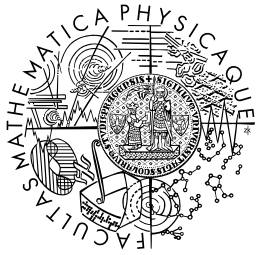

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Juraj Citorík

Predicting targets in Multiple Object Tracking task

Department of Software and Computer Science Education

Supervisor of the master thesis:	Mgr. Filip Děchtěrenko
Study programme:	Computer Science
Study branch:	Theoretical Computer Science

Prague 2016

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Predicting targets in Multiple Object Tracking task

Author: Juraj Citorík

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Filip Děchtěrenko, Department of Software and Computer Science Education

Abstract: The aim of this thesis is to predict targets in a Multiple Object Tracking (MOT) task, in which subjects track multiple moving objects. We processed and analyzed data containing object and gaze position information from 1148 MOT trials completed by 20 subjects. We extracted multiple features from the raw data and designed a machine learning approach for the prediction of targets using neural networks and hidden Markov models. We assessed the performance of the models and features. The results of our experiments show that it is possible to train a machine learning model to predict targets with very high accuracy.

Keywords: Multiple Object Tracking,machine learning,eye movements

None of this would be possible without the support of my family. Thank you. I would also like to thank my thesis supervisor Mgr. Filip Děchtěrenko for incessant encouragement, support and insightful advice. Last but not least, I would like to extend my gratitude to Samuel Bartoš and Lukáš Polák for helping me push through all the setbacks I have encountered in the process of researching and writing this thesis.

Contents

Introduction

In our everyday lives, we have to deal with a vast amount of information. Attention is a mechanism that helps us prioritize the most useful information in a given situation. The invention of eye tracking devices which enable us to track the position of a person's gaze has allowed us to study the connection between eye movements and visual attention. One task which is very suitable for the examination of visual attention mechanisms is Multiple Object Tracking (MOT) introduced by Pylyshyn and Storm [1988]. In a MOT trial, participants are tasked with the tracking of a subset of moving objects presented on a display. The tracked objects are called targets and the remaining objects are called distractors.

Pylyshyn and Storm [1988] presented evidence of a parallel tracking mechanism, indicating that people can divide their attention among multiple tracked objects. Multiple models for eye movement prediction in MOT have been suggested (Lukavsky [2013]). On the other hand, the inverse task of inferring the targets from the movements of the objects and the gaze has not yet been tackled.

We analyzed data from 1148 MOT trials completed by 20 participants. The data contained information about the movements of the targets, distractors and the gaze in MOT trials. First we examined the relationship between the position of the gaze and the position of targets and distractors. Then we attempted to utilize machine learning models to identify targets in the trials. We evaluated and compared the performance of neural networks and hidden Markov models. The results of our experiments are encouraging and suggest that eye movements in MOT task convey significant amount of information about the targets.

1. Attention and eye tracking

Attention is our ability to selectively focus on and process the large amount of information we are confronted with, prioritizing some aspects of the visual scene over others (Carrasco [2011]). The relationship between attention and eye movements is one of partial mutual dependence. Attention is free to move independent of the eyes (Posner [1980]), but eye movements need visual attention to guide them to the goal (Hoffman [1998]).

1.1 Eye tracking

The emergence of eye trackers, devices which allow us to determine where a person directs their gaze, enabled us to analyze and model visual attention. There are three types of eye trackers (Rayner [1998]):

- Eye-attached eye trackers - a special contact lens is attached to the eye, allowing very precise measurement of eye movements.
- Optical eye trackers - eye movements are tracked using video cameras or other optical devices. The gaze coordinates are computed from the center of the pupil and corneal reflection. This type of eye tracking device is very common.
- Electric potential eye trackers - electrodes are placed around the eye and electric potential, which changes as the eye moves, is measured and translated into coordinates.

1.1.1 Types of eye movements

Saccades are rapid movements of the eyes. Between saccades, our eyes remain relatively still during what we call *fixations*. During a saccade, we do not obtain any new information due to the speed of the eye movement (Rayner [1998]). When a person follows a slowly moving object, they perform *smooth pursuit* eye movements.

1.1.2 Visual angle

The size of the image projected on the retina is determined by the size of the object and the distance of the object from the eye. This causes ambiguity if we use the real size of an object, because the image of two objects of a different size can have the same size on the retina. For instance, the retina image size of two different objects is the same, if one is double the size of the other and its distance from the eye is also double the distance of the other object. To avoid this problem, a measure called *visual angle* capturing the ratio between object's size and distance has been defined in the study of visual perception. It can be computed as follows:

$$\alpha = 2 \arctan \frac{s}{2d}$$

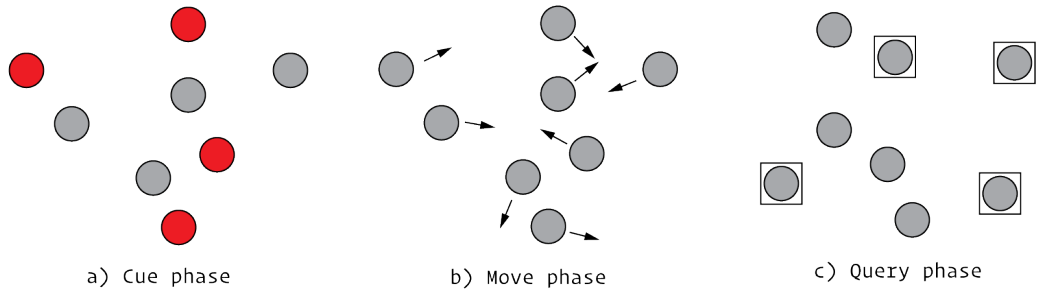

Figure 1.1: Multiple object tracking task

where s is the real size of the object, d the distance from the eye and α is its angular size.

1.2 Models of visual attention

For a static scene, several computational models of attention have been proposed. Saliency maps are one such model, which encodes stimulus conspicuity, or saliency at every location in the visual scene (Itti and Koch [2001]). Najemnik and Geisler [2005] have designed an ideal bayesian observer that gains the most information for the purpose of target finding in a target search task and compared it with the performance of humans.

Multiple Object Tracking (MOT), is an experimental technique introduced by Pylyshyn and Storm [1988], designed to study how our visual attention system tracks multiple moving objects. In a MOT task (Figure 1.1), the subject is presented with a display containing n points, out of which k are marked briefly. These are called *targets*. The remaining points are *distractors*. The marking is removed and the points start moving for 6–10 seconds. Afterwards the subject is asked to select the previously marked targets. Typically, $n = 8$ points with $k = 4$ targets are presented in a MOT trial.

The advantage of MOT is that it occurs in everyday life, for instance when we drive through an intersection, noting the locations of other nearby moving cars, or when playing team sports. An area where high MOT performance is crucial is flight traffic control.

1.3 Multiple Object Tracking

Since the introduction of MOT, multiple tracking mechanism models have been suggested, the main question being whether the tracking mechanism is sequential or parallel. The most popular models are (Cavanagh and Alvarez [2005]):

- Grouping model - all the targets are grouped into one higher order object with each target a vertex in a virtual polygon. This whole shape is then tracked as one object.

- Switching model - includes only one focus of attention that cycles through the targets, remembering their locations and returning to each before it gets

lost.

- Multifocal attention model - assumes that each target attracts a dedicated focus of attention and these follow the targets through the trial. Pylyshyn and Storm [1988] provides evidence for such a parallel tracking mechanism.

1.3.1 Properties of the MOT task

Experiments have suggested the following properties of MOT:

- the mean tracking capacity of trials lasting 5 seconds was 4 (Oksama and Hyönä [2004])
- tracking becomes impossible for displays spanning less than about 1/16th of a degree, even though the dots are clearly seen (Intriligator and Cavanagh [2001])
- accurate tracking of even one target moving on a circular path breaks down for speed greater than 1 to 2 revolutions per second (Verstraten et al. [2000])
- the basic unit of tracking is the dot. Subjects were unable to track targets once they were connected with a distractor, forming bar shaped objects Scholl et al. [2001]
- eye movement patterns are consistent for repeated trials with different number of distractors Dechterenko and Lukavsky [2014]

1.3.2 Models of eye movements in MOT

An intriguing question is, whether it is possible to predict where a person will look during a MOT trial. Revealing a formula for the prediction of the gaze position in MOT could provide us with valuable insights into the inner workings of the underlying tracking mechanism.

Zelinsky and Neider [2008] used trials with 9 animated shark models with 1–4 sharks marked as targets for tracking to analyze eye fixations during a MOT trial. They discovered that the tracking strategy depended on the number of targets. For one target, the subjects fixated the target. For two and three targets, they fixated the centroid of the polygon formed by the targets. When tracking four targets, the subjects switched between fixating on the targets and the centroid.

Experiments run by Fehd and Seiffert [2008] also support the finding that people fixate their gaze on the centroid when tracking more than two objects.

Landry et al. [2001] found out in their airplane tracking experiment that people tend to more often fixate their gaze on planes that are about to collide with another plane.

Lukavsky [2013] evaluated the following strategies and examined how well they predict the actual eye movement data from MOT trials:

- Constant strategy - a person looks at the center of the display.
- All-points centroid strategy - the distance to all points is minimized
- Target-centroid strategy - the distance to targets is minimized

- Object-centroid strategy - centroid of a planar object formed by the targets is followed by the gaze
- Anticrowding strategy - between the distance of target from the gaze point and distance from every distractor

Centroid strategy and its variants explained between 48.8%—54.3% of eye movements, whereas other people's eye movements explained 68.6%. This means that more sophisticated models are necessary to fully explain the eye movements in MOT.

Dechterenko and Lukavsky [2016] trained neural networks to predict the position of the gaze in MOT with results better than those of other current strategies. These findings serve as an incentive to apply machine learning techniques to other problems related to MOT as well.

Zelinsky et al. [2013] presented an approach to infer whether a person's task was to search for a bear or for a butterfly in a static display, based on the person's fixations during the search. He used support vector machines and compared their accuracy with the performance of humans.

Naturally the question arises, whether we can also use machine learning models to predict targets in the dynamic Multiple Object Tracking task. The aim of this thesis is to design a solution to this problem using machine learning models. We will predict targets in MOT trials with 4 targets and 4 distractors.

2. Machine Learning

In this chapter we describe the techniques and models that we used for the analysis of the data and the prediction of targets in Multiple Object Tracking.

Machine learning is a subfield of computer science that studies algorithms which enable computers to learn to solve problems without being explicitly programmed. These algorithms learn from data. There are two main types of machine learning:

- supervised learning - the algorithm learns from examples presented together with the desired answer
- unsupervised learning - the algorithm attempts to find hidden structure in the data, for example by applying cluster analysis

The part of data used for the training of a model is called a *training set*. To verify how well a model performs on inputs not seen during training, we use a small part of the dataset as a *testing set*.

2.1 Cluster analysis

Cluster analysis aims to divide a set of objects into groups in such a way that objects in the same group, called a cluster, are more similar to each other than to objects in other clusters.

2.1.1 Silhouette plot

While it is straightforward to evaluate clustering of two or three-dimensional data by simply plotting it, plotting a clustering of high-dimensional data in a format that is easy to interpret by humans is difficult. We would like to get a clustering of the data such that all objects within a group are close to each other and the groups themselves are clearly separated and far from each other. Rousseeuw [1987] introduced silhouette plot as a means of evaluating the quality of clustering for high-dimensional data. A silhouette plot of a clustering of some data can be obtained as follows: The input is:

- a clustering, i.e. a grouping of the analyzed objects (data rows)
- the distances between the objects, e.g. euclidean distance

For each object i in the data set and the group A to which it has been assigned we define

$$a(i) = \frac{\sum_{x \in A, x \neq i} d(i, x)}{|A| - 1}$$

i.e. the average distance of i to other elements in A. For some other cluster C, different from A, let us define

$$d(i, C) = \frac{\sum_{c \in C} d(i, c)}{|C|}$$

which is the average distance of i to all the elements in C. Moreover, we define

$$b(i) = \min_{C \neq A} d(i, C)$$

We call the cluster B for which $b(i) = d(i, B)$ the *neighbor* of object i. We finally define

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1 & \text{if } a(i) < b(i) \end{cases}$$

For $s(i)$ it holds that $-1 \leq s(i) \leq 1$ for all objects i. In case there is only one object in the cluster of i, we set $s(i) = 0$.

The interpretation of $s(i)$ is very straightforward, it measures how well object i matches the analyzed clustering:

- if $s(i)$ is close to 1, it means that the average within-cluster distance is much lower than the average distance to the objects of the closest neighbor cluster. This indicates that the cluster to which i has been assigned is very appropriate.
- if $s(i)$ is close to 0, it is not clear whether i should have been assigned to its current cluster or to its closest neighbor.
- if $s(i)$ is close to -1, the elements of the nearest neighbor cluster are much closer than the elements in the cluster of i, indicating that the object should probably have been assigned to the neighbor cluster.

We can now use the computed $s(i)$ values to create a *silhouette plot*. The *silhouette* of cluster A is a plot with the $s(i)$ values on the horizontal axis, ordered in decreasing order, for all $i \in A$. The horizontal axis corresponds to the cluster size. We print the silhouettes of all the clusters below each other. A wide silhouette with positive values indicates a good clustering. Figure 2.1 displays an example of a silhouette plot. The wide silhouette of the third cluster indicates that it is very compact and well separated from the other two clusters.

2.2 Neural networks

Neural nets are machine learning models, inspired by networks of neurons in the human brain, which learn in a supervised fashion.

2.2.1 Perceptron

Rosenblatt [1958] introduced a simple binary classifier called *perceptron*. It consists of n inputs and one binary output. A perceptron is characterized by a set of *weights* $w = [w_1 \dots w_n]$ and a *threshold*. For an input vector $x = [x_1 \dots x_n]$ the output $f(x)$ of the perceptron is defined as:

$$f(x) = \begin{cases} 1, & \text{if } w \cdot x > threshold \\ 0 & \text{else} \end{cases}$$

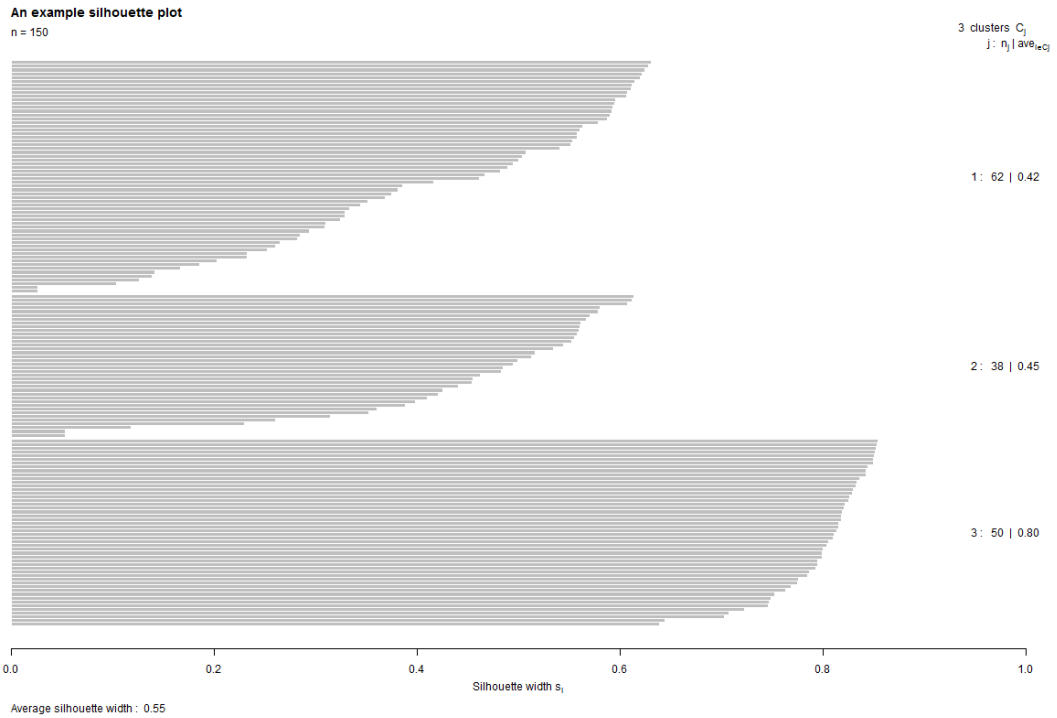

Figure 2.1: A silhouette plot with a well-separated, compact cluster

Perceptron is a simple model with limited capabilities. Even though Minsky and Papert [1988] showed that the perceptron cannot learn the XOR operation, researchers later found out that a network of interconnected perceptrons has much broader capabilities.

2.2.2 Artificial neural networks

A multilayer perceptron network is a machine learning model consisting of perceptrons divided into layers. We call the perceptrons *neurons*. Figure 2.2 shows the layout of such a network. The first layer is the *input layer* and the last layer the *output layer*. The layers between them are called the *hidden layers*. Each neuron in a layer is connected with all the neurons in the next layer, the output of the previous layer serving as the input of the next layer. The neurons within a layer are not connected with each other. The connections are weighted and the weights are adjusted during the process of learning to accommodate the presented training samples. We introduce a *bias* $b = -threshold$ to get an easier notation for the output of a neuron:

$$f(x) = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0 & \text{if } w \cdot x + b \leq 0 \end{cases}$$

This function is a step function and is plotted in Figure 2.3. Its disadvantage is that small changes in the weights can lead to a large change in the output. This is not desirable during training, as it makes the fine-tuning of the weights difficult. We remedy this by introducing a *sigmoid neuron*, which utilizes the

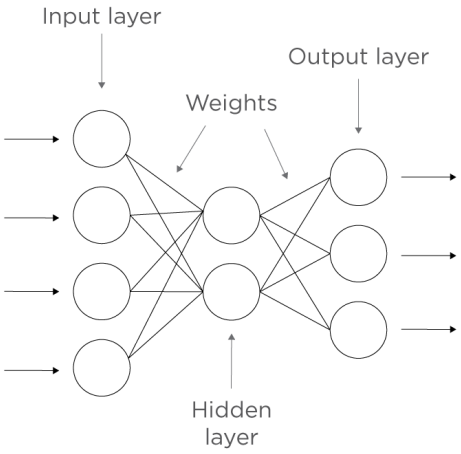

Figure 2.2: A layout of a multilayer perceptron network

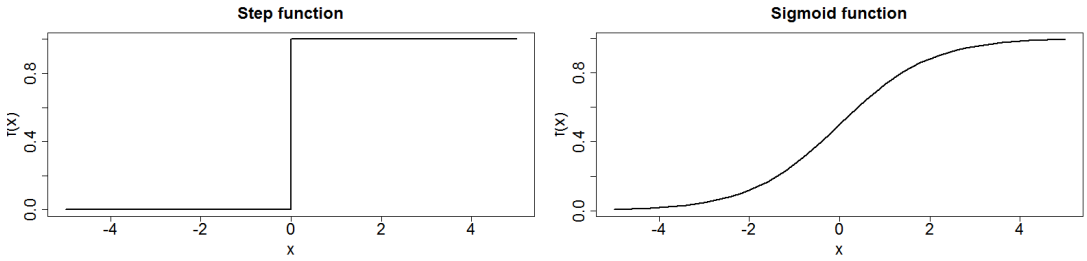

Figure 2.3: Step and sigmoid neuron output function

sigmoid function σ (Figure 2.3):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The output of the sigmoid neuron can then be defined as

$$f(x) = \sigma(w \cdot x + b)$$

2.2.3 Training

Let $x = [x_1 \dots x_n]$ denote an input vector and $y(x)$ the desired output for x. To assess the performance of our neural network, we define a *quadratic cost function*, also called *mean squared error*, which we try to minimize:

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - f(x)\|^2$$

where w and b denote the set of all the network weights and the set of biases, respectively, n is the number of training samples and $f(x)$ the actual output of the net for input x.

2.2.4 Gradient descent

In order to minimize the cost we need to know how it is influenced by the changing of the weights and the biases, which are the parameters of the neural net model. For clarity, we label all these parameters as w_i in this section. Formally, for a vector of parameters $w = [w_1 \dots w_n]$ we want to know what changes $\Delta w = [\Delta w_1 \dots \Delta w_n]$ to apply so that the cost C decreases, i.e. $\Delta C < 0$. Defining the gradient of C as $\nabla C = (\frac{\partial C}{\partial w_1} \dots \frac{\partial C}{\partial w_n})^T$ we can write:

$$\Delta C \approx \nabla C \cdot \Delta w$$

To make ΔC negative, i.e. to decrease the cost we can choose

$$\Delta w = -\lambda \nabla C$$

for a small $\lambda > 0$, called the *learning rate*. The cost decreases ($\Delta C < 0$) because $\Delta C \approx -\lambda \nabla C \cdot \nabla C = -\lambda \|\nabla C\|^2$. In this way, by changing weights and biases w to w' by setting

$$w' = w - \lambda \nabla C$$

we can lower the cost.

Our definition of the cost function requires that we compute the cost for each sample in the training set. This is not feasible for large datasets. We will estimate the true cost by randomly choosing a subset of samples from the training dataset and computing the cost for this subset. We call this subset a *mini-batch*. For a mini-batch $\{x_1 \dots x_m\}$ of size m, weight w_k and bias b_l we have the following rules for updating weights and biases during learning:

$$w'_k = w_k - \frac{\lambda}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial w_k}$$

$$b'_l = b_l - \frac{\lambda}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial b_l}$$

The remaining question is how to compute the gradient of the cost function efficiently.

2.2.5 Backpropagation

Rumelhart et al. [1986] introduced the backpropagation algorithm, which utilizes the error made by the neural network during training to update weights and biases in a way which reduces the cost. We describe the algorithm in this section.

First, we extend our notation so that we can clearly refer to specific weights in the network. In a network with L layers, layer 1 is the input layer, layers $2 \dots L-1$ are the hidden layers and layer L is the output layer. We use w^l_{jk} to refer to the weight between the k^{th} neuron in the $(l-1)^{th}$ layer and the j^{th} neuron in the l^{th} layer. Similarly, b^l_j and a^l_j denote the bias and the activation of the j^{th} neuron in the l^{th} layer, respectively. Using this notation, we can write the activation a^l_j as

$$a^l_j = \sigma \left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j \right)$$

where the sum is over all the neurons in the $(l-1)^{th}$ layer. If we use matrix notation, we can refer to a^l as the vector of activations of all the neurons in the l^th layer:

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

We call $z^l = w^l a^{l-1} + b^l$ the *weighted input* of the neurons in layer l.

Using the weighted input z_j^l of j^{th} neuron in layer l we define the *error* δ_j^l of neuron j in layer l as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

Using this quantity, we can obtain the error δ^L for all the output neurons in output layer L:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

written in matrix form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

where $\nabla_a C$ is a vector of partial derivatives $\partial C/\partial a_j^L$ and $\odot$ denotes elementwise multiplication. Since for quadratic cost $C = \frac{1}{2}\sum_j (y_j - a_j)^2$ is $\partial C/\partial a_j^L = (a_j - y_j)$, we have $\nabla_a C = (a^L - y)$ and as a result we can write

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Using the error terms of the output layer, we can gradually compute the error terms for the preceding layers:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

In this way the error obtained in later layers *propagates back* through the network. Once we have computed the error term for all the layers, we can finally compute the rate of change for the cost function with respect to weights and biases

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

For proofs of the above equations, see Nielsen [2015].

Using the backpropagation algorithm described above, we can now describe the training procedure for a neural net using a mini-batch of size m to estimate the cost function. We iterate through the training set, choosing mini-batches until there are no samples left in the training set. Once all samples have been trained on in this way, a *training epoch* is completed. Training usually entails many epochs, i.e. each sample is presented multiple times during the training of the net.

1. Pick a mini-batch $\{x_1, \ldots, x_m\}$ from the training set

2. Present each input x_i to the neural network:

- For each layer l compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$
- Compute the output layer error $\delta^L = \nabla_a C \odot \sigma'(z^L)$
- Propagate the error back through the network for $l \in \{L-1, \ldots, 2\}$ by computing $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
- Obtain the gradient of the cost function: $\frac{\partial C_{x_i}}{\partial b_j^l} = \delta_j^l$ and $\frac{\partial C_{x_i}}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

3. Update the weights and biases using the cost estimated from the batch:

$$w_k' = w_k - \frac{\lambda}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial w_k}$$

$$b_l' = b_l - \frac{\lambda}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial b_l}$$

4. If there are training samples left, go to Step 1. If the cost is sufficiently low or the number of training epoch has been reached, terminate, else continue with a new epoch.

2.2.6 Regularization and cross-entropy cost function

To improve the speed at which the network learns, we can use *cross-entropy cost function* instead of the quadratic cost function (Nielsen [2015]). The cross-entropy cost function is defined as follows:

$$C = -\frac{1}{n} \sum_x [y \log a + (1-y) \log(1-a)]$$

Another way of improving the learning of a network is *regularization*. This technique prevents the weights of the network from becoming too large. This is desirable, because the larger a weight is, the less likely it is that it is ever going to change significantly in the process of learning, since the learning algorithm only makes small changes to the weights. The way regularization is implemented is by adding a new term to the cost function:

$$C = -\frac{1}{n} \sum_x [y \log a + (1-y) \log(1-a)] + \frac{\alpha}{2n} \sum_w w^2$$

where w are the weights, a is the actual output of the network and y is the desired output. α determines how large the penalization for large weights is. This new cost function only necessitates small changes in our weight update procedure, detailed description can be found in Nielsen [2015].

2.3 Hidden Markov models

Hidden Markov models have been used successfully in many areas, especially speech recognition and signal processing. In our experiments, we will use them for sequence classification.

2.3.1 Markov chains

A *Markov chain* is a sequence of random variables $X_1 \dots X_n$, which satisfies the *Markov property*:

$$P(X_n = x \mid X_{n-1} = x_{n-1}, \dots X_1 = x_1) = P(X_n = x \mid X_{n-1} = x_{n-1})$$

The set of possible values of X_i has to be a countable set and is called the *state space* of the chain. A Markov chain essentially describes a system that changes its state at regular time intervals. It is characterized by a *transition matrix* A, where $a_{ij} = P(X_n = j \mid X_{n-1} = i)$ and by initial state probabilities $P(X_1 = s)$ for each state s. A Markov chain is *time-homogeneous* if the transition probabilities in are independent of n.

2.3.2 Hidden Markov models

In a Markov model each state corresponds to an observable event and is thus fully observable. In a Hidden Markov model (HMM) the observation also depends on the current state, but this state is unknown to the observer. For example, a person hidden behind a curtain tosses either a fair or a biased coin and reports the results of the toss. All we can observe are the reported values, but we do not know whether these were produced by the fair or by the biased coin. We can model this scenario using a HMM with 2 hidden states, one for each coin and 2 output symbols, one for head and one for tail.

A Hidden Markov model is characterized by the following (Rabiner [1989]):

1. N, the number of hidden states. We denote the set of all states as $S = \{S_1, S_2, \dots, S_N\}$ and state at time t as q_t.

2. M, the number of output symbols. We denote the set of all the output symbols as $V = \{v_1, v_2 \dots, v_M\}$.

3. $A = \{a_{ij}\}$, the state transition probability distribution, where $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)$ for $1 \leq i, j \leq N$

4. $B = \{b_j(k)\}$, the observation symbol probability in state j, where $b_j(k) = P(v_k \mid q_t = S_j)$ for $1 \leq j \leq N, 1 \leq k \leq M$

5. $\pi = \{\pi_i\}$, the initial state distribution, where $\pi_i = P(q_1 = S_j)$ for $1 \leq j \leq N$

Given the values of N, M, A, B and π the HMM can generate a sequence of observations $O = O_1, \dots O_T$ as follows:

1. Choose an initial state q_1 based on π and set $t = 1$.

2. Choose output symbol O_t according to the observation symbol probability distribution for state q_t specified in B.

3. Change the state to q_{t+1} according to q_t and the probabilities in A.

4. Increase $t = t + 1$ and return to step 2 if $t < T$, otherwise return O.

There are multiple questions we can ask about a HMM:

1. Given the observation sequence $O = O_1, \ldots O_T$ and a HMM $\lambda = (A, B, \pi)$, what is the probability that the sequence O has been produced by λ? This is called the *evaluation problem*.

2. Given the observation sequence $O = O_1, \ldots O_T$ and a HMM λ, how do we find the corresponding sequence of states that best explains the observed sequence O?

3. How do we choose the model parameters A, B, π for λ with respect to a sequence of observations O so that we maximize the probability $P(O \mid \lambda)$? This is called the *training problem*.

The problems that are of interest in our thesis are the evaluation problem and the training problem.

2.3.3 The evaluation problem

To compute $P(O \mid \lambda)$ for observation sequence O and HMM λ we can use the *Forward Procedure* (Rabiner [1989]). We define the *forward variable* $\alpha_t(i)$ as

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i \mid \lambda)$$

i.e. the probability of partial observation sequence up to time t and being in state S_i at time t, given the model. We can proceed inductively as follows:

1. Initialization:
$$\alpha_1(i) = \pi_i b_i(O_1), \text{ for } 1 \leq i \leq N$$

2. Induction:
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \text{ for } 1 \leq t \leq T-1, \ 1 \leq j \leq N$$

3. We finally compute $P(O \mid \lambda)$:
$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

2.3.4 The training problem

To estimate the parameters of λ we utilize the Forward Procedure from the previous section and define three more quantities $\beta_t(i)$, $\gamma_t(i)$ and $\xi_t(i, j)$ (Rabiner [1989]).

The *backward variable* $\beta_t(i)$ is the probability of the partial observation sequence from time $t + 1$ to the end, given state S_i at time t and the model.

$$\beta_t(i) = O(O_{t+1}, \ldots, O_T \mid q_t = S_i, \lambda)$$

We can again proceed inductively:

1. Initialization:
$$\beta_T(i) = 1 \text{ for } 1 \leq i \leq N$$

2. Induction:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \text{ for } t = T-1, \ldots, 1,\ 1 \leq i \leq N$$

$\gamma_t(i)$ is the probability of being in state S_i at time t, given the sequence of observations O and the model λ

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

We can express the value of $\gamma_t(i)$ in terms of the previously described forward and backward variables as follows:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

where the denominator $P(O \mid \lambda)$ ensures that $\gamma_t(i)$ is a probability measure, i.e. that
$$\sum_{i=1}^{N} \gamma_t(i) = 1$$

Finally, we define $\xi_t(i, j)$, the last quantity needed to describe the iterative HMM parameter estimation procedure:

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda)$$

in other words, the probability of being in state S_i at time t and subsequently in state S_j at time $t + 1$, given the HMM model and the observation sequence O. We can compute its value from the forward and backward variables:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}$$

where the division by $P(O \mid \lambda)$ again ensures that $\xi_t(i, j)$ is a probability measure. The numerator is just $P(q_t = S_i, q_{t+1} = S_j, O \mid \lambda)$.

Using $\gamma_t(i)$ and $\xi_t(i, j)$ we can describe the *Baum-Welch algorithm*(Rabiner [1989]) for estimation of HMM parameters with respect to observation sequence O.

The Baum-Welch algorithm works iteratively, starting with a randomly initialized model $\lambda = (A, B, \pi)$ and repeatedly improves it by reestimating the parameters in the following way:

1. Start with a random initialization of $\lambda = (A, B, \pi)$

2. Compute reestimated parameters $\overline{A}$, $\overline{B}$ and $\overline{\pi}$ as follows:

   - Estimate π as the expected number of times in S_i at $t = 1$
   $$\overline{\pi}_i = \gamma_1(i)$$

- Estimate A as expected number of transitions from S_i to S_j divided by the expected number of transitions from S_i (to any state).

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

- Estimate B as expected number of times in S_j and observing v_k divided by expected number of times in S_j.

$$\overline{b}_j(k) = \frac{\sum_{1 \le t \le T,\ O_t = v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

3. Set $\lambda = (\overline{A}, \overline{B}, \overline{\pi})$ and repeat Step 2 until desired convergence is reached.

Baum and Sell [1968] have proved that for each HMM λ and a reestimated HMM $\overline{\lambda}$ either $\lambda = \overline{\lambda}$ or $P(O \mid \overline{\lambda}) > P(O \mid \lambda)$, i.e. the observation sequence is more likely for the new model. We stop the algorithm once $P(O \mid \overline{\lambda}) - P(O \mid \lambda) < tolerance$ for a certain positive *tolerance* or after a specified number of iterations.

To use the Baum-Welch algorithm for HMMs with continuous observations we need only modify the reestimation procedure for $b_k(j)$, see Liporace [1982] .

3. Methods

In this chapter we outline the data and the machine learning models used in our experiments. First, we describe the preprocessing steps and then explore the properties of the obtained dataset. The second part of the chapter contains the description of Neural Networks and Hidden Markov models.

3.1 Data

The data we used to train our models comes from Lukavsky [2013]. It contained 1148 MOT trials with 4 targets and 4 distractors, completed by 20 subjects on 40 different dot trajectories. Not all subjects saw all 40 trajectories. The size of the display was 30 by 30 degrees. A trial lasted 10 seconds and in the first 2 seconds of each trial, the targets were highlighted and did not move. Afterwards, the dots started to move for 8 seconds. The subject was then asked to mark the targets. Only data from trials where all 4 were marked correctly was used in our experiments. The trial data consisted of two files:

- *eyedata_RpM2.csv* - eye movement data
- *trackdata_RpM2.csv* - dot movement data

3.1.1 Eye movement data

Table 3.1 lists the columns in *eyedata_RpM2.csv*, their data types and description. The data was obtained using the SR Research Eyelink II optical eye tracker with the sampling rate of 250Hz, which produced 250 data points per second. The eye movement data was available only for the 8 seconds of a trial during which the dots moved, the timestamps ranged between $time = 2$ and $time = 10$.

3.1.2 Dot trajectory data

Table 3.2 lists the columns in *trackdata_RpM2.csv*, their data types and description. The data contained 100 data points per second. The timestamps ranged between $time = 0$ and $time = 12$.

3.1.3 Merging eye movement and track data

Since the sampling rates for dot movement and eye movement were different, it was necessary to choose a common sampling rate before joining the data into one dataset. We decided on a sample rate of 10Hz. The resulting 10 frames per second provided enough information about both eye and dot movements. The differences between frames carried meaningful information, as opposed to the original sample rates, where the difference between consecutive frames was too small to be useful for machine learning models.

From *trackdata_RpM2.csv* we discarded rows with timestamp lower than 2 and greater than 10, as there was no eye movement data available for these

Name	Type	Description
id	integer	subject ID
trial	integer	trial ID
block	integer	trial block ID
time	float	timestamp in seconds
x	float	x coordinate of the point of gaze, a visual angle
y	float	y coordinate of the point of gaze, a visual angle
track	int	point trajectory id
set	int	trial type ID
OK.4	boolean	answer of the subject was correct

Table 3.1: Columns in eyedata_RpM2.csv

Name	Type	Description
track	int	point trajectory id
time	float	timestamp in seconds
x1...x4	float	x coordinate of the 4 targets, a visual angle
y1...y4	float	y coordinate of the 4 targets, a visual angle
x5...x8	float	x coordinate of the 4 distractors, a visual angle
y5...y8	float	y coordinate of the 4 distractors, a visual angle

Table 3.2: Columns in trackdata_RpM2.csv

Name	Type	Description
id	int	subject id
trial	int	trial id, unique for trials of one subject
track	int	dot trajectory id
x1...x4	float	x coordinate of the 4 targets, a visual angle
y1...y4	float	y coordinate of the 4 targets, a visual angle
x5...x8	float	x coordinate of the 4 distractors, a visual angle
y5...y8	float	y coordinate of the 4 distractors, a visual angle
x	float	x coordinate of the gaze, a visual angle
y	float	y coordinate of the gaze, a visual angle

Table 3.3: Columns in RpM2_trialData.csv

timestamps in *eyedata_RpM2.csv*. This left us with information about 8 seconds of a trial.

The resulting dataset thus contained 80 frames per trial, at 10 frames per second. Table 3.3 contains the description of the columns in the final merged dataset. Trial frames were grouped by trial and ordered by timestamp (i.e. the first 80 rows constituted one trial and so forth). We omitted the timestamp in the final dataset as it was sufficient to know the order of the frames in a trial.

3.1.4 Basic properties

The coordinates of both dots and the gaze ranged between -10 and 10. The ids of the subjects were not consecutive integers, there were 20 different values ranging from 1 to 116. There were 40 different trajectory ids, ranging from 301 to 340.

3.1.5 Cluster analysis

We used the k-means clustering algorithm (see MacQueen [1967]) to examine the structure of the data. Well-separated clusters would indicate that there are certain situations, i.e. positions of targets and distractors based on which we could identify targets in our models.

Since our main focus was the relationship between the position of the gaze and the positions of targets and distractors, we transformed the coordinates in each frame to what we call *relative coordinates* by making the point of gaze the origin of the coordinate system and recomputing the coordinates of all the points accordingly. Formally, we define the relative coordinates (x_{r_i}, y_{r_i}) for each point

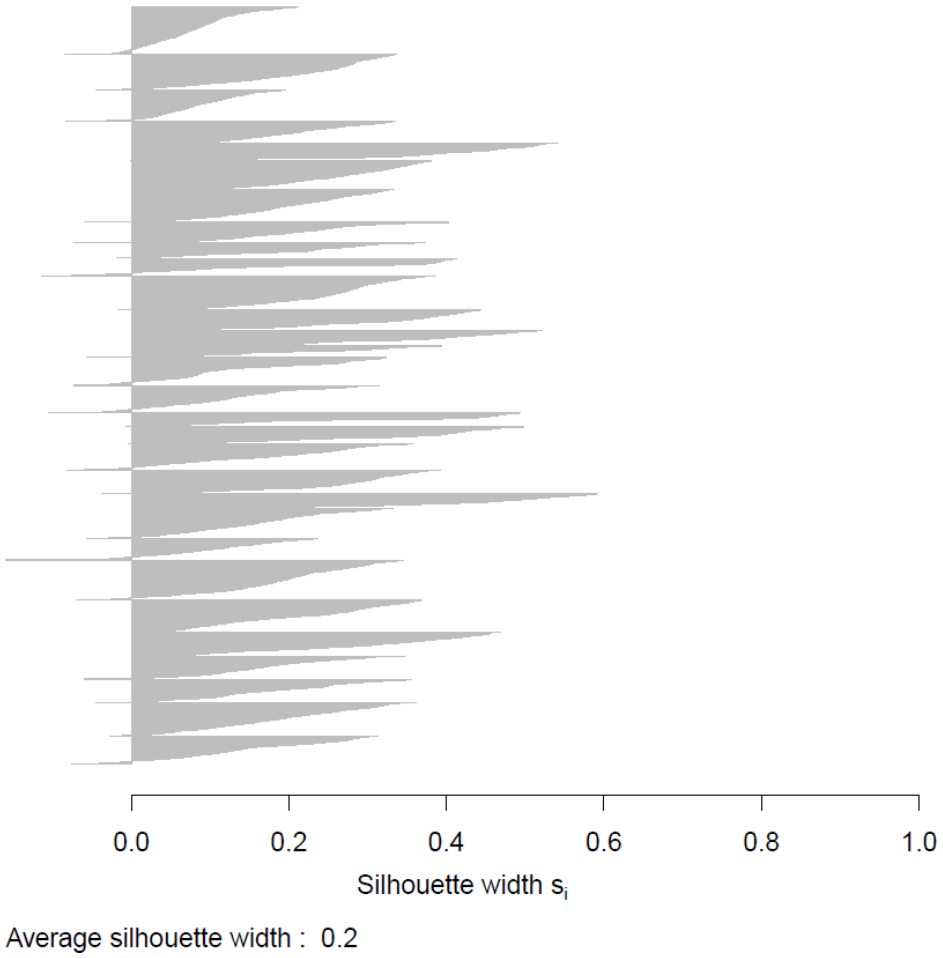

Figure 3.1: Silhouette plot of a clustering obtained for k=30 using the k-means algorithm

(x_i, y_i) as follows:

$$x_{r_i} = x_i - x$$

$$y_{r_i} = y_i - y$$

where (x, y) is the position of the gaze.

The results of the k-means algorithm for $2 \leq k \leq 50$ did not show any suitable grouping of data for any k. The average silhouette width ranged from 0.1 to 0.26, implying inadequate separation between groups. Figure 3.1 shows the resulting silhouette plot for $k = 30$. Other values of k provided similar results.

In summary, the results of the cluster analysis show that the relationship between the positions of targets and distractors and the gaze cannot be categorized into a number of similar situations. This means that the data did not contain any constellations of point positions that would be closely related to a specific position of the gaze. This might be caused by the fact that people have different target tracking strategies in MOT trials and directed their gaze differently. We examine differences between subjects in more detail later in this chapter.

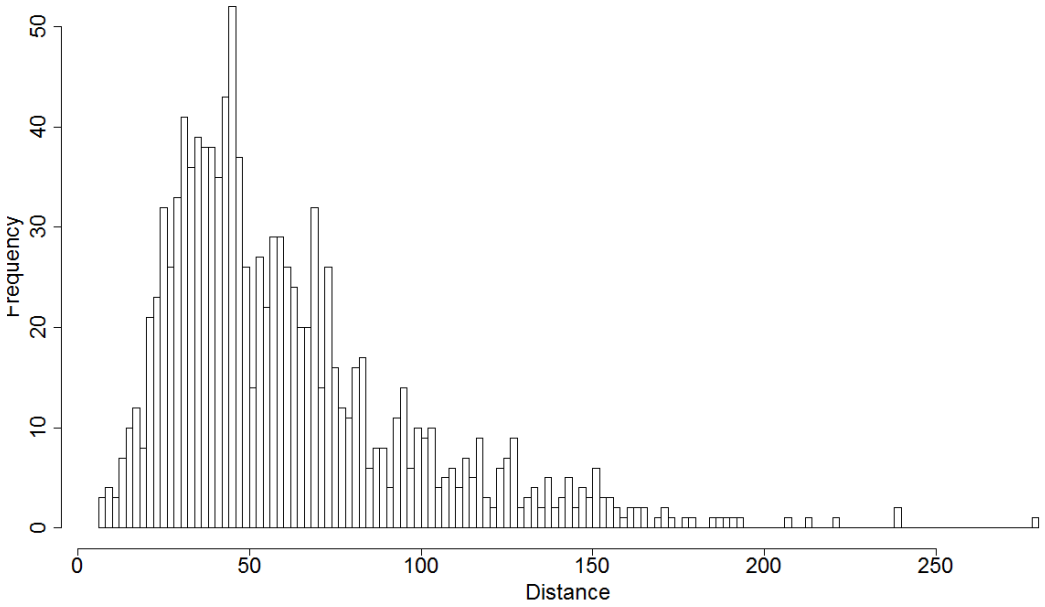

Figure 3.2: Distance covered by gaze in a trial

3.1.6 Distance covered by gaze

In this section, we examine how much the subjects moved their eyes during a trial. We computed the total distance covered by the gaze for each trial. Figure 3.2 shows the distribution of these distances, with mean value 61.2 and standard deviation of 36.8.

Figure 3.3 shows comparison of distance covered by gaze between subjects. We can see that subject 5 moved their eyes much more than most other subjects. This suggests a different tracking strategy.

Figure 3.4 depicts the amount of eye movement made by each subject on all trajectories. We can see that the distance covered by gaze was nearly constant for some subjects, but varied considerably for other subjects.

3.1.7 Average distance of the gaze from targets

To what extent does the gaze follow the targets in a MOT trial? We inspected how far the gaze was from the targets over time in a trial. For each frame, we computed the euclidean distance between the point of the gaze and each target and averaged the values. Figure 3.5 shows this average distance plotted over time, for each subject completing a trial on track 326. The shape of the plotted curve is quite similar, in spite of our previous finding that the amount of eye movement varied significantly among subjects. Plots for other tracks display similar consistency between subjects. What this plot also shows us is that the gaze did not follow the targets closely all the time. In this case it wandered away from the targets between the 4. and 6. second of the trial. Similar parts of trial with large average distance between the gaze and the targets were found in trials on other trajectories as well.

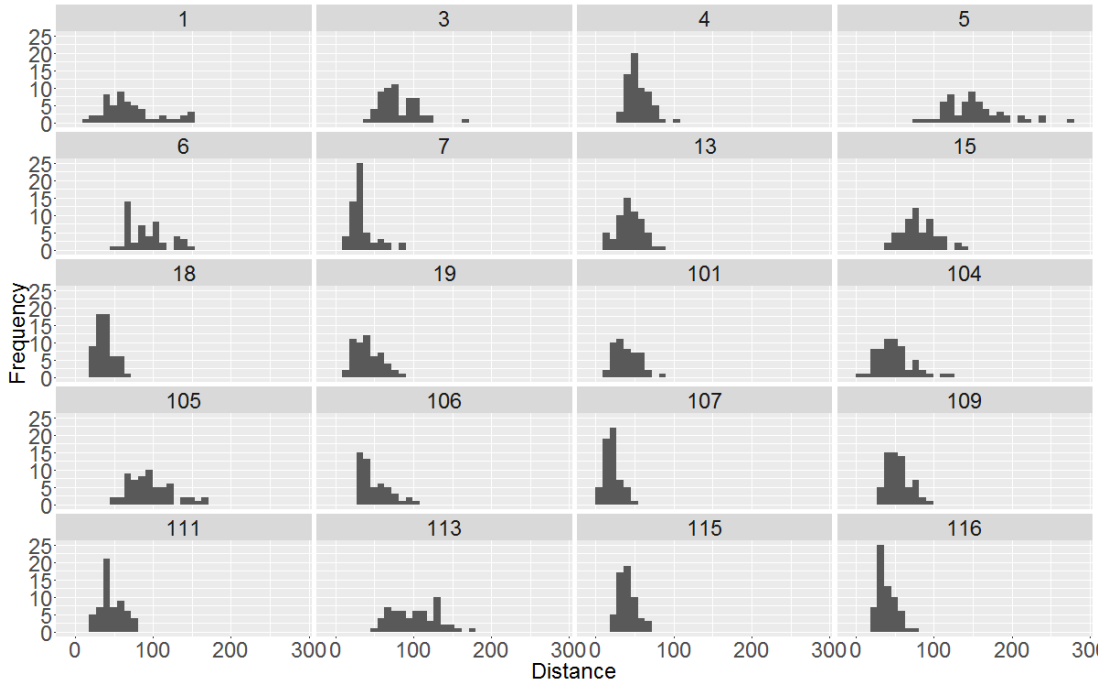

Figure 3.3: Distance covered by gaze in a trial by each subject

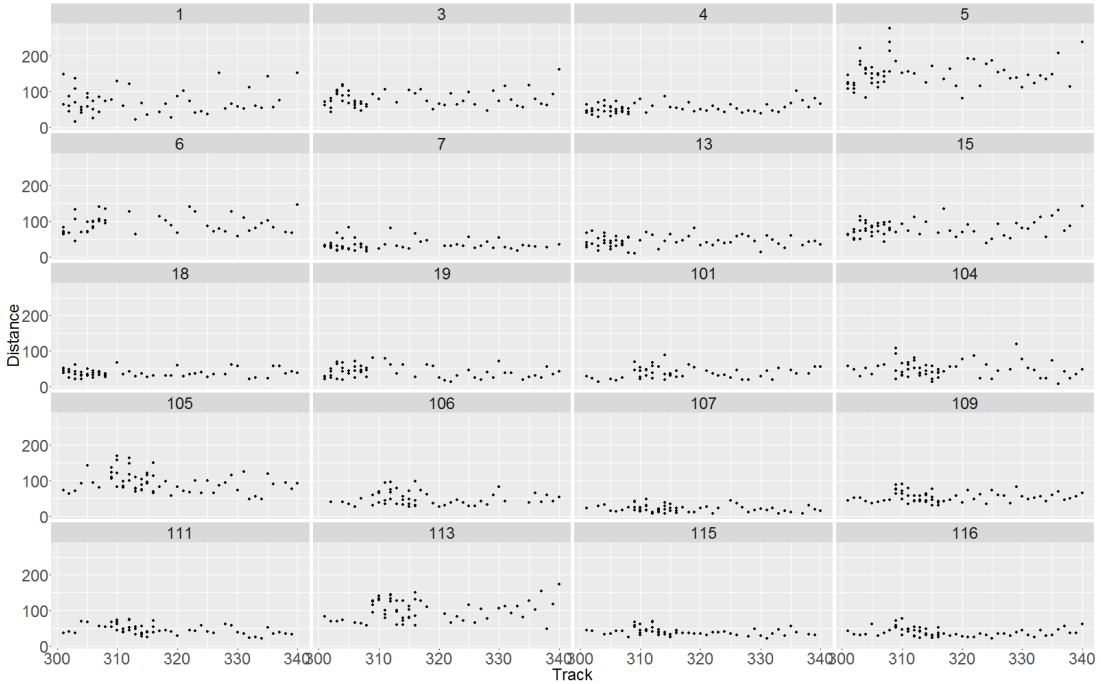

Figure 3.4: Distance covered by gaze for each track by each subject

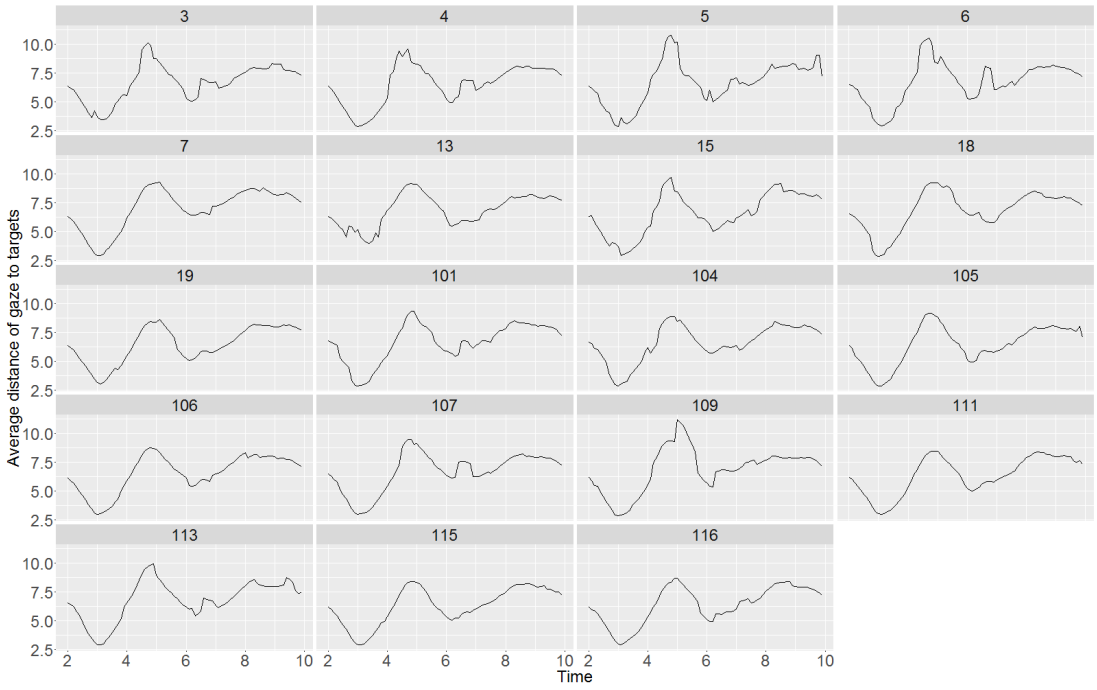

Figure 3.5: Average distance of the gaze to the targets for subjects on track 326

3.1.8 Consistency of eye movements in repeated trials

Across repeated trials on the same trajectory, subjects' eye movements were often very consistent, especially for subjects whose gaze did not wander much. Figure 3.6 shows average distance to gaze for repeated trials on track 303 of subjects 5 and 19. The distance curves in the plot are more similar for subject 19 than for 5. This is a result of their different tracking strategies. Nevertheless, we can see that the general shape of the distance curve remained the same across trials on one trajectory.

3.1.9 Distance of the gaze to targets and distractors

To get an idea of how reliable a predictor of targets the distance of a point to the point of gaze could be, we plotted the distances between all the points and the gaze over time in a trial. In Figure 3.7 we see that although it is quite often the case that a target (red line) is closer to the point of gaze than a distractor (black line), it does not hold for all targets all the time. The situation was even more complicated for subjects who moved their gaze a lot, as is the case with subject 5.

3.2 Models

3.2.1 Data preprocessing

Prior to applying the models, we scaled the raw data using the `scale` function included in the R programming language (R Development Core Team [2008]), which works as follows. For each column x in the dataset, the mean $\overline{x}$ and

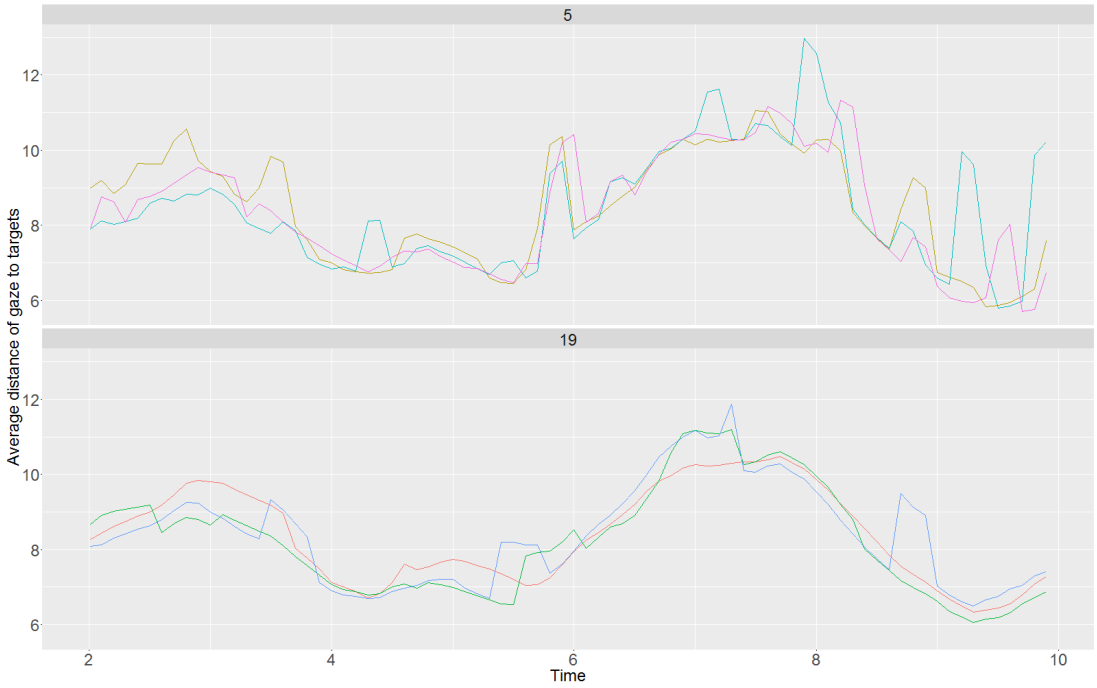

Figure 3.6: Average distance of the gaze to the targets for subjects 5 (top) and 19 (bottom) on track 303. Different color indicates different trial.

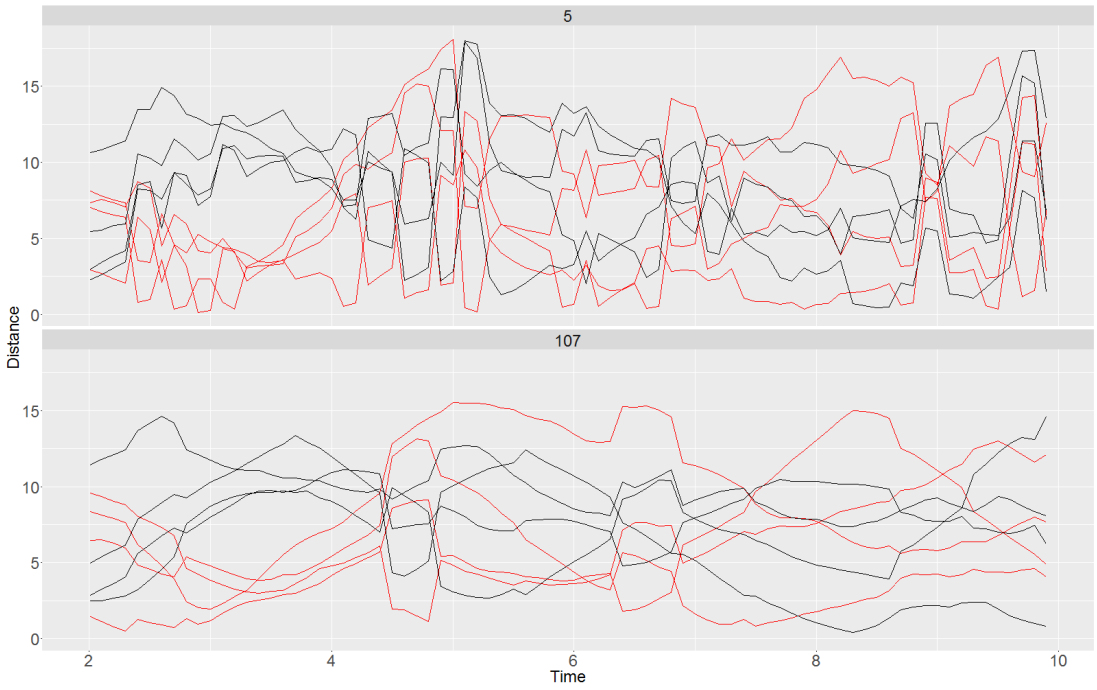

Figure 3.7: Distance of the gaze to targets and distractors for subjects 107 (bottom) and 5 (top) on track 326. Red lines represent distance to targets.

standard deviation σ is computed first. We obtain the scaled value x' as

$$x' = \frac{x - \overline{x}}{\sigma}$$

After scaling, all the columns had mean of zero and unit variance.

3.2.2 Evaluation

If we split the data randomly into training and testing data, there was a danger that good results, if obtained, could merely be a result of the model memorizing the specific trajectories. To ensure that the models generalize well to trajectories not seen during training, we always split the data into training and testing set in such a way that trajectories used for training were not used for testing.

The data used in model assessment contained 1148 MOT trials of 20 subjects. In the first part of the chapter we saw that the amount of eye movement of different people on the same dot trajectory differed significantly. Thus we evaluated the models in two ways:

- Trials of one subject - to examine how a model trained on one person's trials performs for other trials of the same person
- Trials of two subjects - to examine how well the model generalizes when trained on trials of one subject and tested on different subject's trials

In our experiments we used three-fold cross-validation (Kohavi et al. [1995]). The trial trajectories contained in one partition were not included in any other partition.

Apart from evaluating the success rates for the correct prediction of all 4 targets, we also assessed the success rates for the correct prediction of exactly 3 targets, with one 1 mispredicted target. Mispredicting one target can still be considered a meaningful result, as the probability of randomly guessing 3 out of 4 targets is $16/70 \approx 22.9\%$.

3.2.3 Hidden Markov models

Hidden Markov models are used for sequence classification. We used Python and the `hmmlearn` package to train a HMM with continuous observations. We set the training parameters so that the training terminates when the number of reestimation steps reaches 30 or when $\Delta P(O \mid \lambda) < 1$ between reestimation steps.

We trained two HMMs to recognize whether an individual point is a target or not, based on an observed sequence of distance to gaze features computed for that point. In this way, we obtained 8 observation sequences for each trial. One of the models, λ_t was trained to recognize targets and the second model, λ_d recognized distractors. We computed the probability of each point's feature sequence O_i for both models and computed the difference d_i for $1 \leq i \leq 8$

$$d_i = P(O \mid \lambda_t) - P(O \mid \lambda_d)$$

The four points with the largest d_i were predicted as targets.

In our evaluation we also employed the windowing technique, in which we divided each observation sequence into multiple overlapping *windows*. The overlap was constant and set to half the size of the window. When using windows, we computed d_i^w for each window w obtained from observation sequence O_i just like we did previously for the whole observation sequence. Next, we summed the resulting probabilities, again obtaining $d_i = \sum_{w \subset O_i} d_i^w$ for each point. Using these summed d_i values we predicted targets as in the previous case. Our experiments examined the influence of window size and the number of hidden states on the performance of the model.

3.2.4 Neural networks

In our experiments we used an adjusted a Python implementation of neural networks from Nielsen [2015]. We trained a neural network with one hidden layer to classify points in individual frames. The output of the network was an 8-tuple with values between 0 and 1 and each value representing one dot. We chose the four dots with the highest output as the targets. A trained network was then used to predict targets in a MOT trial by classifying all 80 frames of the trial. We summed the outputs for each frame and chose the four points with the highest sum of frame outputs as the targets.

We experimented with various hidden layer sizes. The other parameters of the neural net were:

- Learning rate: $\lambda = 0.5$
- Regularization parameter: $\alpha = \frac{|training\ set|}{10000}$
- Cost function: Cross-entropy cost function
- Mini-batch size: 10

Evaluated features

We compared the following features in model evaluation:

- Raw coordinates - an 18-tuple with the x and y coordinate of each dot and the gaze
- Relative coordinates - a 16-tuple with the x and y coordinate of each dot, with the gaze serving as the origin of the coordinate system, as described earlier in this chapter
- Distance features - an 8-tuple with the distance from the gaze for each point, as described earlier in this chapter

Permutation invariance

We also required that our models correctly marked targets even if the order of points in the input vector was permuted, since the order of the coordinates or the distances in the input vector does not convey any meaning. To achieve this permutation invariance, we permuted each sample in both the training and the testing set. There are $\binom{8}{4} = 70$ ways of choosing four targets out of eight points, hence we had 70 permuted input vectors for each vector in the original dataset.

3.2.5 A simple model

Based on the findings about the data from the previous chapter, we defined a simple algorithm for target prediction that served as a baseline for the comparison of various approaches. The algorithm selected the points that were the most closely followed by the gaze as the targets.
The algorithm works as follows:

1. For each frame f in a trial, compute the Euclidean distance $d_f(i)$ between the gaze and dot i for $1 \leq i \leq 8$

2. Summing the distances over all frames, we get $Dist_i = \sum_{k=1}^{N} d_k(i)$ for $1 \leq i \leq 8$, i.e. the sum of distances from the gaze in a trial for each dot. N denotes the number of frames in a trial.

3. The predicted targets are the 4 dots with the lowest $Dist_i$

4. Results

This chapter summarizes the results of the experiments described in the previous chapter. The description of the source code used for these experiments can be found in Attachment 1.

4.1 Simple model

We begin with the assessment of the simple model, described at the end of the previous chapter. We tested the model on all 1148 trials. The model predicted all targets correctly in 2.4% of trials, which is only a slightly higher success rate than chance ($\frac{1}{70} \approx 1.4\%$). However, the model predicted 3 targets out of four with a success rate 43.7% which is nearly twice as high as chance ($\binom{4}{3} \cdot \binom{4}{3} \approx 22.6\%$).

4.2 Hidden Markov Models

4.2.1 One-subject dataset performance

Table 4.1 provides an overview of the accuracy of a HMM model trained and tested on trials of the same subject. We carried out the evaluation for each of the 20 subjects. The values in the table are averaged over all participants. It is difficult to choose the best performing model parameters based on the results. While it can be easily seen that window size 80 performs the best, the choice of the best number of hidden states is unclear. On average, however, HMM outperformed the simple model.

4.2.2 Two-subject dataset performance

In Table 4.2 we can see that hidden Markov models do not generalize well to trials of subjects different from the training subject. The success rates are below those of the simple model. The significant decrease in accuracy compared to the one-subject dataset performance indicates that our HMM models are prone to overfitting for all combinations of model parameters.

4.3 Neural Networks

Apart from assessing neural network models on one-subject and two-subject datasets, we also evaluated the performance of different features. We compared raw coordinates with relative coordinates and distance to gaze features.

4.3.1 One-subject dataset performance

In Table 4.3 we can see that the best frame classification with raw coordinates features is achieved with the hidden layer size of 20. In spite of the accuracy not being extremely high, combining answers of the net to obtain target prediction for a whole trial yields nearly perfect accuracy, as can be seen in Table 4.4.

Window	States	0 errors	1 error
20	2	0.04	0.45
20	3	0.04	0.47
20	4	0.05	0.51
20	5	0.05	0.47
20	8	0.05	0.49
40	2	0.03	0.43
40	3	0.03	0.45
40	4	0.05	0.51
40	5	0.03	0.48
40	8	0.03	0.52
80	2	0.05	0.44
80	3	0.06	0.38
80	4	0.04	0.59
80	5	0.07	0.53
80	8	0.06	0.55

Table 4.1: Success rates for a HMM trained and evaluated on a one-subject dataset, averaged over all subjects.

Window	States	0 Errors	1 Error
20	2	0.01	0.27
20	3	0.01	0.33
20	4	0.01	0.32
20	5	0.01	0.32
20	8	0.01	0.33
40	2	0.01	0.28
40	3	0.01	0.33
40	4	0.01	0.33
40	5	0.02	0.34
40	8	0.01	0.32
80	2	0.01	0.27
80	3	0.02	0.32
80	4	0.02	0.32
80	5	0.02	0.33
80	8	0.02	0.32

Table 4.2: Success rates for a HMM trained and evaluated on a two-subject dataset, averaged over all subjects.

N.Hidden	0 Errors	1 Error
10	0.26	0.62
20	0.76	0.22
30	0.74	0.24
40	0.71	0.26
50	0.69	0.28

Table 4.3: Individual frame classification success rates for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using raw coordinates features.

N.Hidden	0 Errors	1 Error
10	0.34	0.65
20	1.00	0.00
30	1.00	0.00
40	1.00	0.00
50	0.94	0.00

Table 4.4: Target prediction success rates for a whole trial for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using raw coordinates features.

N.Hidden	0 Errors	1 Error
10	0.28	0.57
20	0.77	0.21
30	0.78	0.20
40	0.76	0.22
50	0.75	0.23

Table 4.5: Individual frame classification success rates for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using relative coordinates features.

Relative coordinate features describe the constellation of the dots and the gaze and neglect the information about the specific location of the gaze and the dots on the screen. Due to this, we expected better generalization. The trained neural network did indeed perform better in individual frame classification, as shown in table 4.5. The number of hidden neurons needed to achieve the best performance increased to 30 compared to raw coordinate features. The better performance in frame classification did not, however, translate into an improvement of the accuracy of prediction in a whole trial, as we can see in table 4.6. The number of trials in which only 3 targets were marked correctly is greater than in the case of raw coordinates. Surprisingly, the accuracy of prediction when using only 10 hidden neurons is significantly higher compared to raw coordinates.

Distance features used with neural networks performed worse than when used in hidden Markov models and were outperformed even by the simple model. In tables 4.7 and 4.8 we can see that the hidden layer size had little influence on the accuracy of target prediction. This result coupled with the fact that distance features were outperformed even by the simple model indicate that the distance

N.Hidden	0 Errors	1 Error
10	0.58	0.40
20	0.93	0.06
30	0.93	0.06
40	0.93	0.06
50	0.91	0.08

Table 4.6: Target prediction success rates for a whole trial for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using relative coordinates features.

N.Hidden	0 Errors	1 Error
10	0.02	0.29
20	0.02	0.28
30	0.01	0.29
40	0.02	0.29
50	0.01	0.29

Table 4.7: Individual frame classification success rates for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using distance to gaze features.

N.Hidden	0 Errors	1 Error
10	0.04	0.36
20	0.03	0.35
30	0.03	0.35
40	0.03	0.37
50	0.02	0.39

Table 4.8: Target prediction success rates for a whole trial for a neural network trained and evaluated on a one-subject dataset, averaged over all subjects, using distance to gaze features.

to gaze does not contain enough information about a point being a target or not. Due to the poor performance of the distance features, we omitted them in the following section, which includes evaluation results on a two-subjects dataset.

4.3.2 Two-subject dataset performance

When evaluating neural networks on trials of subjects different from the subject used for training, the best accuracy achieved was the same for both raw coordinates and relative coordinates. Table 4.9 shows that the best results were achieved with 20 hidden neurons for raw coordinates and for relative coordinates the number was again 30, as shown in table 4.10.

Figure 4.1 contains a detailed overview of model performance on two-subject datasets using raw coordinates. Each plot describes how a neural network with 20 hidden neurons trained on the corresponding subject's trials performs when evaluated on the trials of all the other subjects. Figure 4.2 contains the same comparison for relative coordinates and a neural network with 30 hidden neurons.

N.Hidden	0 Errors	1 Error
10	0.07	0.70
20	0.97	0.03
30	0.94	0.06
40	0.90	0.10
50	0.85	0.15

Table 4.9: Target prediction success rates for a whole trial for a neural network trained and evaluated on a two-subject dataset, averaged over all subjects, using relative coordinates features.

N.Hidden	0 Errors	1 Error
10	0.25	0.56
20	0.95	0.05
30	0.97	0.03
40	0.94	0.06
50	0.94	0.05

Table 4.10: Target prediction success rates for a whole trial for a neural network trained and evaluated on a two-subject dataset, averaged over all subjects, using relative coordinates features.

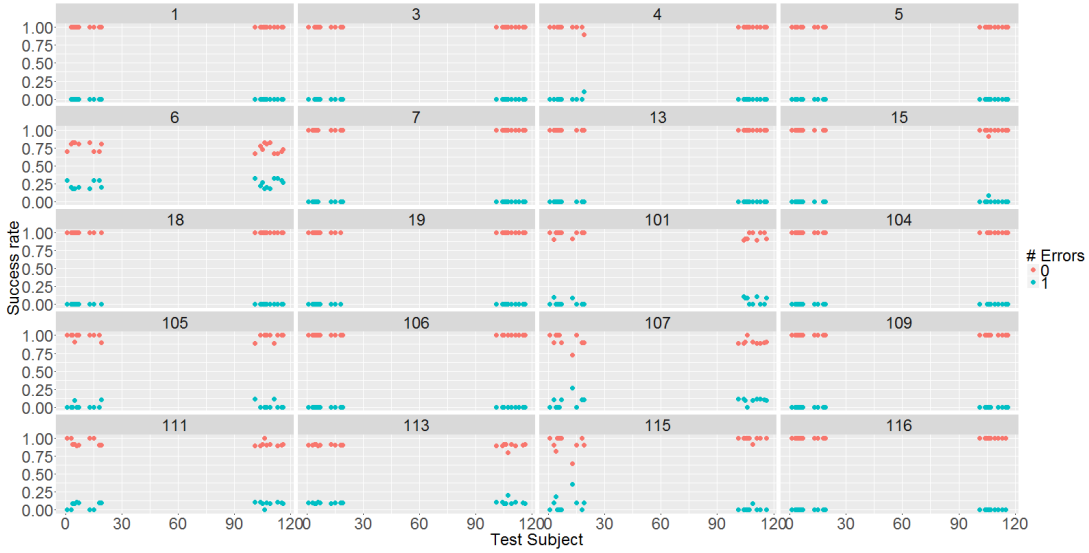

Figure 4.1: Neural net performance on two-subject datasets using raw coordinate features. Each plot describes how a model with 20 hidden neurons trained on trials of the corresponding subject performs on trials of all the other subjects.

Notable is the performance of models trained on the trials of subject 5. This subject moved their eyes much more in all trials than the other participants did. In spite of their apparently different tracking strategy, the neural network managed to learn enough to predict targets with perfect accuracy in the case of raw coordinates and nearly perfect accuracy when using relative coordinates.
On the other hand, the trials of subject 107, whose eye movements were small and consistent in all trials, did not contain enough information to achieve as good accuracy as with other subjects.
We can also see that the accuracy of neural networks trained on subjects 4 and 6 differs significantly when using different features. It is not clear why this is the case, because in our initial data analysis the subjects did not stand out in any way.

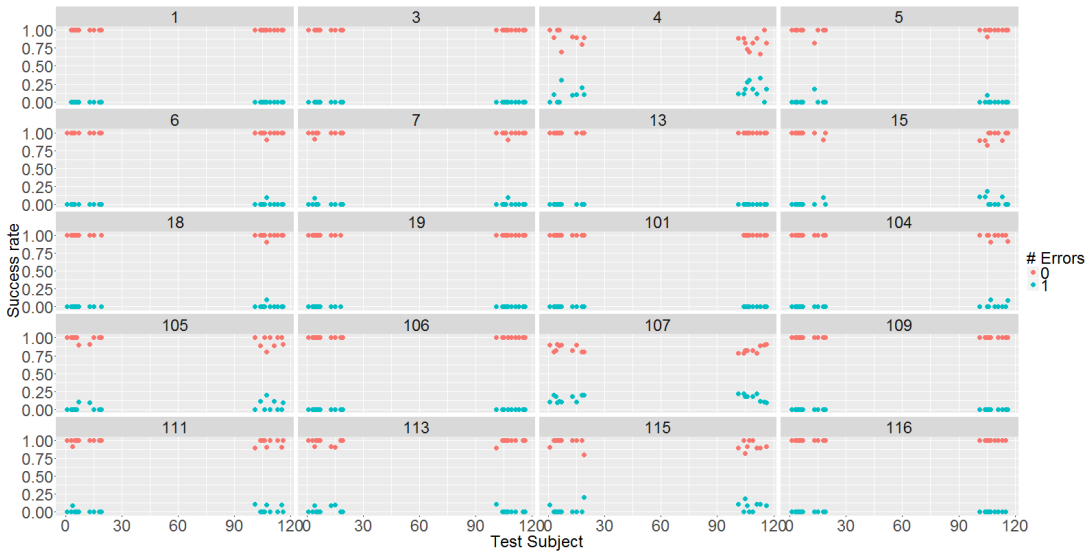

Figure 4.2: Neural net performance on two-subject datasets using relative coordinate features. Each plot describes how a model with 30 hidden neurons trained on trials of the corresponding subject performs on trials of all the other subjects.

5. Discussion

In our experiments we have evaluated the following three models for target prediction in Multiple Object Tracking trials.

5.1 Simple model

We found out that the simple model cannot predict all 4 targets in almost any trial, but predicts 3 targets correctly quite well given the straightforward nature of the model.

5.2 Hidden Markov models

Hidden Markov models also fail to make fully correct predictions, but when evaluated on a one-subject dataset, they outperform the simple model when the right parameters are used. However, HMMs did not generalize well to trials of subjects different from the training subject and performed worse than the simple model. This suggests overfitting.

Our models were limited in that they used one-dimensional observations, which in turn limited the choice of features we could use. Hidden Markov models can, however, be adjusted to work with multi-dimensional observations and using these modified HMMs with, for instance, raw coordinates could yield better results.

5.3 Neural networks

Finally, we found out that neural networks can predict all 4 targets in a MOT trial with remarkable accuracy, even when evaluated on a two-subject dataset. Last but not least, we evaluated three kinds of features that we used with neural networks, discovering that raw and relative coordinates perform equally well in most cases and that distance features are unsuitable for MOT target prediction.

The disadvantage of neural networks is that by including all the possible permutations of each input vector in the training set, we increased the size of the training set seventy times. This significantly increased the time needed to train a network. To reduce the training time, we tried sorting the points in the input vector by the x coordinate. The results of this approach were poor, however.

5.4 Other models

Apart from the aforementioned models, we also attempted to apply Naive Bayes model, the accuracy of which was only slightly above chance.

Conclusion

In this thesis, we have used neural networks and hidden Markov models to predict targets in Multiple Object Tracking task. First we processed data containing information about the positions of targets, distractors and eye gaze in MOT trials and performed exploratory data analysis to examine the relationship between the position of the gaze and that of targets and distractors.

Subsequently we trained neural networks and hidden Markov models to predict targets in the trials. We also extracted different features from the raw data and compared the performance of these features when used with neural networks. The results of our experiments are encouraging and suggest that it is possible to predict the targets from eye movement data with high accuracy. Our work has shown that the application of machine learning techniques to Multiple Object Tracking problems is a very promising area that deserves more attention.

Further augmentations of our work might extend the target prediction to real-world MOT tasks, such as air traffic control, where an interesting task would be to analyze and assess the performance of air traffic controllers. Extending the MOT target prediction to video is also an intriguing challenge.

Bibliography

Leonard E Baum and George Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.

Marisa Carrasco. Visual attention: The past 25 years. *Vision Research*, 51(13): 1484–1525, jul 2011. doi: 10.1016/j.visres.2011.04.012. URL `http://dx.doi.org/10.1016/j.visres.2011.04.012`.

P Cavanagh and G Alvarez. Tracking multiple targets with multifocal attention. *Trends in Cognitive Sciences*, 9(7):349–354, jul 2005. doi: 10.1016/j.tics.2005.05.009. URL `http://dx.doi.org/10.1016/j.tics.2005.05.009`.

Filip Dechterenko and Jiri Lukavsky. Models of eye movements in multiple object tracking with many objects. In *Visual Information Processing (EUVIP), 2014 5th European Workshop on*, pages 1–6. IEEE, 2014.

Filip Dechterenko and Jiri Lukavsky. Predicting eye movements in multiple object tracking using neural networks. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pages 271–274. ACM, 2016.

Hilda M Fehd and Adriane E Seiffert. Eye movements during multiple object tracking: Where do participants look? *Cognition*, 108(1):201–209, 2008.

James E Hoffman. Visual attention and eye movements. *Attention*, 31:119–153, 1998.

James Intriligator and Patrick Cavanagh. The spatial resolution of visual attention. *Cognitive psychology*, 43(3):171–216, 2001.

Laurent Itti and Christof Koch. Computational modelling of visual attention. *Nature reviews neuroscience*, 2(3):194–203, 2001.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

Steven J Landry, Thomas B Sheridan, and Yan M Yufik. A methodology for studying cognitive groupings in a target-tracking task. *IEEE Transactions on intelligent transportation systems*, 2(2):92–100, 2001.

L Liporace. Maximum likelihood estimation for multivariate observations of markov sources. *IEEE Transactions on Information Theory*, 28(5):729–734, 1982.

J. Lukavsky. Eye movements in repeated multiple object tracking. *Journal of Vision*, 13(7):1–16, jun 2013. doi: 10.1167/13.7.9. URL `http://dx.doi.org/10.1167/13.7.9`.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley,

Calif., 1967. University of California Press. URL http://projecteuclid.org/euclid.bsmsp/1200512992.

Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-63111-3.

Jiri Najemnik and Wilson S Geisler. Optimal eye movement strategies in visual search. *Nature*, 434(7031):387–391, 2005.

Michael A Nielsen. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/.(visited: 11.07. 2016)*, 2015.

Lauri Oksama and Jukka Hyönä. Is multiple object tracking carried out automatically by an early vision mechanism independent of higher-order cognition? an individual difference approach. *Visual cognition*, 11(5):631–671, 2004.

Michael I Posner. Orienting of attention. *Quarterly journal of experimental psychology*, 32(1):3–25, 1980.

Zenon W Pylyshyn and Ron W Storm. Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spatial vision*, 3(3):179–197, 1988.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org. ISBN 3-900051-07-0.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Keith Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3):372–422, 1998. doi: 10.1037/0033-2909.124.3.372. URL http://dx.doi.org/10.1037/0033-2909.124.3.372.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, nov 1987. doi: 10.1016/0377-0427(87)90125-7. URL http://dx.doi.org/10.1016/0377-0427(87)90125-7.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. doi: 10.1038/323533a0. URL http://dx.doi.org/10.1038/323533a0.

Brian J Scholl, Zenon W Pylyshyn, and Jacob Feldman. What is a visual object? evidence from target merging in multiple object tracking. *Cognition*, 80(1): 159–177, 2001.

Frans AJ Verstraten, Patrick Cavanagh, and Angela T Labianca. Limits of attentive tracking reveal temporal properties of attention. *Vision research*, 40 (26):3651–3664, 2000.

G. J. Zelinsky, Y. Peng, and D. Samaras. Eye can read your mind: Decoding gaze fixations to reveal categorical search targets. *Journal of Vision*, 13(14): 10–10, dec 2013. doi: 10.1167/13.14.10. URL `http://dx.doi.org/10.1167/13.14.10`.

Gregory J Zelinsky and Mark B Neider. An eye movement analysis of multiple object tracking in a realistic environment. *Visual Cognition*, 16(5):553–566, 2008.

List of Figures

List of Tables

List of Abbreviations

- MOT - Multiple Object Tracking
- HMM - Hidden Markov Model

Attachment 1 — Source code description and data

We briefly describe the source code and data included on the attached DVD. All the experiments were made with Python 3.5. The `hmmlearn` library is required to run HMM experiments.

Source code

The source code is divided into two parts, each part located in a separate folder.

- Simple model
- Neural Networks and Hidden Markov Models

The *Simple Model* folder contains `simpleModel.py`, which can be run in order to execute the evaluation of the model.
The *Neural Nets and HMM* folder contains Python modules for neural networks and hidden Markov models, feature extraction and experiment execution. The following five Python scripts execute HMM and neural network experiments:

- `e_hmm_one_subject.py` - executes one-subject dataset HMM evaluation
- `e_hmm_two_subjects.py` - executes two-subjects dataset HMM evaluation
- `e_nn_one_subject.py` - executes one-subject dataset Neural Network evaluation
- `e_nn_two_subjects_raw.py` - executes two-subjects dataset Neural Network evaluation with raw coordinates
- `e_nn_two_subjects_rel_dist.py` - executes two-subjects dataset Neural Network evaluation with relative coordinates and distance to gaze features

Data

The preprocessed data used in all experiments is located in the `Neural Nets and HMM/CSV` folder.

Trial videos

We included videos of MOT trials as well. The videos in the `All subjects` folder show the gaze of all the subjects at once on that specific trajectory. The remaining videos show individual trials.