 ($t_0$)
    for all $p \in Points$ do
        $correctPoint \leftarrow FALSE$
        while $correctPoint = FALSE$ do
            generate $x^{'}$ and $y^{'}$ from p
            if constraints respected then
                $correctPoint \leftarrow TRUE$
            end if
        end while
        update square size of p
        The new coordinates, the parent number and the relatives distance to the parents are added to newPoints
    end for
end for
for all $step \in Points$ do                                    ▷ Expand the dataset on other steps
    for all $p \in Points$ do
        for all $offspring \in newPoints$ do
            if $offspring.parent = p$ then
                generate the coordinates of the offspring at the step
                if constraints respected then
                    add the coordinates to newPoints
                else
                    Delete the offspring in newPoint
                end if
            end if
        end for
    end for
end for
add newPoints to Points
```

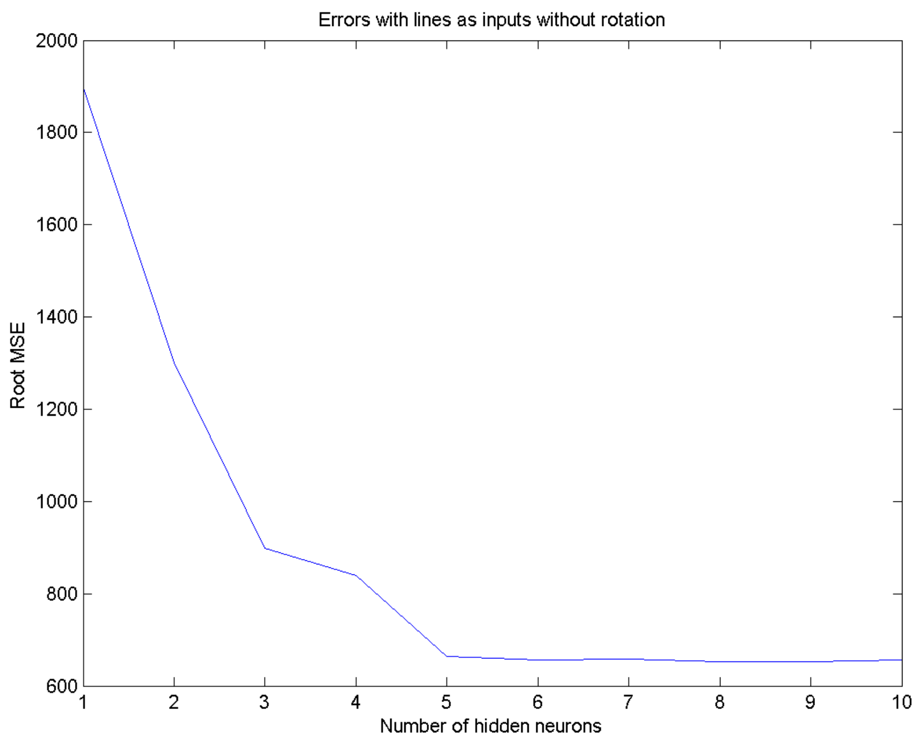


Figure 6.1: Root MSE depending on the number of hidden neurons, with the input **L** (the two end points of the lines in input without transformations).

Originally multilayer perceptrons were also used but after the expansion of the original dataset, discussed in section 3, the evaluation cycle of a network became too long and the final results were obtained only with RBFs.

6 Results

6.1 Inputs without transformations

The error rates were highest when using the input type **L** as shown in figures 6.1 and 6.7. Using more than 5 hidden units did not seem to have a large effect on the network performance with this input type.
The error rates observed with the input type **P** were also high as seen on figure 6.2. The difference of this type of input vectors from the type **L** was that error rate decreased when the number of hidden neurons increased and so the lowest root MSE was reached with 10 hidden neurons. Therefore adding more hidden units might have been useful with input type **P**.
The networks trained with the best learning rate between 0.01 and 0.5 were chosen to show the corresponding error rates (figure6.3).

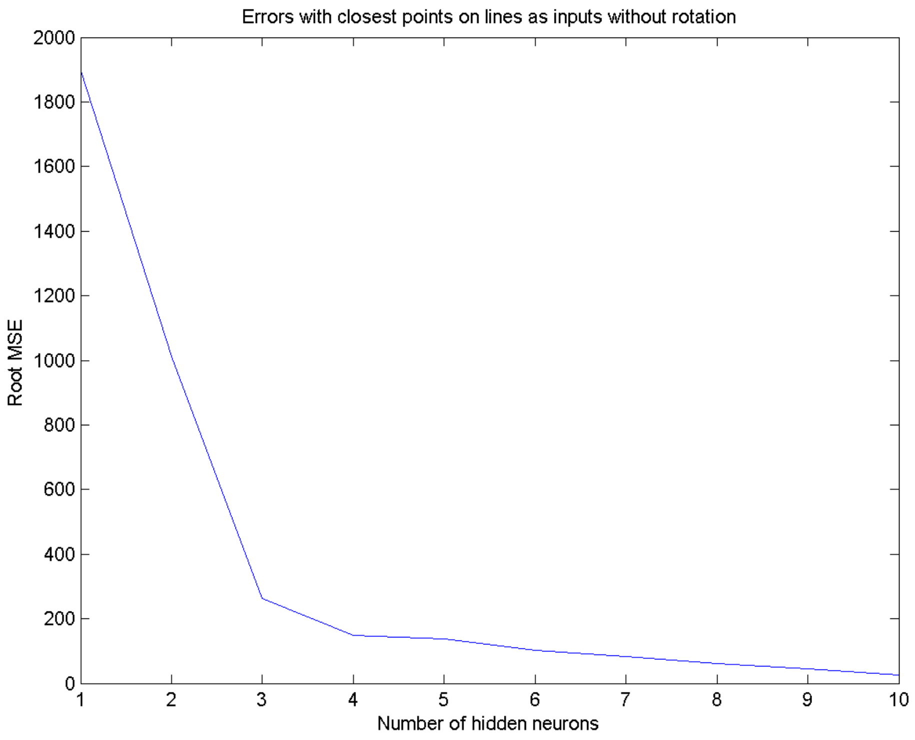
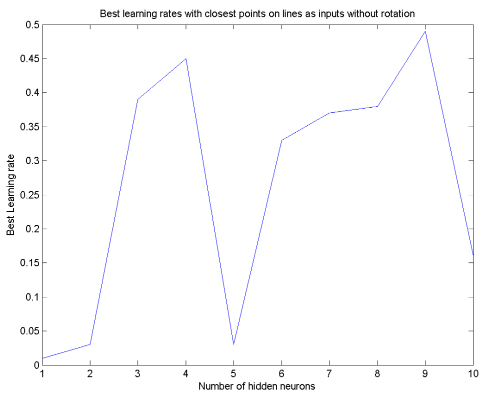
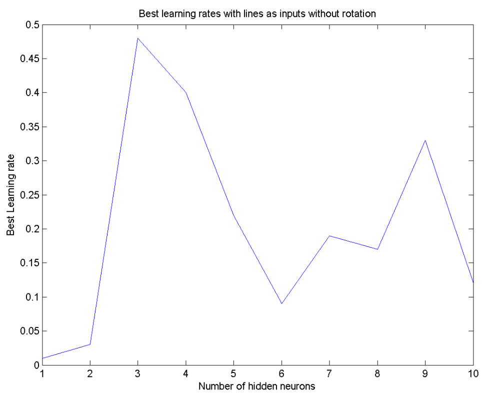


Figure 6.2: Root MSE depending on the number of hidden neurons, with the input **P** (only the closest points of the lines in input without transformations)



(a) Input **P**



(b) Input **L**

Figure 6.3: Best learning rates depending on the number of hidden neurons using a non-transformed input

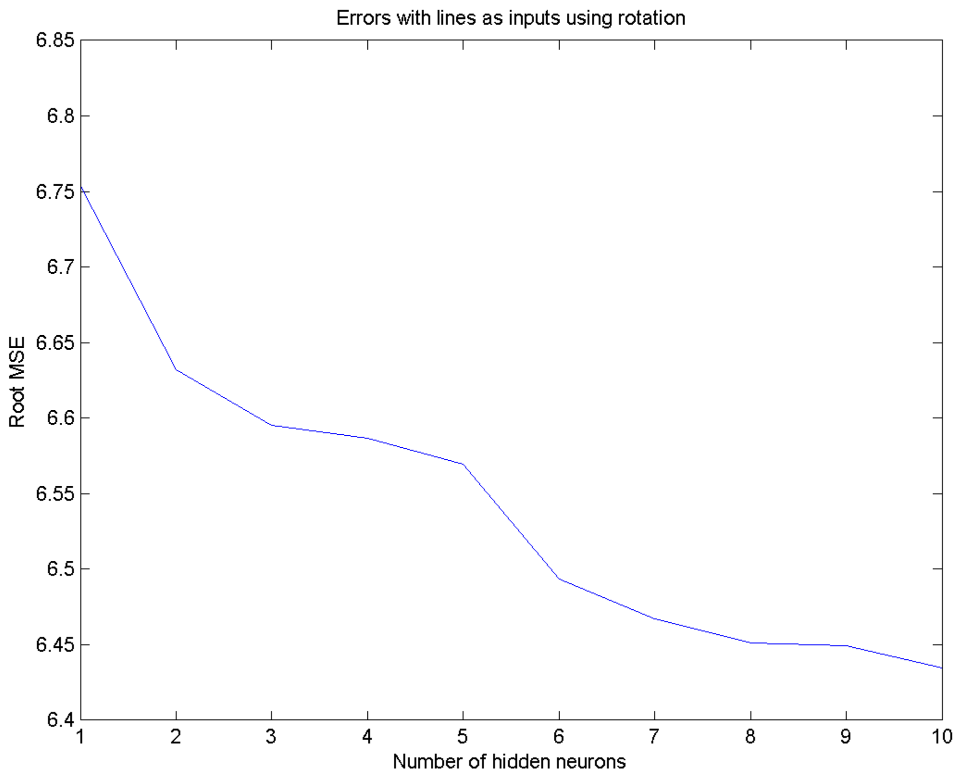


Figure 6.4: Root MSE depending on the number of hidden neurons, with the input $\mathbf{L_t}$ (with rotation).

6.2 Inputs with transformations

The error rate decreased when the number of hidden neurons increased with the input type $\mathbf{L_t}$ as seen on figures 6.4. This implies that evaluation with even more hidden neurons might have been useful with type $\mathbf{L_t}$.
With input type $\mathbf{P_t}$, the lowest error rate was reached with 9 hidden neurons as seen on figure 6.5. The root MSE with these hidden units was close to the one obtained with input type $\mathbf{L_t}$ using 10 hidden units. Both of these values were approximately 6.45 which is enought to be acceptable. It could be deducted that adding more than 9 hidden units would not help to minimize the error with this input type. However, this deduction can not be made with strong certainty using only the evidence obtained from hidden unit amounts within the range $[1, 10]$.

Here again, the networks trained with the best learning rate between 0.01 and 0.5 were chosen to show the error rates (figure6.6). The best learning rates were 0.15 with Pt input and 0.2 with Lt input.

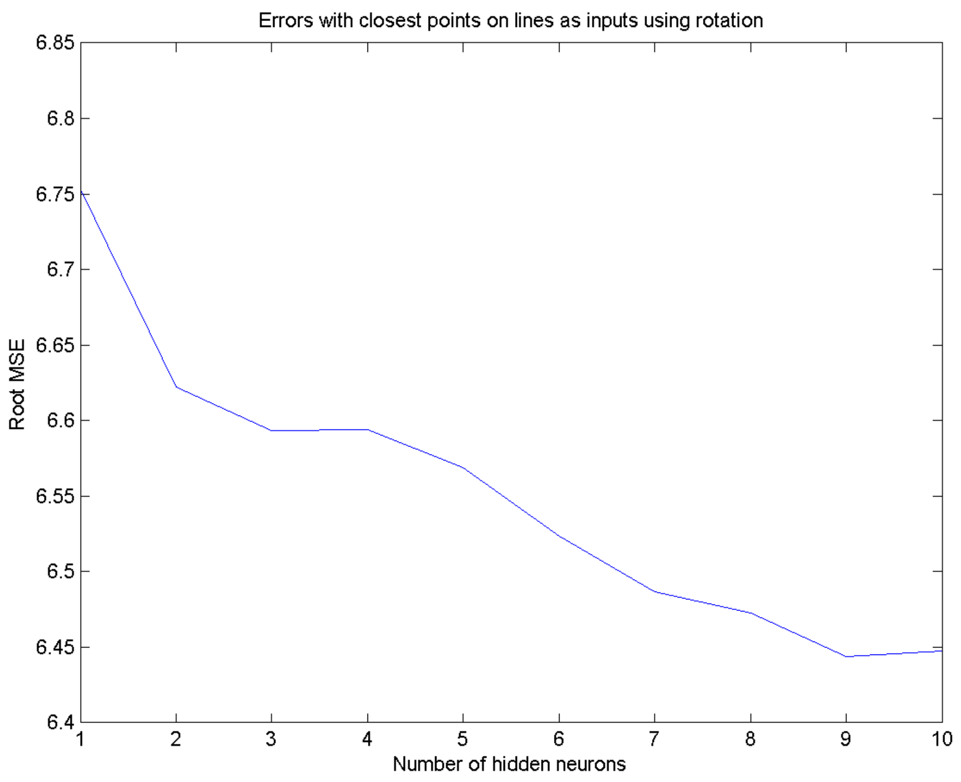
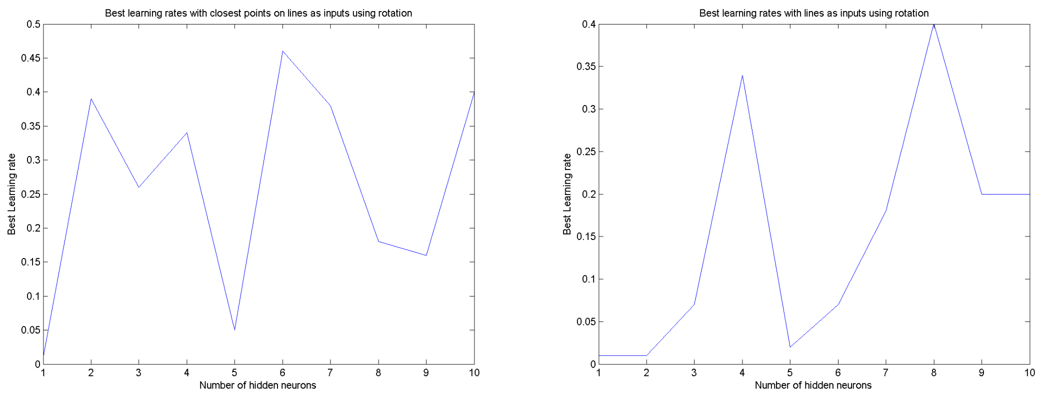


Figure 6.5: Root MSE depending on the number of hidden neurons, with the input $\mathbf{P_t}$ (with rotation).



(a) Input $\mathbf{P_t}$

(b) Input $\mathbf{L_t}$

Figure 6.6: Best learning rates depending on the number of hidden neurons using a rotated input

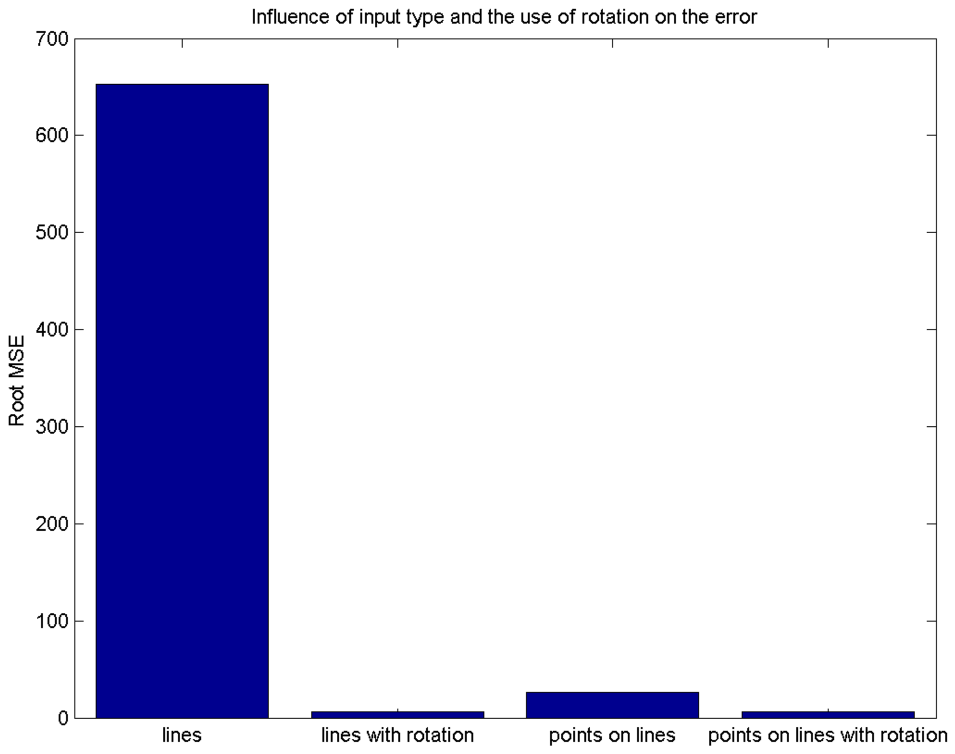


Figure 6.7: Influence of the input type on the root MSE. The best number of hidden units were used to measure the error rate for each input type. *lines* = **L**, *lines with rotation* = $\mathbf{L_t}$, *points on lines* = **P**, *points on lines with rotation* = $\mathbf{P_t}$.

6.3 All input vector types

If adding more hidden units would not produce better results with type $\mathbf{P_t}$, it would mean that adding more than 5 and 10 hidden units would not decrease the error with input vetor types **L** and $\mathbf{P_t}$ respectively. Similary types **P** and $\mathbf{L_t}$ would both benefit from the use of more than 10 hidden units. It is however possible that type $\mathbf{P_t}$ would also produce better results with more than 10 hidden units. The figure 6.7 clearly shows the advantage of transforming the input since the root MSE with these types of input vectros, that is $\mathbf{L_t}$ and $\mathbf{P_t}$, appears much lower than others root MSE.
The representation of lines had large effect on the errors with input vectors that were not transformed, as seen on figure 6.7. From figure 6.8 it can be seen that this was not the case when using the transformed inputs.
The duration of the training process in regards to the 4 different inputs is described in figure 6.9. The duration time was relative to the length of the input vector. The transformation of the inputs decreased the length of the inputs as did the use of only the closest points of the lines. With input types **P** and $\mathbf{P_t}$ the time for the process was similar which might be due

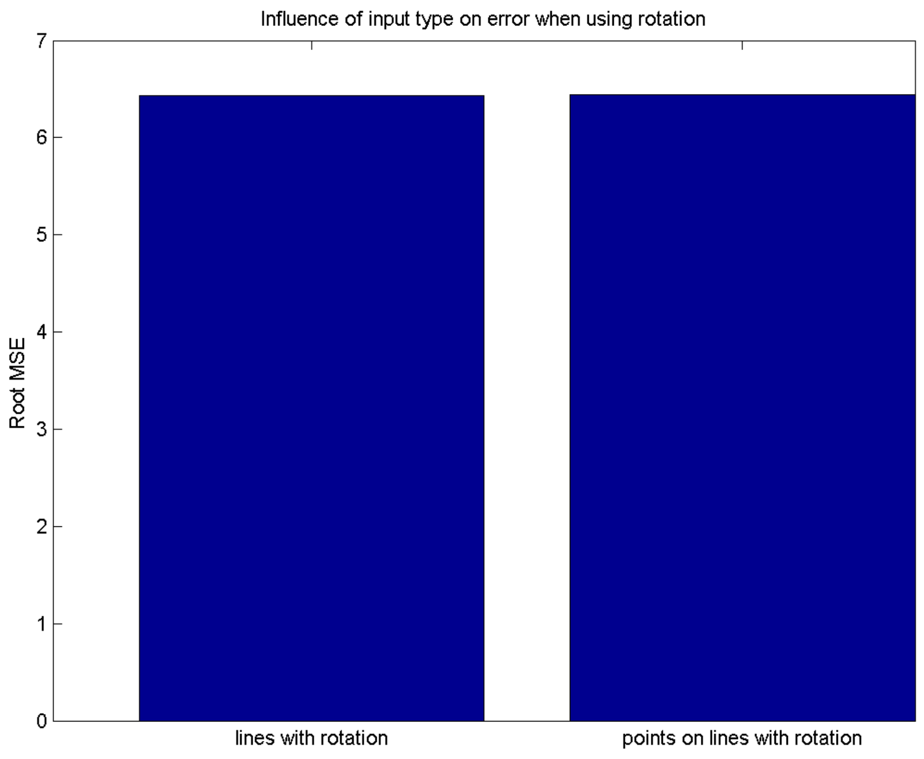


Figure 6.8: Influence of the input type on the root MSE. The best number of hidden units were used to measure the error rate for the transformed input types. *lines with rotation* = $\mathbf{L_t}$, *points on lines with rotation* = $\mathbf{P_t}$.

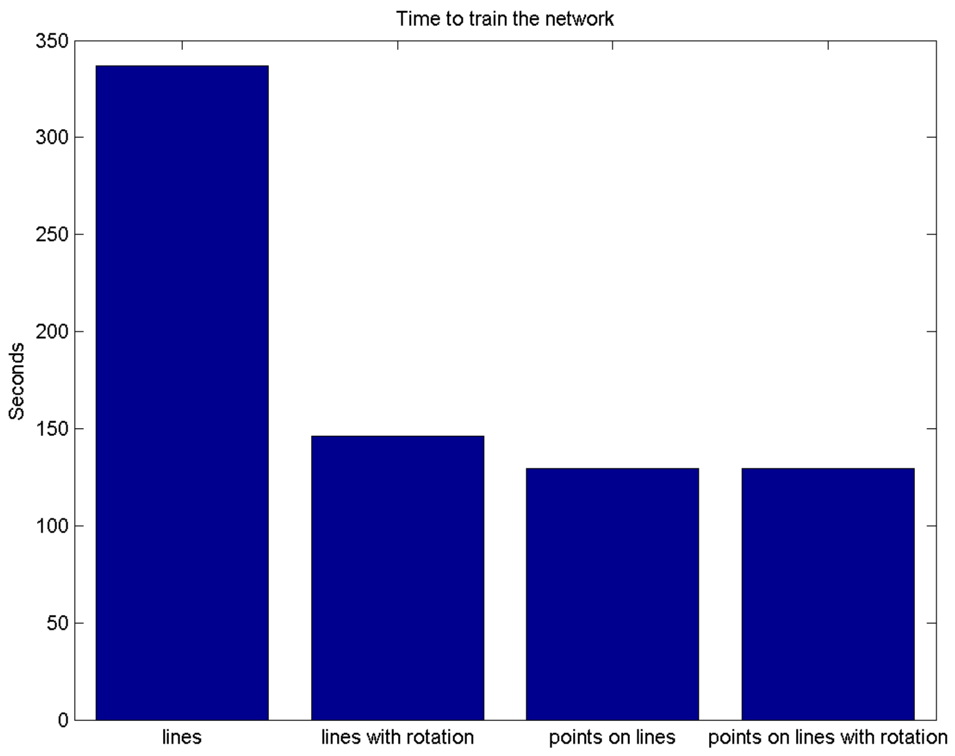


Figure 6.9: Influence of the input type on the training time. lines: input **L**, lines with rotation: input $\mathbf{L_t}$, points on lines: input **P**, points on lines with rotation: input $\mathbf{P_t}$.

to the overhead from calculating the transformations for each point in the input and output vectors. More time related benefits might be obtained when using larger networks.

7 Conclusions

Since networks with $\mathbf{P_t}$ and $\mathbf{L_t}$ produced equally accurate results, the best input vector type among those tested was with $\mathbf{P_t}$ because it required only 9 hidden units when $\mathbf{L_t}$ required 10. However, it could be argued that the two setups performed equally well since there was no difference in time during the training process. Also the overhead of calculating the closest points might make the use of $\mathbf{L_t}$ more beneficial. To determine this issue with certainty, the actual times to simulate the behavior of all the persons would be needed.

Also it would be useful to evaluate the performance of networks with more than 10 hidden units since with some of the input vectors the error rate seemed to be decreasing with more hidden units. This was the case especially with the input type $\mathbf{P_t}$.

If the hypothesis discussed in section 2.2 about the fences (lines) and other obstacles acting as pushing forces and corners of the buildings as pulling forces was true, it might be useful to experiment with RBF networks where the centers were placed in these points. Also the source of the panic would propably act as pushing force at least at small time steps. For the centers representing the pushing points one could use a non-localised function and for pulling points a localised function as the radial basis function.