Generating Trajectories for Offensive Players in Professional Football

Arnold Dijk
student number: 2675389
June 2022

A thesis presented for the degree of
MSc Business Analytics

Beyond Sports
Noorderkade 145
1823CJ ALKMAAR

supervisor: Daniel Karavolos

Vrije Universiteit
De Boelelaan 1081a
1081 HV Amsterdam

supervisor: Vincent François-Lavet
second reader: Rianne de Heide

1 Abstract

The opinions of analysts during the break of professional football matches are subjective and cannot be objectively confirmed due to a lack of analytical tools. So, the need arises for tools that can visualize what would have happened in alternative scenarios. Developing this kind of tools is a broad problem that can be divided into sub-problems. In this study, we focus on the sub-problem of generating trajectories of offensive professional football players. Here, we have investigated multiple models and feature sets. We found out that the Hierarchical Policy Network shows the highest performance. The architecture of the Hierarchical Policy Network is based on the paradigm that trajectories of players in team sports can be decomposed into a raw-micro and macro policy. Here, the macro policy guides the player toward the desired position on the field. Besides, the micro policy makes sure that the generated trajectory looks natural. The model takes features, describing all players and the ball as input. These features are given to the model consistently by applying a role-assignment-based procedure. At the end, we show that the best found model generates realistic trajectories for 9 out of 10 situations.

2 Preface

This report shows the result for the graduation project of the master of Business Analytics. This graduation project is executed at the Machine Learning team at Beyond Sports in the form of an internship with a duration of six months.

First of all, I would like to express my gratitude to my supervisor Daniel Karavolos for guiding me for the past six months. Furthermore, I would like to thank all other members of the Machine Learning team at Beyond Sports for all the nice and insightful discussions. I also would like to thank Vincent François-Lavet for the nice discussions and suggestions during our 2-weekly meetings.

At the end, I would like to thank my family and friends. But especially my parents who always supports me in my decisions.

3 Abbreviations and Notation

Notation	Description
A	set of all actions
P	set of all attacking players
p	attacking player
e	segment in which the game is alive and ball-owning team does not change
D	Set of all extracted segments from the raw data
π	probability-distribution over actions
$S_{(p,t)}$	State of player p at time t
$C_{(p,t)}$	Cartesian Coordinates of player p at time t
$v_{(p,t)}$	velocity-vector of player p at time t
$a_{(p,t)}$	action (discretized $v_{(p,t)}$), taken by player p at time t
$a^{macro}_{(p,t)}$	macro intent action of player p at time t
$m_{(p,t^*)}$	boolean, indicating if the position of player p at time t is a macro intent
$v^{macro}_{(p,t)}$	velocity vector of player p at time t on the straight line towards the next macro intent
F^1, F^2, F^3	different Feature sets (see section 6.3)
P^{raw}_{θ}	raw-micro policy which corrects the macro policy in the HPN
P^{macro}_{Φ}	macro policy which guides the player towards a macro intent.
p^{true}	probability of selecting the true trajectory (see section 6.5.1)
p^{gen}	probability of selecting the generated trajectory (see section 6.5.1)
macro intent	desired position of a player on the field.
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
HPN	Hierarchical Policy Network

Contents

4 Introduction

Football is the most popular sport in many countries around the globe. Most matches played in the highest national leagues of these countries are broadcast live on television. Here, analysts often discuss the decisions made by the players during the break in a match. Visual tools are used to support their arguments, where, for instance, arrows can be drawn on the screen to highlight better decisions. However, these explanations are subjective and cannot be objectively confirmed due to a lack of analytical tools. So, one cannot prove- or disprove the analysts' hypothetical scenarios. For this reason, the need arises for tools that can visualize what would have happened in such situations. The way situations during matches evolve depends on multiple critical aspects. For instance, the position of a defender depends on the positions of attacking players[1] since covering attacking players is the main task of defenders. Moreover, the behavior of an attacking player depends on the ball-owning player. Thus, the way a situation evolves depends on multiple complex patterns which interact with each other. For this reason, the determination of how certain situations in football will evolve is a broad problem that can be divided into sub-problems.
Generating the trajectories for defending players is different from that for attacking players. The ultimate goal of attacking players is to score in the opponent's goal. This is mostly done by building up an attack where a player eventually shoots at the target in a promising situation. Building up an attack can be done in different ways which introduce high uncertainty. For instance, the attacking team in figure 1 is building up an attack where the ball-owning player (red) has the option to run along the line or to turn around. Therefore, the resulting trajectory of the ball-owning player is uncertain. This is different from the trajectories of players in the defending team since defending players in general react to the decisions of attacking players. Therefore, generating trajectories for attacking and defending players are considered to be different sub-problems. Some work is done in generating defensive trajectories [16], [7], and [4]. This is in contrast to attacking players which is seen as a harder problem due to the uncertainty.
In this thesis, we focus on generating trajectories of attacking players. Here, we take a true situation with a duration of n_2. Then, the problem is to generate a realistic trajectory for a single attacking player in time-interval $[n_1, n_2]$ where $[0 < n_1 < n_2]$. Furthermore, we assume that the attacking team does not change in the entire situation and the game stays active. We describe the implementation of different generative models and compare these by using quantitative and qualitative evaluation procedures. We show that the best found model can generate realistic trajectories for 4 seconds.

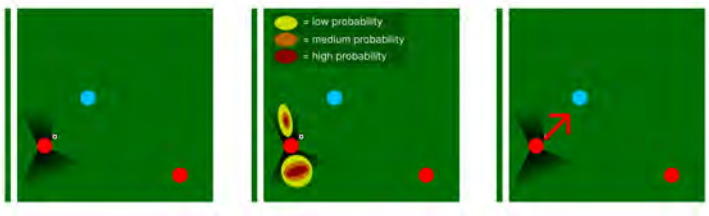

Figure 1: The model will suggest a movement towards the defender when the problem is approached as a regression problem (red arrow in the right image). This direction is a trade-off between 2 directions with high likelihood. However, the result itself is unlikely to happen.

[1] an attacking player is a player in the ball-owning team

The remainder of this thesis is structured as follows: In the Related Work section, we discuss literature that is taken into consideration to solve the problem. In the methodology, the procedure to solve the problem is explained. This includes the explanation of the data-preprocessing, feature sets, models and evaluation metrics. Then, in the Experimental Setup, we explain different experiments to find the best performing model. The results of these experiments are shown and discussed in the Results and Discussion sections respectively.

5 Related work

Part of the previous section was the introduction to the problem. In the next step, the literature which can be used to solve the problem is discussed. In general, we observe three sub-problems. First of all, computing features that explain the player formations is not a straightforward task. In section 5.1 we explain this problem and provide potential solutions from the literature. Then, we discuss different approaches for generative modeling in section 5.2 where the uncertainty in decision-making is taken into account. The third problem is about the evaluation procedure which is discussed in section 5.3. The discussed literature will be used to construct an approach which is described in section 6.

5.1 Features

In team sports, a group of players are working together to accomplish a task or multiple tasks. Here, different players can have different roles which can change over time. It is not obvious from the data what the role of each player is. In our setting, we have information about all players which is concatenated in a random order (see section 6.1). There are 11 players in one team which means that information about players in a single frame can be expressed in $11! \times 11!$ ways. Therefore, feeding the information to models in a random order is inconsistent which makes training models very hard, if not impossible. So, we need to come up with a solution to have a consistent representation of the player-data.

One option to solve this problem is procedure to learn the right role assignment for each player [8]. Here, a Hidden Markov Model with Gaussian emissions is trained to define the spatial distribution of players with a specific role. The locations of these distributions are then used in a role-assignment procedure: every observed player in a frame can be assigned to a role using the Hungarian algorithm. This makes sure that the data can be ordered consistently. Furthermore, this approach makes it possible to develop models for different roles separately. The behavior of a keeper is much different compared to a striker. So, we can develop a Keeper and striker model which simplifies the task. However, 22 players must be observed by the device. This is a problem when not all players are detected or at least one player has received a red card earlier in the match.

the observed players and the ball in basketball can be represented as 4-layer images [9]. Here, one half of the field is represented as a 400 by 380 grid. In these grids, a cell can be either 0 or 1, indicating if an entity is located in that particular cell. In the first and second layer of the image, the objects represent the positions of the players in the attacking, and defensive team respectively. the third layer represents the player in focus and the fourth layer represents the location of the ball. However, authors Devin Pleuler [11] developed a model that can predict in which direction the ball-owning player will pass the ball. Here, all coordinates of the players are mapped into an image as well. But instead of using binary values, continuous values are used indicating the distance from the concerning point to its closest object. This representation gives more information compared to the approach discussed before [9] as it gives an idea of where spaces on the pitch are. So, representing the player formations as images is another approach. In this approach, it is not necessary to have information about all players. However, it is not possible to build separate models for different roles since the roles are unknown.

Another potential representation of the player data is using Graph Neural Networks [12]. Here, Graph Neural Networks are used to compute local and global features about the player formation

at a given moment. Using graph neural networks neglects the player data representation problem since the final output does not depend on the order of player data. In this approach, not all players need to be observed. However, this procedure does not provide the option to model different roles separately.

5.2 Generative modelling

Some work has been done in the field of generating trajectories for defending players in team-sports [4], [7], [14] . However, only a few focus on generating attacking player trajectories. The main reason is that the behavior of attacking players contains a lot of uncertainty compared to defensive players [16]. In general, attacking players define the game while defensive players react to these decisions. Nevertheless, there is literature available about attacking player behavior. For instance, the work done by [9]. Here, a Hierarchical Policy Network (HPN) is developed. Their model architecture is based on the paradigm that the behavior of attacking players can be decomposed into micro and macro intents. Here, micro and macro intents are goals for the short- and long-term respectively. For instance, an attacking player could have a macro intent to reach a certain position on the pitch. Then, the macro policy is a straight path to that macro goal. However, this will not be the true policy when a defender is positioned in between the concerning attacking player and his macro intent. Then the micro policy should guide the attacking player around the defending player. This model accounts for the uncertainty by providing a distribution over the next positions at each step instead of suggesting one position. Then, the next position can be defined by sampling from the provided distribution. The results show high-quality trajectories, especially in the long term. Then, as a continuation, a Variational Recurrent Neural Network is developed to predict the trajectories of attacking basketball-players [5], [6]. Here, trajectories can be generated by sampling from a multivariate Gaussian Distribution and decode these into actions which are likely to happen. The model takes the features describing the macro intents of all players into account. In this way, the model can generate realistic trajectories for multiple attacking players in the same situation simultaneously.

5.3 Evaluation metrics

In generative modeling, it is hard to evaluate generated since there are no clear rules which help indicate the quality of a generated trajectory. Decisions of attacking players are at most unlikely but certainly not impossible. However, their exists multiple metrics which quantify the quality of defending player trajectories in American Football [16]. Most of these metrics are focused on quantifying good player behavior. For instance, they state that at least one defender should move toward the ball-owning player to increase pressure. This statement is then used to develop a metric. However, it could happen that professional football players are making worse decisions. In our problem, a good model should generate with a certain probability trajectories where a player is not necessarily behaving optimally: the goal is mimicking instead of optimizing. It is therefore not a good idea to use this kind of metrics since it will penalize trajectories where the player is not behaving optimally. Instead, some models are evaluated qualitatively by a panel of domain experts [9], [5], [6].

6 Methodology

In this section, we discuss the proposed approach. Here, we make use of identified solutions, described in section 5. First, the preprocessing of the data is explained. Then, we will explain how to take the uncertainty into account by approaching the problem as a classification problem. In section 6.3, we discuss how features are extracted from the preprocessed data. Different types of models which will be investigated are discussed in section 6.4. The procedure to evaluate the different models is explained in 6.5.

6.1 Data preprocessing

There is data available that describes 400 matches from the Eredivisie, played during the season 2018/2019 and 2019/2020. Data about a match contains a recording of 25 frames per second for 2 hours. each frame in the data contains the coordinates of all players and the ball. Furthermore, the current status of the match (dead or alive[2]) and ball-owning team (Home-team or Away-team) are known for each frame. Since the frames are recorded with a rate at 25 frames per second, each match contains around 180.000 frames. This results in a dataset with 72.000.000 frames in total. Due to time constraints, we decided to only use 42 matches. developing models based on all data is considered as future work. Next, some pre-processing needs to be done since certain rows are irrelevant. For instance, the device is logging data during the break of a match. This is a problem since we are not interested in situations where the game is not active. The pre-processing is described below.

6.1.1 Segment extraction

We define segments that satisfy the following constraints: The ball-owning team does not change and the ball status stays alive during the entire sequence. Complex situations like ball interceptions and fouls are avoided in this way and will be considered as future work. Then, we flip the coordinates of all players and the ball if the ball-owning team is on the right half of the field. This is possible because a football field is symmetric in the y-axis. By doing this, we make sure that the ball-owning team is always on the left half of the field such that the target of the ball-owning team is to score in the goal on the right half of the field. Afterward, we expect to see a general flow from left to right of all players. Furthermore, we skip alternate frames to reduce the original rate of 25 frames per second. This has increased the performance for similar studies [7], [8].

At this moment, we have a preprocessed dataset D of segments e of consecutive frames. Here, each frame describes the positions of all observed players and the ball. From each segment $e \in D$, we compute all actions and feature-sets, according to the procedures described in 6.2.1 and 6.3 respectively.

6.2 Labeling

As mentioned before, the goal is to generate realistic trajectories. As stated in section 4, the decision of a player in the attacking team is uncertain. This uncertainty in decision-making leads to different trajectories which are not necessarily unrealistic. The final solution should behave similarly. That is: different roll-outs for the same situation should not always be similar. Different approaches, for

[2]dead indicates inactivity. For instance, when the ball is out or right after a goal etc.

instance, the generative models, discussed in section 5.2 [5], [6], [9] do not predict the true trajectory of the concerning player. Instead, it models the uncertainty by predicting the distribution over the next positions at each step. Positions which have a high probability are more likely to occur in reality. Then, trajectories can be generated by sampling from this estimated distribution at each step. In this way, each roll-out will be different. A similar approach is used in this study and is explained in the next section.

6.2.1 Actions

To define the next position of a player, we will introduce velocity vectors $v_{(p,t)}$. These are defined as follows:

$$v_{(p,t)} = C_{(p,t)} - C_{(p,t-1)} \tag{1}$$

Here, $C_{(p,t)}$ is the Cartesian Coordinate of player p at time t. So, $v_{(p,t)}$ is the difference between 2 consecutive positions. Notice that $C_{(p,t)}$ can be recovered when $C_{(p,t-1)}$ and $v_{(p,t)}$ are known:

$$C_{(p,t)} = C_{(p,t-1)} + v_{(p,t)} \tag{2}$$

Thus, knowing the position at the start of a trajectory and the velocity vectors at all other steps provides all information to recover the original trajectory. Therefore, trajectories can be constructed by generating velocity vectors. This simplifies the problem since the velocity of a ball-owning player is limited ($\leq$ 80 cm per frame, which is 36 km/h). So, the norm of the velocity vector $v_{(p,t)}$ is limited as well which limits the search space.
To deal with the uncertainty, we discretize the search space. Here, we transform the velocity-vectors $v_{(p,t)}$ into an action $a_{(p,t)}$. This is the action, executed by player p at time t and is a one-hot encoding in a 21 by 21 grid. the cell at index (11, 11) can be interpreted as the origin which means that the player is not moving. one single cell to the right increases the velocity in the X-direction with 8. Furthermore, a single cell upwards increases the velocity in the y-direction with 8. This idea is explained in figure 2. Now, instead of predicting an action immediately, we estimate $\pi|S_{(p,t)}$. Here, π is the probability distribution over actions, and $S_{(p,t)}$ represents the state of player p at time t. So, $\pi|S_{(p,t)}$ is the probability distribution over actions, given the concerning situation. Then, $a_{(p,t)}$ can be defined by sampling from $\pi|S_{(p,t)}$.

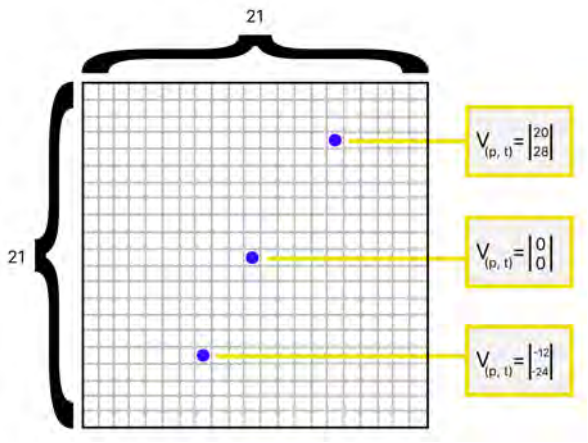

Figure 2: Action space explained, $v_{(p,t)}$ is discretized. Notice that these velocities are based on 25 frames/sec. Now, the problem is to classify the right action.

6.3 Features

As explained in 6.1, The segments extracted from the data do not have the right format to serve as input for the models. Therefore, we need to compute features from these sequences. In this section, we describe different sets of features. We introduce 3 feature sets: F^1, F^2, and F^3. The first feature set F^1 contains 16 basic features. Then, feature set F^2 is a concatenation of F^1 and a latent representation of formation images where the observed coordinates of all players and the ball are mapped. The third feature set F^3 contains information about all players, where the features about players are ordered using a role-assignment procedure. These are explained in the next paragraphs.

6.3.1 F^1: Base features

The first feature-set contains 16 base features about the concerning player, its closest attacking-, and defending player, and the ball. From the concerning player, we define the current and previous Cartesian Coordinates (integers). This will help the model since the behavior of a player depends on its current location. The previous coordinates are taken into account to give the model information about its current direction and velocity. Furthermore, the closest home-, and away team-player are defined. From these players, the Cartesian Coordinates and distances are computed. Note that information about all other players is not taken into account. The Cartesian Coordinates and velocity of the ball are taken into account. This could help since players tend to move towards the ball. In the end, we have 12.1 million instances in 61532 trajectories in the training set and 0.5 million instances in 3155 trajectories in the validation-set.

6.3.2 F^2: Base features + image representation

The base features only describe the current and previous positions of the concerning player, the closest hometeam-player and the closest awayteam-player. It does not take information about other players into account. As described in section 6, the player formation can be represented as images [11] [9]. Therefore, we decided to represent the players and the ball as a 3-layer image. each layer is a 32 by 16 image which represents the football field. each pixel in the first and second layer represents the distance to the closest attacking and defensive team-player respectively. Furthermore, each pixel in the third layer represents the distance to the ball. The final image contains 1536 pixels in total. an example is shown in Figure 3. However, there will be a large number of features from which many of them are correlated. For example, the values of 2 pixels right next to each other will have almost the same values, since their distances to the closest player are similar. Therefore, it is a good idea to compress the information in the image into a latent representation. This will be done by training a Variational Autoencoder (VAE) [3].

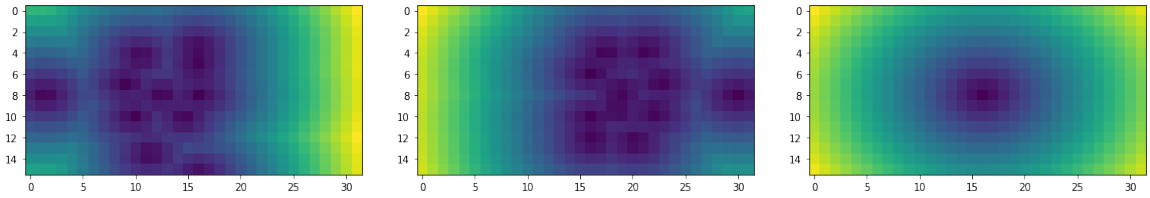

Figure 3: Player- and ball coordinates represented as a 3-layer image, each pixel in the first, second and third channel represents the distance from the concerning point to its closest attacking team-player, defensive team-player and the ball respectively

VAE's are models which have an encoder- and decoder network. The encoder estimates the parameters of a prior distribution in latent space Z based on the original data F. In this study, we assume a Multivariate Gaussian Distribution (μ, σ) The decoder does the other way around: it estimates the parameters of the posterior distribution of the original data given a point in Z. a point with a high likelihood in this posterior distribution with estimated parameters by the decoder network should be similar to the original data F. In this study, we assume a multivariate Gaussian distribution as posterior distribution with $\tilde{\mu}, \tilde{\sigma}$ as parameters. This is because the formation images are based on measured coordinates of the players and the ball and we assume that the error of these measurements follows a normal distribution. To reduce complexity, we assume that the variances of all pixels in the formation images are the same at any time. Therefore, parameters $\tilde{\sigma}$ of the posterior distribution become a single learnable parameter, which is not part of the VAE. The network is trained by minimizing the negative Evidence Lower Bound (ELBO) using gradient descent.

$$ELBO = RL - KL \tag{3}$$

Here, RL stands for the reconstruction loss and corresponds to the likelihood of observing the original data F under the posterior distribution from which the parameters are estimated by the decoder network. Then, KL stands for the Kullback-Leibler divergence and is a measure of the similarity between 2 distributions. This term makes sure that the encoded data follows a multivariate Gaussian Distribution with $\tilde{\mu} = 0$ and $\tilde{\sigma}^2 = 1$ (see section A.3 for the derivation). This has some nice properties. First of all, instances that occur more often in the original data should have a higher likelihood according to the proposed Multivariate Gaussian Distribution. So, the encoding network reveals information about the likelihood of instances. This can be used to identify the likelihood of certain formations which can be useful for future work. Next to this, we can generate random formation images which are likely to occur by sampling from the Multivariate Gaussian Distribution in Z, and decode this point using the decoder.

Now, we use the estimated $\mu \in Z$ of the encoded formation image as additional features. We choose μ because this point has the highest probability-density in the Multivariate Gaussian Distribution. Then, the second set of features F^2 will be a concatenation of the latent representation of the formation images and feature set F^1. The number of instances and trajectories are the same as feature set F^1.

6.3.3 F^3: Role-assignment based features

Another way to take information about all players into account is via a role-assignment procedure [8]. Here, a Hidden Markov Model with Gaussian emissions is trained to define the distribution of the visited positions on the field for each of the players (see figure 4). Then, a Hungarian algorithm [2] is used to assign a role to each of the observed players at the start of the segment. Here, we assume that the roles do not change during a segment. In this way, the information about the different players is consistently ordered which makes it possible to take this information into account. In this study, we will use existing work. The training data contains 196.867 trajectories based on 17.897 segments. Furthermore, the validation data contains 7304 trajectories based on 664 segments.

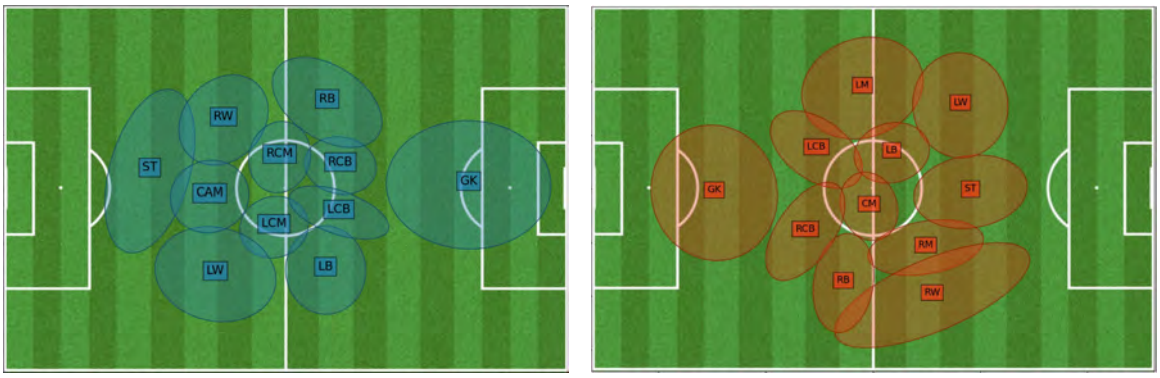

Figure 4: distribution of positions of different roles in the defending team (left) and attacking team (right). These are used to assign roles to the players at the beginning of a segment.

6.4 Models

Now that the problem is defined and useful feature-sets are defined, the next step is defining models which can estimate the distribution over actions $\pi|S_{(p,t)}$ (see section 6.2.1). We will investigate three different models. First, The Multilayer Perceptron (MLP) is described in 6.4.2. Then the Recurrent Neural Network is described in section 6.4.3 and the Hierarchical Policy Network is described in section 6.4.4. Section 6.4.1 describes the general setup, including the type of model and loss function.

6.4.1 General setup

The actions of a player in the ball-owning team during a segment depend on multiple complex patterns. For instance, a player tends to move towards good positions which depends on complex interactions between the coordinates of all players on the field. The model should be able to capture these patterns to provide an accurate estimate of $\pi|S_{(p,t)}$. For this reason, all models which will be investigated during this study are deep neural networks since these are flexible in capturing complex patterns [10]. Each of these neural networks will have 441 output nodes (21 by 21) on which a Softmax activation is applied [10]. In this way, the output of all models can be interpreted as $\pi|S_{(p,t)}$. We train all models by minimizing the following Cross-Entropy loss:

$$L = -\sum_{e \in D} \sum_{t=0}^{n_2} \sum_{p \in P} \sum_{a \in A} y_a \log[\hat{y_a}] + (1 - y_a) \log[(1 - \hat{y_a})] \tag{4}$$

Here, y_a is a boolean indicating if the concerning action was executed or not. Furthermore, $\hat{y}$ is the probability of the concerning action, estimated by the model. All models are developed and trained using Gradient Descent [10]. The architectures of the different models are described in the next paragraphs.

6.4.2 Multi-layer Perceptron

The first type of model we will investigate is the Multi-layer Perceptron (MLP) [10]. This is a neural network that takes features $F_{(p,t)}$, describing $S_{(p,t)}$ as input. It contains at least 2 layers on which non-linear activation functions are applied. Due to this non-linearity and having multiple layers, the model can capture high complex patterns. This makes it possible to fit the complex patterns,

explained in the introduction of this section. Once the model is trained, trajectories can be generated by algorithm 1. This algorithm is visualized in Appendix A.5.

6.4.3 Recurrent Neural Network

Another type of model which we will investigate are Recurrent Neural Networks (RNN). This is a sequential model which is able to remember important historical information [13]. It has shown to achieve high performance in many sequential problems. There are multiple different RNN's, from which we will use Gated Recurrent Units (GRU) [13]. GRU's show high performance in similar tasks [9], [5], [6] which is the reason to choose for this model. It can save important information for both short and long term in one hidden state. The cell has 2 gates: a reset-gate and an update-gate. The reset-gate defines what information on the short term should be taken into account. Besides, the update-gate defines which short term and long term information should be remembered. Trajectories can be generated according the procedure, described in algorithm 1 and is shown in appendix A.4.

Algorithm 1 Generating trajectories

compute $F_{(p,n_1)}$ which represents $S_{(p,n_1)}$ ▷ Burn-in in time-interval [0 and n_1]
if MLP **then**
  estimate $\pi|S_{(p,n_1)}$ using MLP
else
  estimate $\pi|S_{(p,n_1)}$ and $H_{(p,n_1)}$ using RNN
end if
get action $a_{(p,n_1)}$ by sampling from $\pi|S_{(p,n_1)}$
for $n_1 \leq t \leq n_2$ **do** ▷ Generate in time-interval [n_1 and n_2]
  execute $a_{p,t}$
  derive $S_{(p,t+1)}$
  compute $F_{(p,t+1)}$ which represents $S_{(p,t+1)}$
  if MLP **then**
    estimate $\pi|S_{(p,t+1)}$ using MLP
  else
    estimate $\pi|S_{(p,t+1)}$ and $H_{(p,t+1)}$ using RNN
  end if
  get $a_{(p,t+1)}$ by sampling from $\pi|S_{(p,t+1)}$
end for

6.4.4 Hierarchical Policy Network

Models which generate trajectories often have poor performance in the long term. As discussed in section 5.2, a hierarchical Policy Network (HPN) can be used to deal with this problem [9]. The results show high performance. Therefore, it is decided to investigate this framework in this study. To explain the framework, some notation needs to be introduced. Let π^{raw}, π^{macro} be the raw-micro policy and macro policy such that:

$$\pi^{raw} \odot \pi^{macro} = \pi \tag{5}$$

The final distribution over actions π is decomposed into a raw-micro policy and a macro policy. The idea is that the raw-micro policy optimizes the short term behavior while the macro policy guides the player towards a macro goal. The final policy π is the Hadamard-product of the raw-micro policy and macro policy and can be interpreted as a trade-off between long- and short-term behavior. The

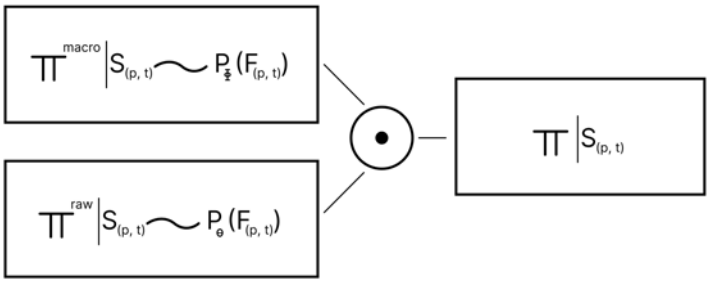

Figure 5: Overview HPN

procedure to develop an HPN is explained in the next paragraphs.

Macro intent labeling heuristic The first step is defining the macro intent of a player. It is unclear from the data what these macro intents of all players are. The creators of the HPN solved this by introducing a heuristic that extracts weak labels from the data. All points in a trajectory, where the concerning player is not moving ($\|v_{(p,t)}\| < \epsilon$) are considered as macro intent. However, the creators of the HPN [9] applied the framework to basketball which is different from football in many aspects. From the data, we observe that attacking football players are often stationary or moving relatively slow. Thus, considering stationary points will not work. Instead, we decided to consider the points where an attacking player is changing direction as macro intents. Let $m_{(p,t)}$ be a boolean indicating if player p at timestep t is changing direction and is defined as follows:

$$m_{(p,t)} = \begin{cases} True & \text{if } v^x_{(p,t)} < 0 < v^x_{(p,t+1)} \\ True & \text{if } v^x_{(p,t)} > 0 > v^x_{(p,t+1)} \\ True & \text{if } v^y_{(p,t)} < 0 < v^y_{(p,t+1)} \\ True & \text{if } v^y_{(p,t)} > 0 > v^y_{(p,t+1)} \\ False & \text{otherwise} \end{cases}$$

Now, the macro-action $a^{macro}_{(p,t)}$ is defined by drawing a straight line between $C_{(p,t)}$ and the first point $C_{(p,t^*)}$ where $m_{(p,t^*)} = True$ and $t^* > t$. Then, $v^{macro}_{(p,t)}$ is the average velocity vector on the straight path towards $C_{(p,t^*)}$. By discretizing $v^{macro}_{(p,t)}$ we get $a^{macro}_{(p,t)} \in A$. This idea is explained in figure 6.

Multi-stage learning approach We apply a multi-stage learning approach to make sure that the raw-micro and macro policy behave according to the intended paradigm. First, the macro policy is introduced as P^{macro}_{Φ} which is an RNN, parameterized by Φ. First, good values for Φ need to be found such that 6 is minimized:

$$L_{macro} = -\sum_{e \in D} \sum_{t=0}^{n_2} \sum_{p \in P} \sum_{a \in A} y^{macro}_a \log[\hat{y}^{macro}_a] + (1 - y^{macro}_a) \log[(1 - \hat{y}^{macro}_a)] \tag{6}$$

This makes sure that the values in Φ are in an area where the macro policy behaves according to the proposed paradigm. However, the action towards a macro intent is often not the true action. So, we introduce P^{raw}_{Θ} which is another RNN parameterized by Θ, representing the raw-micro policy. This policy corrects the macro-policy and can be seen as short-term optimization. finding good values for Θ is done by minimizing 4. Here, $y = a_{(p,t)}$ and $\hat{y} = P^{macro}_{\Phi}(F_{(p,t)}) \odot P^{raw}_{\Theta}(F_{(p,t)})$ Parameters Φ are

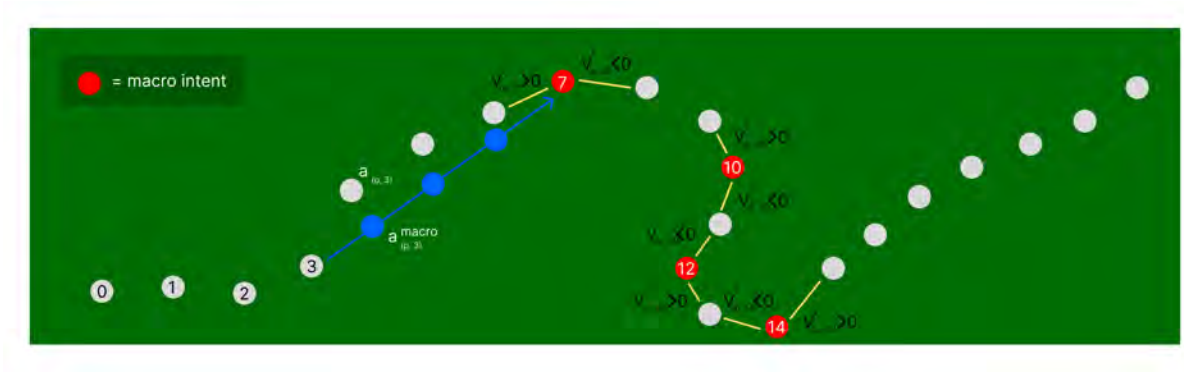

Figure 6: Macro intents and Macro actions explained. Consider position 7 in the shown trajectory which is a macro intent since $v^y_{(p,7)} > 0 > v^y_{(p,8)}$. Now, the macro-action at time-step 3 can be defined by drawing a straight line towards its first macro intent (blue line) and take the average velocity vector. The same procedure holds for all other steps.

frozen during this step to make sure that the raw-micro policy learns to correct the macro policy. In the last stage, parameter-sets Θ and Φ are both unfrozen. Then, the entire HPN is fine-tuned by minimizing equation 4. An overview of this procedure is shown in Figure 7. The algorithm to generate trajectories is the same as described in 1.

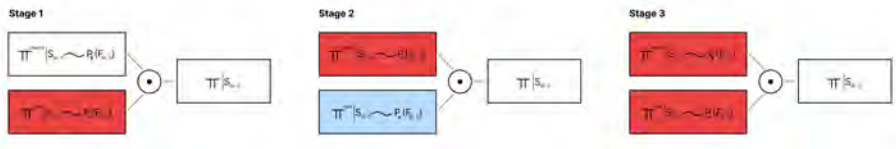

Figure 7: Overview of training procedure HPN. In the first stage, only the macro policy is trained where parameters Φ are considered (red) and macro-actions are used as labels. Then, in the second stage where parameters Φ are frozen (blue), the raw-micro policy is learned to correct the macro policy. The entire model is fine-tuned in the final stage. The true actions are used as labels in stage 2 and 3

6.5 Evaluation metrics

So far, we use the cross entropy-loss as optimization-function to train the proposed model. Once an optimal solution is found, the resulting model can estimate $\pi|S_{(p,t)}$. So, it provides a probability distribution over actions, conditioned on the concerning situation. However, using the cross-entropy loss as evaluation metric alone will not suffice.
In the first place, a generated trajectory is a sequence of consecutive samples from an estimated π. The model could perform well in predicting the next action, given a certain situation. However, this does not mean that the combination of consecutive actions in a trajectory is likely to happen.
Next to this, the cross-entropy loss is a measure of uncertainty. Higher values indicate more variance in the distributions. Now, consider the distributions shown in 8. The actions right next to each other in the left image are much more similar compared to the two actions in the right image since they are in almost the same direction with almost the same velocity. Therefore, sampling

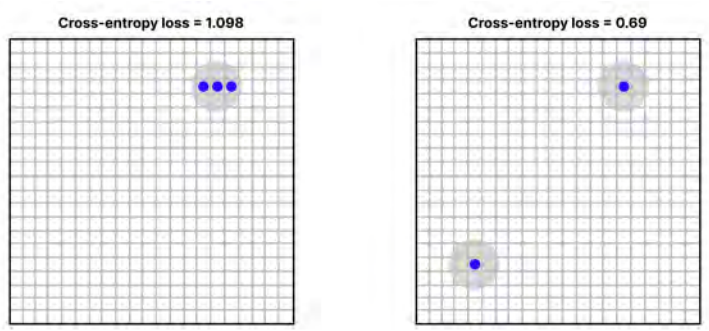

Figure 8: π on the left is distributed over 3 similar actions. This will lead to similar trajectories which can be seen as low uncertainty. However, π on the right is distributed over 2 different actions which will result in different trajectories (high uncertainty). The Cross-Entropy loss indicates the opposite. Therefore, it can not be fully trusted.

for multiple steps from the left distribution will result in similar trajectories. Namely, a trajectory toward the top right. However, the variance in the distribution on the right is lower, but sampling will let the player move towards the top right or bottom left. This is caused by the fact that the similarity between actions is not fixed. Thus, a model which estimates distributions with relatively high variances could still be able to generate high-quality trajectories, if π is mostly spread out over multiple similar actions instead of fewer "less" similar actions. So, using the Cross-Entropy loss as an evaluation metric alone will not suffice. Therefore, next to the cross-entropy loss, we inspect the generated trajectories visually and evaluate the models in a qualitative way.

6.5.1 Questionnaire

We let a panel evaluate the generated trajectories according to the same procedure, described by the creators of the HPN [9]. Here, the goal is to define if the generated trajectories by the models are realistic. Due to the risk of losing interest during the evaluation, We do not consider all models but select the best 3 models, based on the lowest cross-entropy loss and qualitative evaluation of the generated trajectories. The experiments are constructed as follows: First, we select different segments (e.g. free-kick, long-range shot etc.) from the validation set. Then, for every segment, one of the models will generate a trajectory for a forward player in the attacking team for 50 frames (4 seconds). Then, each of these trajectories is shown in an animation. Here, an animation first shows the original trajectories of all players up to a certain setpoint n_1. Then, after this setpoint, both the true and generated trajectories of the concerning player are shown but all other players and the ball are frozen (see section 9.1 for the reasoning). Now, the task for an individual is to define which of the 2 trajectories is the most realistic one. An example is given in figure 9.

Figure 9: example of an experiment. In the first and second images, the true situation is shown. In the third and image, the true and generated trajectories are shown while all other players are frozen. Each individual defines which trajectory looks the most realistic.

left-tailed binomial test We will use the obtained results to define how realistic the generated trajectories are. Now, suppose that the generated trajectory in an animation is realistic. Then, an individual cannot distinguish between the true and generated trajectory since both are realistic. Thus, an individual will select the true trajectory or generated trajectory, both with 0.5 probability. However, when the generated trajectory is not realistic, then it is less likely that an individual will choose the generated trajectory. This means that the probability of choosing the generated trajectory by an individual (p^{gen}) is less than 0.5. These statements can be used to define the following hypothesis:

$$h_0 : p^{gen} = p^{true} \tag{7}$$

$$h_1 : p^{gen} < p^{true} \tag{8}$$

So, for each animation, we can test these hypotheses by conducting a statistical test. When h_0 is approved, we conclude that the panel is not able to recognize the true trajectory meaning that the generated trajectory is realistic. Furthermore, when h_0 is rejected, we conclude that the generated trajectory is not realistic. Now, we have N persons in the panel. Then, the number of individuals from the panel who selects the generated trajectory, under h_0, has a binomial distribution with parameters $p = 0.5$ and $n = N$. Therefore, we use a left-tailed binomial test to test the hypothesis. In all tests, 0.05 is chosen to be the significance threshold.

7 Experimental setup

We discussed multiple feature sets and model architectures in section 6. Now, we need to define an experimental setup to investigate which combination of feature set and model will give the best result. The first step is finding a good VAE to construct feature-set F^2. These experiments are explained in 7.1. After this, experiments to find the best MLP, RNN and HPN are discussed in sections 7.2, 7.3, and 7.4 respectively. Note that we do not tune Hyperparameters since this does not necessarily improve the performance (see section 8.3.4 for the reasoning).

7.1 Variational Autoencoder

The VAE is used to derive latent features, describing the formation of all players and the ball as explained in section 6.3.2. We will execute multiple experiments to find a good VAE. As architecture, we investigate Convolutional Neural Networks (CNN) [4] and MLPs as VAE's. CNN's shows high performance in many image-related problems [11] and [9]. This is the reason for choosing this architecture. On the other hand, the image is entirely based on the formation of the image. Therefore, a model which captures the formation of the players is automatically able to reconstruct the image. These formations are complex, and therefore we will use MLP's as second type of model because these are flexible in capturing complex patterns. We will investigate different architectures for both types. These are listed in 1. All CNN-VAE have 2 convolutional layers. The first layer has 3 input channels, 8 output channels, a kernel size, padding and stride of 7, 3 and 3 respectively. The second layer has 8 input channels, 16 output channels, a kernel size, padding and stride of 3, 2, 1 respectively. This architecture is based on the work, done by Devin Pleuler [11]. The model is trained by using gradient descent for 500000 iterations. Here, the Adam optimizer will be used [1].

Table 1: Different VAE-architectures considered in this study.

MLP	Architecture: input-hidden1-hidden2-latent	CNN	hidden1-latent
MLP-VAE-32	1536-128-64-32	CNN-VAE-64	64-32
MLP-VAE-64	1536-256-128-64	CNN-VAE-64	128-64
MLP-VAE-128	1536-512-256-128	CNN-VAE-128	256-128
MLP-VAE-256	1536-1024-512-256	CNN-VAE-128	512-256

7.2 Multi-Layer Perceptron

The first and most straightforward model we will investigate is the MLP. It contains 2 fully-connected hidden layers (FC) on which ReLU-activation) is applied. The output layer represents the action-space and has 441 nodes. Here, a softmax-activation is applied. This type of model is not able to capture complex patterns over time, since it is not a sequential model. Therefore, we expect a lower performance of this model, compared to the RNN and HPN. For this reason, we decided to study this model based on feature set F^1 only. All MLP's are trained for 200000 iterations where the Adam optimizer is used [1].

Table 2: Different MLP-architectures considered in this study

Name	Architecture: input-hidden1-hidden2-output
MLP-F1-16	16-16-32-441
MLP-F1-32	16-32-64-441
MLP-F1-64	16-64-128-441

7.3 Recurrent Neural Network

Concerning the RNN, we consider a fixed architecture. This is based on the HPN [9]. First, there are 2 fully-connected layers (FC) on which a ReLU-activation is applied, then a batch-normalization is applied to normalize the latent features which stabilizes training [15]. After this, a GRU is used which can capture important information about the past. Then, after another batch normalization, 2 fully-connected layers are used. We will investigate the performance of the RNN on 2 different feature sets: F^1 and F^2. The experimental setup is shown in table 3. All RNN models are trained for 200000 iterations by using the Adam optimizer [1].

Table 3: layer-sizes of considered RNN models

Name	Architecture: input-hidden1-GRU-hidden2-output
RNN-F1-16	16-32-32-64-441
RNN-F1-32	16-64-64-128-441
RNN-F2-150	144-150-150-200-441
RNN-F2-300	144-300-300-400-441

7.4 Hierarchical Policy Network

For both the raw-micro and macro policy networks, we use the same architecture as the RNN. We assume this model to be the most promising. For this reason, we will use feature sets F^2 and F^3. When using feature set F^2, we will train a single model. However, by using feature set F^3, we can train separate models for each of the roles. We only investigate the Striker model in this study, due to time restrictions. The best found setup will be used to train models for all other roles. The different experiments we will execute are listed in table 4.

Table 4: layer-sizes of considered HPN's

Name	raw-micro architecture:input-hidden1-GRU-hidden2-output	macro architecture
HPN-F2-150	144-150-150-200-441	144-150-150-200-441
HPN-F2-300	144-300-300-400-441	144-300-300-400-441
HPN-F3-16	229-50-50-100-441	229-50-50-100-441
HPN-F3-32	229-150-150-250-441	229-100-100-200-441

7.5 questionnaire

There are 33 experiments in total (explained in section 6.5.1). For the three best models, 10 experiments are constructed. Furthermore, 3 dummy experiments are constructed to define the quality of

the panel. These are similar to a general experiment, but instead of showing a generated trajectory, we show a frozen player. This indicates that the concerning player is staying stationary for a long period and can be seen as unrealistic. Therefore, all individuals should select the true trajectory in these dummy experiments. Otherwise, the quality of the panel is low.
There are 22 individuals in the panel who executed the experiments individually and right after each other. Furthermore, experiment 4, 14 and 29 are the dummy experiments. It is allowed to repeat an animation in order to get a better understanding of the concerning situation. After the execution, there is time reserved to discuss the experimental setup and get useful insights from the participant.

8 Results

In this section, the results of the different experiments are discussed. In section 8.1, we discuss some general observations. After this, we discuss the results concerning the VAE in section 8.2. The best VAE is then used to define latent features, in order to construct feature set F_2 (see section 6.3.2). Then, In section 8.3, the results of the different models are compared. This includes a comparison in Cross-Entropy loss, visual explanation of generated trajectories, based on situations from the validation set, and qualitative evaluation. In section 8.3.4 we will draw conclusions from the obtained results

8.1 General observations

Macro intent labeling heuristic Some results of the macro intent labelling heuristic are shown in 10. Here, the blue line represents a true trajectory of an attacking player somewhere on the pitch. As we can see, the heuristic performs well in most cases since all points where the player is changing directions are detected by the heuristic. However, it sometimes classifies a point on a straight path as macro intent. Nevertheless, this is not an issue because these labels are only used to make the macro-policy behave as intended by the paradigm.

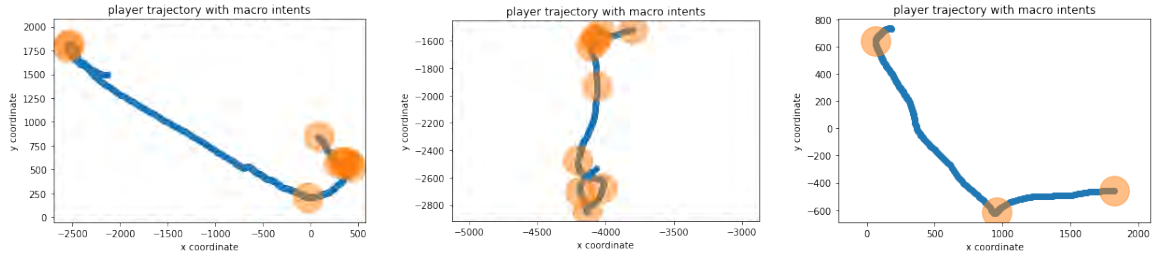

Figure 10: examples of macro intent labeling heuristic.

Overall Distribution over (macro) actions the left image in Figure 11 shows the general distribution over actions of all players. As we can see, the actions where an attacking player moves toward the right occur more often than actions towards the left. This confirms the general flow from left to right, as stated in 6.1. Furthermore, most of the actions are relatively close to the center which means that an attacking player is in general moving slowly. This was expected since a player is most of the time not actively involved in the build-up. Furthermore, the majority class is slowly moving towards the right (8 cm/frame which is 1 meter per second). The image on the right in Figure 11 shows the general distribution of the macro actions. As we can see, actions towards the right are over-represented which indicates a general flow from left to right in terms of macro intents. However, this distribution is different from the general distribution of the true actions. For instance, there are more macro-actions towards the top- and bottom right, meaning that the macro intent of a player is often towards the top-, or bottom-right.

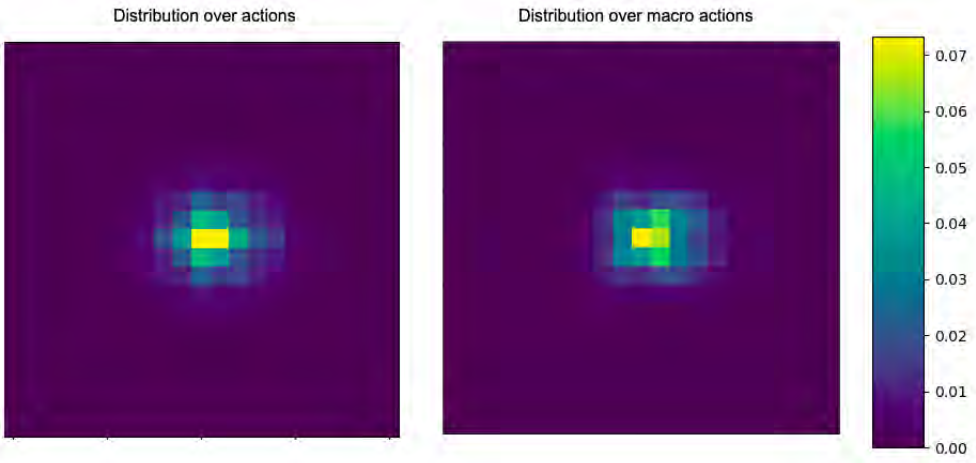

Figure 11: Overall distributions of $a_{(p,t)}$ (left) and $a^{macro}_{(p,t)}$ (right)

8.2 Variational Autoencoder

The results of all experiments concerning the VAE are shown in table 5. In general, the MLP-VAE outperforms the CNN-VAE in terms of ELBO-loss. Note that not all experiments concerning the CNN-VAE are executed due to time restrictions. Next, we can see that the ELBO-loss does not decrease much more when the latent dimension is greater than 128. Therefore, the MLP-VAE-128 is the best found model. So, this model will be used to define the additional latent features for feature set F_2.

Table 5: Negative Variational Lower Bound (ELBO) of different VAE's

Model	negative ELBO
CNN-VAE-128	2532
MLP-VAE-32	2516
MLP-VAE-64	1941
MLP-VAE-128	1699
MLP-VAE-256	1641

In figure 12 we see a true and reconstructed formation image. As we can see, the reconstructed formation image is similar to the true formation image which shows that most of the information stays intact during encoding and decoding. In Figure 13, a formation image is generated by the best found VAE. The quality is lower since the shapes are rougher compared to true formation images. However, most of the characteristics of true formation images are observable in the generated images.

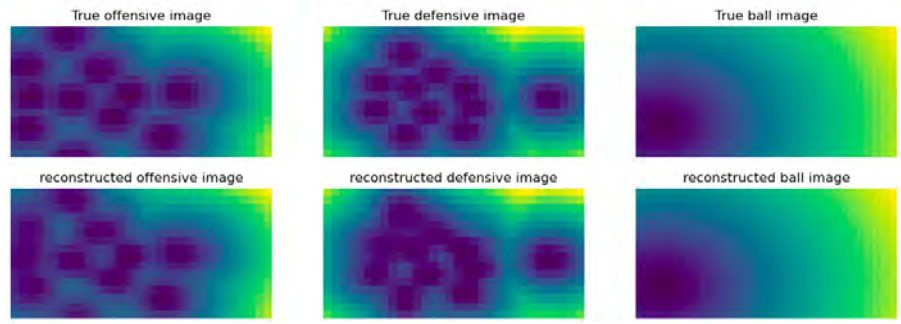

Figure 12: reconstruction performance of best found VAE (MLP-VAE-128).

Figure 13: generated formation image by VAE. the left, middle and right image represents the attacking players, defensive players and the ball respectively.

8.3 Evaluation

In this paragraph, the behavior of the different models is compared. Here, we consider the Cross-Entropy loss, a qualitative evaluation of the generated trajectories and the results of the questionnaire evaluation (see section 6.5.1).

8.3.1 Cross-Entropy loss

In table 6, the cross-entropy loss of all models is shown. As we can see, the loss of all MLP models is around 1.3 which means that increasing the complexity does not increase the performance. Then, the loss of the different RNN models is around 1.15. This shows that feature set F_2 does not increase the performance of the RNN models. Furthermore, increasing the complexity of the model does not decrease the loss. However, all RNN models performs better than the MLP models. So, including a memory cell decreases the loss. Concerning the HPN models, the loss is similar to the RNN models when feature set F_2 is used, but is higher when feature set F_3 is used. This is the result of only taking the striker trajectory into account since the behavior of a striker is more uncertain.

Table 6: Cross-Entropy loss different MLP models on validation-set

Model	Cross Entropy-loss	Model	Cross Entropy-loss	Model	Cross Entropy-loss
MLP-F1-16	1.31	RNN-F1-16	1.13	HPN-F2-150	1.14
MLP-F1-32	1.28	RNN-F1-32	1.16	HPN-F2-300	1.16
MLP-F1-64	1.3	RNN-F2-150	1.15	HPN-F3-100	1.31
		RNN-F2-300	1.16	HPN-F3-200	1.26

8.3.2 Generated trajectories

In this paragraph, we show and discuss 4 generated trajectories per model. All stated arguments in this paragraph are subjective and serves to explain the behavior of the models.
In each of the Figures 14, 15, 16, 17, and 18, the blue and red lines represent the true and generated trajectories respectively. Then, the yellow and purple dots represent the attacking- and defending team respectively at the moment that the model starts generating. Furthermore, the first 2 images show generated trajectories of a length of 50 frames (4 seconds), while the third and fourth images show generated trajectories with a length of 100 frames (8 seconds). First, all models are discussed individually in the next 5 paragraphs. In the last paragraph, the different models are compared.

MLP-F1 In Figure 14, four generated trajectories by the MLP-F1 are shown. The trajectory in the top left image concerns the goalkeeper. In reality, the goalkeeper stays close to the goal which is confirmed by the true trajectory. However, the generated trajectory is drifting in a forward direction. In the top right image, the true trajectory is in a left-forward direction with a smooth turn to the right while the generated trajectory is drifting in a left-forward direction. The generated trajectory looks shorter due to the drifting behavior. In the bottom left image, the true trajectory goes in a backward direction and eventually makes a turn to the left. This makes the concerning player move towards the right side of the field. Once the model starts generating, it decides to make a turn to the right which is the opposite of the true trajectory. This makes the concerning player move towards the left side of the field. In the bottom right image, the true trajectory immediately makes a turn to the left in backward direction. After approximately 5 meters, the trajectory makes another turn to the left toward the right side of the field. The generated trajectory is drifting at the start and eventually moves towards the right side of the field.
In general, we observe a drifting behavior which is confirmed by all 4 images. This can be seen as unrealistic. Therefore, we conclude that the MLP-F1 cannot generate realistic trajectories.

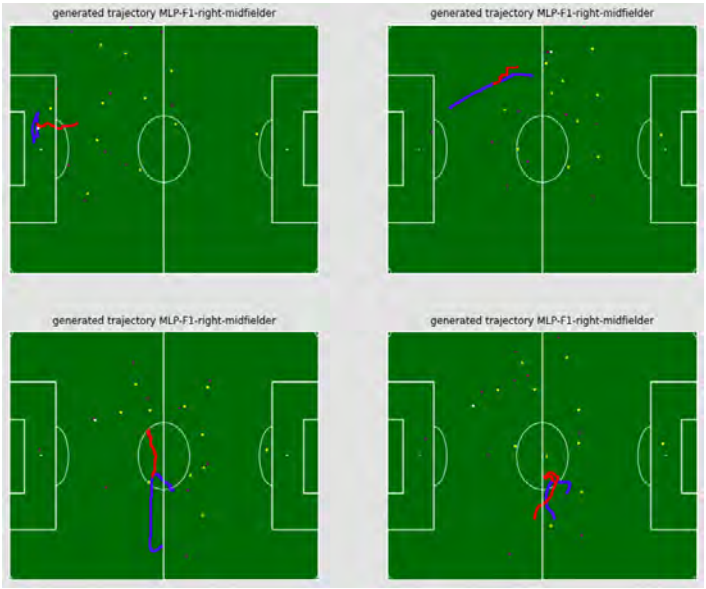

Figure 14: Generated trajectories using MLP-F1.

RNN-F1 The generated trajectories by the RNN-F1 are shown in Figure 15. In the top left image, the true trajectory is in a forward direction. However, when the model started generating, it immediately decided to make a turn to the right towards the right side of the field where the ball was positioned. In the top right image, the true trajectory starts in a right-forward direction and makes a turn to the left after approximately 3 meters. At the same point, the model decides to make a turn to the right in the opposite direction of the true trajectory. In the bottom left image, the goalkeeper launched the ball forward. The true trajectory starts in a right-forward direction where it eventually makes a smooth turn to the left. However, when the model starts generating, it immediately decides to make a turn to the right. Then, when the ball flies over, the model decides to make a turn to the left towards the ball. In the bottom right image, the trajectory starts in a backward direction and makes a smooth turn to the left. Then, after approximately 2 meters, the true trajectory makes a u-turn towards backward direction. At the same point, the generated trajectory stays moving toward the right side of the field.
In general, the generated trajectories by the RNN-F1 are smooth in contrast to the MLP-F1. This could be the result of taking a memory cell into account. However, the model tends to move towards the ball. This could be realistic, Here, it is unrealistic that a player moves toward the keeper, knowing that the ball will fly over. So, the RNN-F1 performs better than the MLP-F1, but is still unrealistic in the majority of the cases.

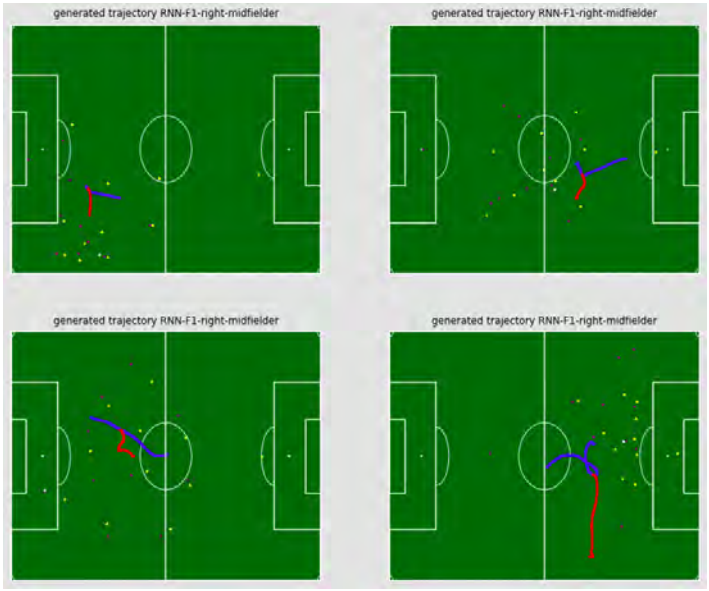

Figure 15: Generated trajectories using RNN-F1

RNN-F2 The trajectories, generated by the RNN-F2 are shown in Figure 16. In the top left image, the true trajectory starts in a backward direction. Then, after approximately 2 meters, a u-turn is made towards a right-forward direction. Once the model starts generating, it quickly decides to make a smooth turn to the left in a forward direction. This is different from the true trajectory. However, the ball-owning player could have decided to launch the ball in a left-forward direction if the concerning player moved according to the generated trajectory. Therefore, the generated trajectory is reasonable. In the second image, the true trajectory starts moving in a backward direction. After approximately 2 meters a turn to the right is made. The generated trajectory is approximately the same as the true trajectory. The true trajectory in the third image starts in left-forward direction and eventually makes a turn to the right and ends close to the goal. During generating, the model is moving slower than the true trajectory. Moving slowly in this situation is unrealistic since it is a promising build up. Here, the concerning player should move toward a good position as quickly as possible. In the fourth image, the true trajectory starts moving toward the right side of the field and stays stationary after approximately 8 meters. In contrast, the generated trajectory decides to keep moving toward the right side of the field. In this situation, the generated trajectory concerns the left-forward player. It is unrealistic that the left forward player moves toward the right side of the field since the ball is on the left side of the field. Therefore, this trajectory is unrealistic.
In summary, the RNN-F2 can generate realistic trajectories as is shown in the top left and right images. But some generated trajectories are unrealistic as is shown in the bottom left and right images.

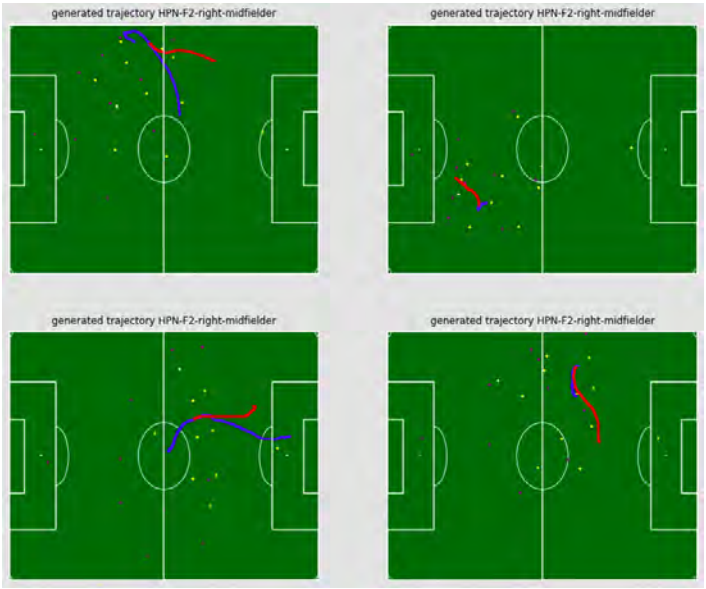

Figure 16: Generated trajectories using RNN-F2

HPN-F2 The trajectories, generated by the HPN-F2 model are shown in Figure 17. Concerning the top left image, the generated trajectory is similar to the true trajectory. However, the model eventually decides to make a slight turn to the left toward the ball. This decision is not unrealistic since it is still close to the true trajectory. Then, concerning the top right image, the situation is promising. Here, the model decides to position itself in front of the goal which is a realistic scenario. however, the true trajectory is towards the right side of the field. In the bottom left image, the model decides to make a sharp turn to the left towards the right side of the field. However, this is unrealistic since the ball is launched toward the left side of the field where the concerning player is positioned. Therefore, a decision towards the left side of the field, similar to the true trajectory, is the only realistic scenario. In the bottom right image, the generated and true trajectories are similar. Therefore, the generated trajectory is realistic.
In summary, most of the generated trajectories are realistic as shown in the top left, top right, and bottom right images. However, the trajectory in the bottom left image is unrealistic which shows that the model could generate unrealistic trajectories.

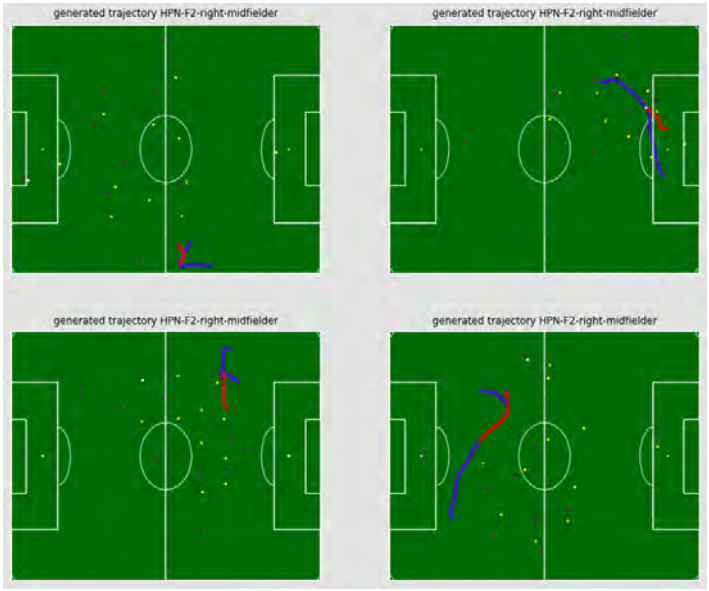

Figure 17: Generated trajectories using HPN-F2

HPN-F3 The generated trajectories by the HPN-F3 are shown in Figure 18. In the top left image, the model decides to make a turn to the left. After this, the model keeps going straight which is unrealistic. The generated trajectory in the second image is similar to the true trajectory where both are moving in a forward direction. In the third image, the model decides to make a turn to the right which is realistic. Then, the model goes straight while being off-side which is unrealistic. Then, in the bottom right image, the model eventually decides to make a sharp turn to the left while the turn in the true trajectory is also to the left in approximately the same position, but less sharp. In general. The model changes directions on points close to the points in the true trajectories. This indicates that the model can define when it has reached a macro intent. Then, it often defines a macro intent that is different from the true trajectory which is shown in the bottom left and right images.

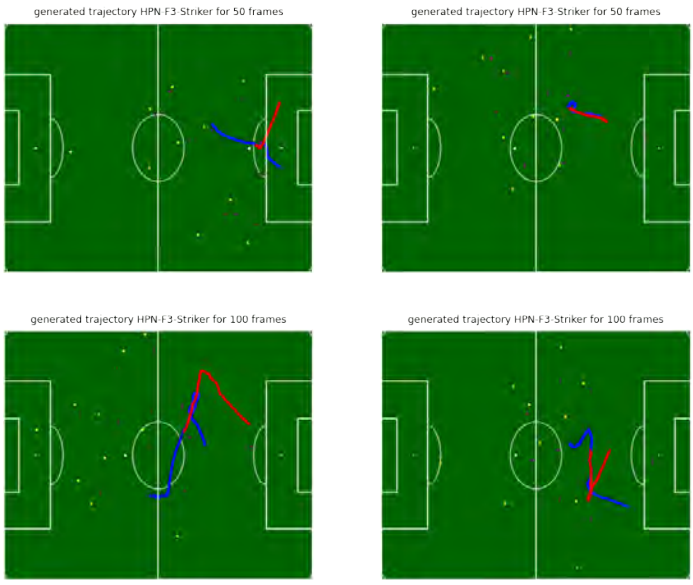

Figure 18: Generated trajectories using HPN-F3

Next to the generated trajectories, we can also analyze the behavior of the micro and macro policy of the HPN. In general, the macro policy guides the player toward the next macro intent and the micro policy makes the turn toward the next macro intent look smooth and natural. Furthermore, the micro policy is also responsible for the acceleration and deceleration of the player. Next to this, we observe high velocities for the micro policy and low velocities for the macro policy. An example is given in Figure 19. Here, the macro policy guides the player toward the top right. However, the raw-micro policy makes the player move toward the top. Concerning the HPN-F2, we observe the same behavior.

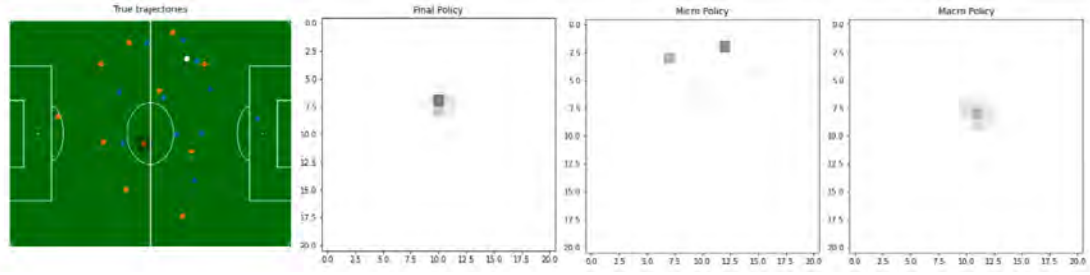

Figure 19: visualization of raw-micro, macro and final policy of the central midfielder (marked by black shadow).

Comparison After all, the generated trajectories by the MLP-F1 are of low quality due to their drifting behavior and random overall direction. The trajectories, generated by the RNN-F1, RNN-F2, HPN-F2, and HPN-F3 are smooth which is an improvement, compared to the MLP-F1. However, the RNN-F1 tends to move toward the ball in contrast to the RNN-F2, HPN-F2, and HPN-F3. Moreover, the generated trajectories by the HPN-F2 and HPN-F3 are different from the RNN-F2

since these are able to generate trajectories that have sharp turns in contrast to the RNN-F2. This shows that the quality of the generated trajectories by the HPN is higher. In general, all models sometimes stays stationary or repeat the same action for long periods. Repeating the same action for a long period is shown in the left bottom image of both Figures 16 and 17. These unrealistic events are unpredictable. According to the described observations, we consider the RNN-F2, HPN-F2, and HPN-F3 as best models. Therefore, these models are selected for the questionnaire. These results are discussed in the next paragraph.

8.3.3 Questionnaire

The results of the questionnaire, described in section 6.5.1 and 7.5, are discussed in this paragraph. In table 7, the results per model are shown. Here, a cell contains 2 values, indicating the number of votes on the generated trajectory and true trajectory respectively. Besides, the color indicates the result of the binomial test where red means that h_0 is rejected and green means that h_0 is approved. As explained in 6.5.1, the generated trajectory in an experiment is realistic when h_0 is approved. Now, h_0 is approved for 5 out of 10 experiments concerning the RNN-F2, 6 out of 10 concerning the HPN-F2 and 9 out of 10 concerning the HPN-F3. This means that the HPN-F3 generates the most realistic trajectories, followed by the HPN-F2 and RNN-F2 respectively.

Table 7: results of the questionnaire.

experiment nr.	RNN-F2	HPN-F2	HPN-F3
1	4 / 18	9 / 13	12 / 10
2	11 / 11	0 / 22	7 / 15
3	3 / 19	3 / 19	10 / 12
4	7 / 15	8 / 14	8 / 14
5	11 / 11	6 / 16	10 / 12
6	7 / 15	15 / 7	9 / 13
7	6 / 16	12 / 10	5 / 17
8	5 / 17	9 / 13	15 / 7
9	5 / 17	4 / 18	8 / 14
10	14 / 8	9 / 13	10 / 12
total	73 / 147	75 / 145	100 / 120

We also included 3 dummy experiments from which the results are shown in Table 8. The results of the second and third dummy experiments are as expected since the majority selected the true trajectory. However, the first dummy experiment shows unexpected results. Here, 13 participants decided to select the trajectory where the concerning player stays stationary. We found out that being stationary in the first dummy experiment is often considered to be realistic since multiple defenders are positioned around the concerning player.

Table 8: results dummy experiments

experiment nr.	4	14	29
total	13 / 9	3 / 19	2 / 20

8.3.4 Overall comparison

All models are evaluated by comparing the Cross-Entropy loss, a qualitative evaluation of the generated trajectories, and a questionnaire. When we only consider the results of the Cross-Entropy loss (see section 8.3.1), we conclude that the RNN-F1, RNN-F2, and HPN-F2 show the highest performance. However, this is not confirmed by the qualitative evaluation and questionnaire. For instance, the results of the qualitative evaluation show that the generated trajectories by the RNN-F1 are often unrealistic. Furthermore, the Cross-Entropy loss of the HPN-F3 was higher than the RNN-F1, RNN-F2 and HPN-F2, but the quality of the generated trajectories is more realistic according to the results obtained by the questionnaire. This discrepancy is not unexpected and confirms the statements made in section 6.5. Thus, optimizing the Cross-Entropy loss does not necessarily optimize the quality of the generated trajectories. Furthermore, we neglect the Cross-Entropy loss in the further analysis of the results.

The obtained results from the qualitative evaluation and the questionnaire are similar. For instance, according to the qualitative evaluation, the HPN generates higher quality trajectories since it can make sharp turns. Besides, the HPN-F2 and HPN-F3 outperform the RNN-F2 according to the results obtained by the questionnaire. From these observations, we conclude that the HPN architecture can generate better trajectories, compared to the RNN-F2. However, the HPN-F3 outperforms the HPN-F2 in the questionnaire. This is unexpected since the behavior of the HPN-F2 and HPN-F3 is similar to the qualitative evaluation. There are multiple explanations. First of all, the HPN-F2 failed in generating 2 trajectories for the second and ninth experiments in the questionnaire. The trajectory was stationary in the second experiment resulting in 0 votes by the panel. Besides, the generated trajectory in the ninth experiment was considered to be unrealistic since the trajectory is towards the ball-owning player who takes a corner kick. The HPN-F3 did not fail in any of the 10 generated trajectories. This distorts the results since we observed stationary behavior by the HPN-F3 as well in the qualitative evaluation. More experiments need to be done to increase the reliability of the questionnaire.

In summary, the Cross-Entropy loss is not a good metric for evaluating the generated trajectories. Furthermore, both HPN-F2 and HPN-F3 show the best generated trajectories according to the qualitative evaluation. However, the HPN-F3 outperformed the HPN-F2 which could be the result of the low number of experiments. According to the obtained results, we consider the HPN-F3 as the best model.

9 Discussion and future work

In the end, we have shown that our best model can generate realistic trajectories according to the used evaluation metrics. However, there are multiple limitations concerning our proposed approach. In this section, we discuss these limitations and provide potential solutions for future work.

9.1 Interaction effects

In section 4 we introduced the goal of this thesis that solves a sub-problem of a broader problem. It turns out that solving the different sub-problems separately has limitations. In this study, we only focused on generating attacking player trajectories. Other sub-problems like generating defending player trajectories and ball-trajectory, defining pass probabilities etc. are not taken into consideration. It is hard to generate attacking player trajectories without having solutions of all other sub-problems due to the multiple complex interaction effects. For instance, defending players cover attacking players in general. Now, consider a situation where an attacking player can go left or right. In the true situation, the player decides to go to the left side of the field. The defending player who covers the attacking player reacts to this decision by moving toward the left side of the field as well. Thus, the attacking and defending player are moving toward the left side of the field. Now, suppose we use the model to generate a trajectory for the concerning attacking player and it decides to move to the right, then the defending player should also move towards the right. Thus, the true trajectory of the defending player does not fit the generated trajectory of the attacking player since this trajectory is toward the left side of the field. So, the combination of the true defender trajectory and generated attacking trajectory is unrealistic. Besides, unrealistic situations are not represented in the data. Thus, the model has never seen these situations which can lead to poor performance of the model. Next to this, evaluating the generated trajectories is difficult since the true trajectory of the defending player makes the situation unrealistic. For this reason, we can only look at how realistic the shape of a trajectory is. But we cannot define how realistic a decision of a player is since this is dependent on the trajectories of all other players on the field.

9.2 Features

During this study, we have investigated three different feature sets. It turns out that using feature set F^2 or F^3 (see sections 6.3.2 and 6.3.3) provides the best-generated trajectories. However, using these feature sets have some limitations which are described in this paragraph.

One approach in this study to represent the player formation was via image representation. These images are expensive to compute since the distance to the closest entity should be defined for each pixel. Next to that, the computed formation image needs to be encoded by the encoder network of the VAE to compute the latent features. This procedure is done at each step to generate a trajectory. This can be unnecessarily complex. Therefore, it is suggested to consider other approaches for future work. For instance Graph-Neural-Networks (see section 5.1).

Then, none of the investigated feature sets describes information about the player itself. For instance, the model does not know if the player is "Martinez" or "Veerman". So, the model generates trajectories which behave similarly to an average player in the Eredivisie. However, the model could be improved when information about the concerning team or player is taken into account. This can be done by introducing team or player embeddings. These embeddings are latent representations of

specific teams or players and could decrease the uncertainty in decision-making. Furthermore, the embeddings provide useful information and can be applied to other areas as well.

9.3 Models

In this study, we have investigated three different models as explained in section 6.4. However, the general setup and models have some limitations which are discussed in this paragraph.

In the first place, all investigated models generate trajectories by sampling actions from a discretized actionspace A. Predicting the right action for the current moment is a simple task. However, the combination of multiple consecutive predictions often leads to unrealistic trajectories as is shown by the MLP-F1 model. The HPN has solved this problem partially. However, there are still challenges left. For instance, we have only focused on generating trajectories for a single player at each time. The ultimate goal is to generate trajectories for multiple players simultaneously. This is possible by extending our setup. Here, we take the macro intents of all players as extra features into account [5], [6]. However, this solution is complex and there may other potential approaches where complete trajectories of multiple players are generated in one step. For instance, there is a Generative Adversarial Network that can generate the trajectories of 5 defending basketball players [14] in one step.

Then, all investigated models show straight lines in their generated trajectories which is the result of repeating the same action for a longer period which is unrealistic. Here, the same action results in the same velocity vector that results in a straight line However, adding noise to the generated velocity vector could be a solution to fix this kind of straight trajectories. Another solution could be to use Variational Recurrent Neural Networks due to their continuous actionspace.

9.4 Evaluation

Three different evaluation approaches are applied: the Cross-Entropy loss as a quantitative evaluation metric, a qualitative evaluation of the generated trajectories, and a questionnaire. However, these metrics has limitations which are discussed in this paragraph.

During this study, we found out that the Cross-Entropy loss cannot be used to quantify the quality of the generated trajectories. Thus, in the end, the quality of the generated trajectories is not expressed in quantitative evaluation metrics. As discussed in section 5.3, it is an option to create customized evaluation metrics. For instance, we know that the maximum rates of acceleration and deceleration are limited. Furthermore, the maximum rate in changing the direction of a player depends on the velocity and is limited as well. These statements can be used to construct quantitative evaluation metrics that can provide more insights into the performance of the models.

The questionnaire served as the third evaluation metric. However, there are some limitations. For instance, the number of experiments for the investigated model is low (10). Therefore, we suggest to execute more experiments to make the results more reliable. On the other hand, we conduct a statistical test for each of the experiments where we assume a significance threshold of 0.05. However, we have executed 33 experiments. Thus the probability of making an error of type 1 for at least 1 experiment is relatively high. We suggest to apply techniques that reduce the probability of making an error of type 1.

References

[1] Diederik Kingma Jimmy Lei Ba. "ADAM: a method for stochastic optimization". In: *ICLR* (2015).

[2] Ayorkor Mills-Tettey Anthony Stentz Bernardine Dias. "The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs". In: *Robotics Institute Carnegie Mellon University* (2007).

[3] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8237. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[4] Marc Schmid Patrick Blauberger Martin Lames. "Simulating Defensive Trajectories in American Football for Pedicting League Average Defensive Movements". In: *Frontiers* (2021).

[5] Eric Zhan Yisong Yue Stephan Zheng Long Sha Patrick Lucey. "Generating multi-agent trajectories using programmatic weak supervision". In: *ICLR* (2019).

[6] Eric Zhan Yisong Yue Stephan Zheng Patrick Lucey. "Generative Multi-Agent Behavioral Cloning". In: *ICLR* (2018).

[7] Hoang Le Peter Carr Yisong Yue Patrick Lucey. "Data-Driven Ghosting using Deep Imitation Learning". In: *mo* (2021).

[8] Hoang M. Le Yisong Yue Peter Carr Patrick Lucey. "Coordinated Multi-Agent Imitation Learning". In: *International Conference on Machine Learning* (2016).

[9] Yisong Yue Stephan Zheng Patrick Lucey. "Generating Long-term Trajectories Using Deep Hierarchical Networks". In: *NIPS* (2016).

[10] Umberto Michelucci. *Applied Deep Learning*. Apress, 2018.

[11] Devin Pleuler. "Player Masks: Encoding Soccer Decision-Making Tendencies." In: *NESSIS* (2021).

[12] Michael Stöckl Thomas Seidl Daniel Marley Paul Power. "Making Offensive Play Predictable Using a Graph Convolutional Network to Understand Defensive Performance in Soccer". In: *MIT SLOAN* (21).

[13] Fathi M. Salem. *Recurrent Neural Networks From Simple to Gated Architectures*. Springer, 2022.

[14] Guande Wu Shuya Zhao Jianzhe Lin Claudio Silva. "Basketball GAN: Sportingly Acceptable Trajectory Prediction". In: *AISA* (2021).

[15] Sergey Ioffe Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: (2015).

[16] Lin Lee Cheong Xiangyu Zeng Ankit Tyagi. "Prediction of Defensive Player Trajectories in NFL Games with Defender CNN-LSTM Model". In: *NGS* (16).

A Appendix

A.1 Rewriting hadamard product

The Hadamard-product in the Hierarchical Policy Network should be interpreted as follows:

$$\pi^{raw}|S_{(p,t)} = \sigma(Y^{raw}|F_{(p,t)}) \tag{9}$$

$$\pi^{macro}|S_{(p,t)} = \sigma(Y^{macro}|F_{(p,t)}) \tag{10}$$

$$\pi^{raw}_{(p,t)}|F_{(p,t)} \odot \pi^{macro}_{(p,t)}|F_{(p,t)} = N(\sigma(Y^{raw}|F_{(p,t)}) \odot \sigma(Y^{macro}|F_{(p,t)})) = \sigma(Y_{raw}|F_{(p,t)} + Y_{macro}|F_{(p,t)}) \tag{11}$$

Here, $Y_{raw}|F_{(p,t)}, Y_{macro}|F_{(p,t)}$ are the output values before the Softmax activation of the raw micro-policy and macro policy respectively. σ stands for the Softmax-activation and N is a function which scales the sum of the probabilities to 1.

A.2 Design choices Recurrent Neural Network

input shape during training It is necessary to have entire sequences as inputs, in order to learn the model which information from the past should be taken into account. The input for the RNN will have the following shape: $[1, n_2, F^x]$. Here, n_2 is the length of the sequence and is variable. F^x represents the size of the feature space. Note that we only feed 1 sequence at each time to the model.
In many sequential problems, it is common to create batches by concatenating multiple sequences. Here, padding is applied to make all sequences of the same length. However, we decided to avoid this approach for two reasons.
First of all, we do not want to learn the model detecting the end of a sequence This causes problems when the model incorrectly predicts that the sequence is over. Besides, we believe that the end of a sequence (e.g. if the ball-status changes to "Dead") does not depend on the concerning player in most cases. Eventually, this should be done by a different model which will be considered as future work. Furthermore, imputing missing values for the features should be avoided as well. So, single sequences are used as input during training which introduces new problems.

batch-normalization The architecture which is used by [9] contains batch-normalization [15] on multiple layers. Here, the latent features of some hidden layers are normalized to stabilize training. This normalization is done by executing the following transformation:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{VAR[x^{(k)}] + \epsilon}} \tag{12}$$

Here, $\hat{x}^{(k)}$ represents the normalized feature, $x^{(k)}$ represents the computed feature before normalization, ϵ represents a small value to avoid dividing by 0. Furthermore, $E[x^{(k)}]$ is the expectation of the feature. However, $E[x^{(k)}]$ is estimated based on mini-batch statistics during training but the running statistics are used for evaluation. Therefore, the estimations of the mean and variances of the mini-batches should be unbiased to get similar results for training and evaluation. The mini-batch statistics in our setup are biased since the statistics of a single trajectory are different from the statistics of the data. For instance, the observed coordinates of the player in a trajectory are often in a small area of the field while players in the data can be in any area of the field. Since we only have one trajectory in a batch, the estimated variances of the batches will be much lower than

the true variances. Besides, the estimated means of the batches are biased towards the area of the concerning trajectory. We solve the problem by using the running statistics during training instead of the mini-batch statistics.

Variable sequence length When training a model using batch gradient descent. It is common to use the average Cross-Entropy loss per instance in a batch. In that case, each batch should have the same number of instances in it. Otherwise, the model will overfit on smaller batches. For this reason, we are forced to use the sum of the Cross-Entropy loss of all instances in the concerning sequence. However, the length of a sequence varies between 100 and 1800. This will result in larger gradients for longer sequences which could lead to unstable training. The learning rate should be reduced in that situation.
Another option could have been to split longer sequences into multiple smaller sequences. However, this could make it harder for the model to recognize long-term patterns. After splitting a sequence into 2, a decision that is made in the second sequence could depend on a feature in the first sequence. However, the sequences are treated independently during training which makes it impossible for the model to recognize these patterns. Verifying this statement in our problem will be considered as future work.

A.3 Detailed explanation Variational Autoencoder

The Variational Autoencoder can be seen as a regularized version of a general Autoencoder that has an encoder and decoder network. Here, the encoder network, parameterized by ϕ, maps the original data X into latent space Z and the decoder, parameterized by θ does the other way around: it maps points from latent space back to the original data X. However, the distribution of the encoded data in latent space Z of general Autoencoders can have any complex distribution. However, this distribution is regularized by Variational Autoencoders by incorporating the Kullback-Leibler divergence in the loss-function. The Kullback-Leibler divergence is a measure for the similarity between 2 distributions. This makes sure that the distribution of the encoded data is similar to some prior distribution (a multivariate Gaussian distribution with mean vector $\vec{0}$ and variance vector $\vec{1}$).

A.3.1 Mathematical derivation

First of all, we have data from which we assume that it represents reality. So, we need to come up with a loss that maximizes the likelihood of the observed data X. In other words, we will maximize $p(X)$. However, it is also allowed to maximize $\log(p(X))$ since the logarithmic function is monotonic. However, this objective does not depend on the distribution of $Z|X$. In order to regularize the distribution of $Z|X$ we need to incorporate this term by using the rule of conditional probability:

$$\log P_\theta(X) = \log \frac{P_\theta(X, Z)}{P_\theta(Z|X)} \tag{13}$$

Then, the next step is to multiply by $\frac{Q_\phi(Z|X)}{Q_\phi(Z|X)}$. Here, Q_ϕ is a proposal distribution for $P_\theta(Z|X)$, constructed by the encoder network.

$$\log P_\theta(X) = \log \frac{P_\theta(X, Z)}{Q_\phi(Z|X)} \frac{Q_\phi(Z|X)}{P_\theta(Z|X)} \tag{14}$$

Now, these terms can be rewritten into separate terms.

$$\log p(X) = \log \frac{P_\theta(X,Z)}{Q_\phi(Z|X)} + \log \frac{Q_\phi(Z|X)}{P_\theta(Z|X)} \tag{15}$$

Here, the right term is the Kullback-Leibler divergence of the $Q_\phi(Z|X)$ and $P_\theta(Z|X)$ but cannot be computed. However, know that the Kullback-Leibler divergence is always positive. This means that the left term is a lower bound of the log-likelihood of X. Therefore, the left term is the evidence lower bound (ELBO). Now, we rewrite the this right term is follows:

$$\log \frac{P_\theta(X,Z)}{Q_\phi(Z|X)} = \log P_\theta(X|Z) \frac{P(Z)}{Q_\phi(Z|X)} = \log P_\theta(X|Z) + \log \frac{P(Z)}{Q_\phi(Z|X)} \tag{16}$$

In this equation, $\log P_\theta(X|Z)$ is the log likelihood of observing the original data, under the posterior distribution from which the parameters are estimated by the decoder network. This can be seen as the reconstruction loss. Furthermore, the right term is the Kullback-Leibler divergence between the $Q_\phi(Z|X)$ (distribution of the encoded data by the encoder network) and $P(Z)$ (prior distribution which is the Multivariate Gaussian Distribution with mean vector $\vec{0}$ and variance vector $\vec{1}$). Now, the Variational Autoencoder which has an encoder network, parameterized by ϕ, and a decoder network, parameterized by θ can be trained by maximizing the ELBO.

A.4 RNN trajectory procedure

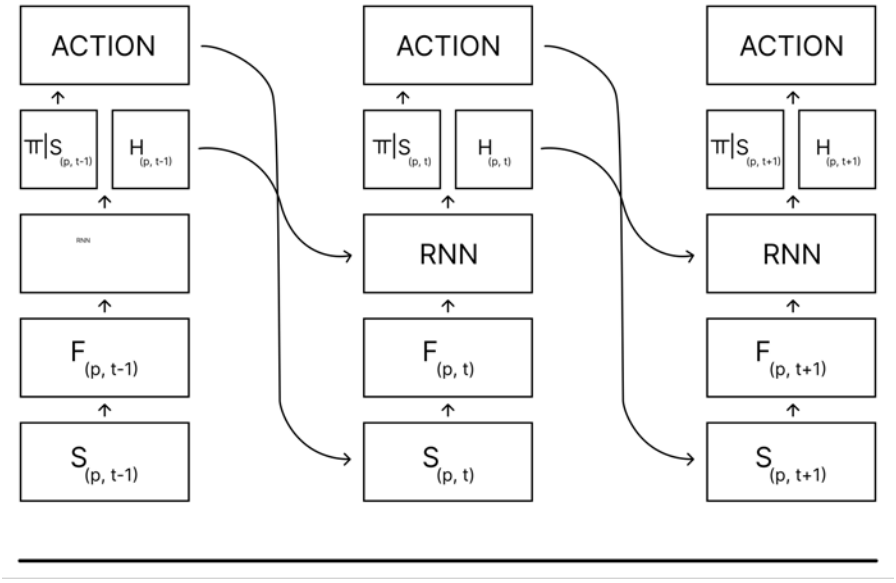

Figure 20: generating trajectory with MLP explained

A.5 RNN trajectory procedure

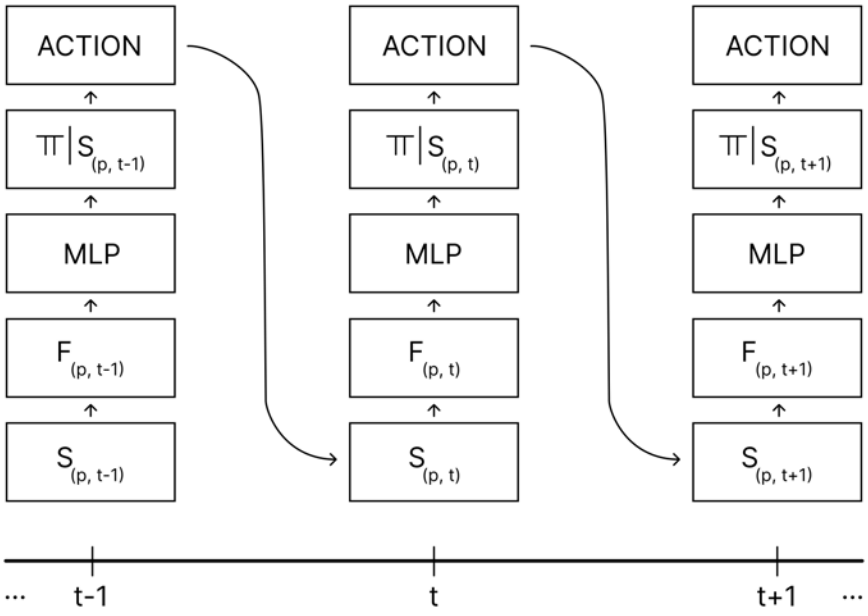

Figure 21: generating trajectory with MLP explained