KU LEUVEN

FACULTY OF ENGINEERING SCIENCE

Learning Contextualized Soccer Player Representations using Variational Autoencoders

Maxel Withofs

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor:
Prof. dr. J. Davis

Assessoren:
Maaike Van Roy
Prof. dr. D. Hughes

Begeleider:
Maaike Van Roy

Academiejaar 2021 – 2022

Preface

First of all, I would like to thank my daily supervisor, Maaike Van Roy, for her great advice and assistance when working on this thesis. I also thank my promotor Prof. dr. Jesse Davis and assessor Prof. dr. Danny Hughes for reading my text and for their guidance. Finally, I thank my family, friends, and girlfriend for their continuous support throughout the years.

Maxel Withofs

Contents

Abstract

In the field of soccer analytics, representing the playing style of players is one of the main tasks. This can be done using player representations, which are fixed-size vectors that aim to describe a player's behavior. With these player representations, one could do things like comparing players (e.g. to find similar players for scouting purposes) or monitoring a player's development, by studying the player's representation over time. Various action-based approaches exist that learn these representations, but they do not explicitly take into account the situations (or match contexts) in which the players perform those actions.

In this thesis, player representations are obtained that depend on the match context using different variations of Variational Autoencoders (VAEs) that are trained on event stream data of real-life soccer games. Three models are constructed, where each model builds on the previous one and addresses some of its problems. The final model is called the Variational Recurrent Ladder Agent Encoder (VaRLAE), which uses Recurrent Neural Networks (RNNs) to make use of the play history that is present in the data and a hierarchy of latent variables to embed the player information in.

The models are tested and evaluated by using the player representations that they learn in a series of tasks. These tasks include identifying players from anonymized data, where the VaRLAE model performs better than previous approaches, and finding similar players, in which similarities between players are captured to some extent. The representations also prove useful in the VAEP framework for valuing actions and rating players, where the performance on some important metrics like the Brier score is improved when they are used as extra features during training. Although the representations could be improved in terms of interpretability, they have the potential to be successfully incorporated in other soccer analytics tasks.

Samenvatting

In het gebied van voetbalanalyse is het weergeven van de speelstijl van spelers een van de belangrijkste taken. Dit kan worden gedaan met behulp van speler representaties, wat vectoren met een vaste grootte zijn die het gedrag van een speler beschrijven. Met deze speler representaties kunnen taken uitgevoerd worden, zoals spelers vergelijken (bv. om gelijkaardige spelers te vinden voor scoutingdoeleinden) of de ontwikkeling van een speler opvolgen, door de speler representaties op verschillende tijdstippen te bestuderen. Er bestaan verschillende actiegebaseerde benaderingen die deze representaties leren, maar daarbij wordt er niet uitdrukkelijk rekening gehouden met de situaties (of wedstrijd contexten) waarin de spelers deze acties uitvoeren.

In deze dissertatie worden speler representaties verkregen die afhankelijk zijn van de wedstrijdcontext met behulp van verschillende variaties van Variational Autoencoders (VAEs) die getraind zijn op event stream data van echte voetbalwedstrijden. Drie modellen worden geconstrueerd, waarbij elk model voortbouwt op het vorige en enkele van zijn problemen aankaart. Het laatste model heet de Variational Recurrent Ladder Agent Encoder (VaRLAE), die gebruik maakt van Recurrent Neural Networks (RNNs) voor het integreren van de spelhistorie die aanwezig is in de data en een hiërarchie van latente (of verborgen) variabelen om de spelersinformatie in te bedden.

De modellen worden getest door de geleerde representaties te gebruiken in een reeks taken. Deze taken bestaan uit het identificeren van spelers uit geanonimiseerde gegevens, waarbij het VaRLAE model beter presteert dan eerdere benaderingen, en het vinden van gelijkaardige spelers, waar gelijkenissen tussen spelers tot op zekere hoogte worden vastgelegd. De representaties blijken ook nuttig in het VAEP kader voor het waarderen van acties en het beoordelen van spelers, waarbij de prestaties op een aantal belangrijke metrieken zoals de Brier score verbeteren wanneer ze worden gebruikt als extra attribuut tijdens het trainen. Hoewel de representaties verbeterd kunnen worden op het gebied van interpreteerbaarheid, hebben ze het potentieel om met succes gebruikt te worden in andere voetbalanalysetaken.

List of Abbreviations

AUROC	Area Under the ROC-curve
CVAE	Conditional Variational Autoencoder
CVRNN	Conditional Variational Recurrent Neural Network
ELBO	Evidence Lower Bound
FAWSL	FA Women's Super League
KL-divergence, KLD	Kullback-Leibler divergence
LSTM	Long short-term memory
MRR	Mean Reciprocal Rank
NMF	Non-negative Matrix Factorization
PCA	Principal Components Analysis
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
SPADL	Soccer Player Action Description Language
t-SNE	t-distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder
VAEP	Valuing Actions by Estimating Probabilities
VaRLAE	Variational Recurrent Ladder Agent Encoder

Chapter 1

Introduction

Data analytics in sports is being used more and more these days. In the case of soccer, almost every professional football team now has a person or group of people who maintain player data and use this data in many different tasks. The primary goal of collecting this data is to use it for analyzing performances of the team's own players, but also getting insight in opposing teams and their tactics. At the end of the line, soccer teams want to do whatever they can to increase their chances at winning games. Soccer analytics can help with this in various ways.

One of the core tasks of soccer analytics is representing the playing style of players and teams. These player- or team representations are like their "fingerprints", they aim to gain insight in how the player or team acts, i.e. what actions they perform, in which match contexts. A match context refers to the situation in which the team or player finds themselves in during a game. For individual players, this can include the position on the field of the player, the current score of the match, the time that has passed, etc. With these representations, many different tasks can be done. By comparing player representations, we can find players that are similar to a target player, which can in turn be used to find players that fit a certain style for scouting. Another use is in match tactics, where a team can get to know the strategy or playing style of its opponent by analyzing its representation.

There already exist several approaches that try to capture playing style via representations in soccer, like Player Vectors [1] and SoccerMix [2]. However, these approaches do not sufficiently take into account the full match context in which players or teams display a certain playing style. Liu et al. [3] proposed a model that learns contextualized player representations for ice hockey. They developed a Variational Autoencoder (VAE) which learns the representations from a large dataset containing millions of events from real life games of the National Hockey League (NHL). It is this model that we will deploy in the context of soccer to learn player representations.

1.1 Problem

This thesis tries to achieve a number of goals:

1. Develop a Variational Autoencoder similar to that of Liu et al. [3] that learns contextualized representations of soccer players, and measure its performance.
2. Analyze and interpret the representations that come from the model, along with trying to determine which factors of the representations are important to distinguish players.
3. Examine the applicability of the learned representations in down-the-line analytics tasks like finding similar players and in frameworks for player performance evaluation like VAEP [4].

1.2 Approach

To tackle the problem statements above, the following steps were carried out. First, multiple variations of VAE architectures that learn the player representations were constructed. As a starting point, the simplest extension of a VAE was built: a Conditional Variational Autoencoder (CVAE), which includes match context as an input to the model. An extension to that model is a Conditional Variational Recurrent Neural Network (CVRNN), which takes play history into account by the usage of hidden states and sequences of actions as input instead of a single action at a time. The final extension is the Variational Recurrent Ladder Agent Encoder (VaRLAE), which is also the model described by Liu et al. [3] for the ice hockey case. In this model, a separate latent variable exists for each component of the context (the components are the actions themselves, the result of the actions, and the game state), and these latent variables are dependent on one another. This hierarchy of latent variables essentially solves the posterior collapse problem (where the network ignores the latent variables to compute its outputs) from which the CVRNN model suffers.

For training and testing the models, we use event stream data from real-life games of the 2019/2020 La Liga season, and of the 2019/2020 and 2020/2021 seasons of the FA Women's Super League. Event stream data describes soccer games as a series of on-the-ball actions, where each action is presented by a number of features like the type, location, and timestamp of the action. All models are compared to each other in terms of accuracy on a test set and performance on a player de-anonimization task. The models are also tested visually, by looking at the different clusters of representations that these models have learned. After building the models, the representations they produce are analyzed. We do this by examining the distributions of specific components and determining by which factors the player representations can be distinguished. Furthermore, we look at what happens when certain components of the representations are altered. Finally, it is tested whether the learned representations can be used to find similar players to a target player and in the VAEP framework for valuing actions. We do the latter by adding the player representations as additional features and look how that affects the evaluation metric scores of VAEP.

1.3 Overview

This thesis is structured as follows. Chapter 2 gives some necessary background information about techniques that will be used later, as well as the working of VAEs. Chapter 3 gives a small overview of previous work on player representations, including the work that was used as a large inspiration for this thesis. The VAEP framework is also briefly introduced. Chapter 4 gives an explanation and implementation of the three VAE models that were built, along with visualizations of the representations that they learn and comparisons of their performance. Chapter 5 tests different configurations of the final model, and analyzes the obtained player representations in depth. In Chapter 6, we test the applicability of the representations in the VAEP framework and to find similar players. As a conclusion to this text, Chapter 7 summarizes the work and results again and discusses possibilities for future work.

Chapter 2

Background

This chapter gives some necessary background information on the core techniques that are used throughout this dissertation. First, the concept of dimensionality reduction and one of the most used methods to achieve this is discussed. Second, a small introduction to neural networks is given, followed by an explanation of a specific type of neural network, the autoencoder. Finally, the variational autoencoder is explained, which is a variation on the classic autoencoder and is the core technology that the final architecture of this thesis is based upon. The majority of this chapter is based on the blog post by Joseph Rocca [5].

2.1 Dimensionality reduction

Dimensionality reduction is a concept that is essential to understanding autoencoders and VAEs, which form the basis of the models that learn our player representations in Chapter 4 (where a form of dimensionality reduction is applied on high-dimensional input data). It is a technique to reduce the dimensions (or the number of features) of data used as input to machine learning models. The data is then described with fewer features than before, leading to information loss during the data compression. The purpose of dimensionality reduction is to learn important information about the data, while keeping the loss of information minimal. It can also be useful in situations where low dimensional data is required, e.g., visualizing data in the 2D or 3D space.

A general model where most dimensionality reduction techniques build upon is the encoder-decoder model. We can view the encoder as something that takes in the original (high-dimensional) data, and produces the compressed (low-dimensional) data. The new space that this data lives in is called the encoded space or the latent space. The decoder does the reverse operation, it takes the compressed data as input and produces output of the same dimensions as the data that was put into the encoder. An illustration of this general model can be seen in Figure 2.1. The goal of such models is to make the encoder keep as much information as possible when compressing the input, so that the decoder produces output that is maximally similar to the original input data. In other words, we want to minimize the reconstruction error between the encoder input and the decoder output. As a result of this, the encoded data will have to contain only relevant information and discard superfluous information. Dimensionality reduction can thus be seen as a way to focus on the most important features of your data.

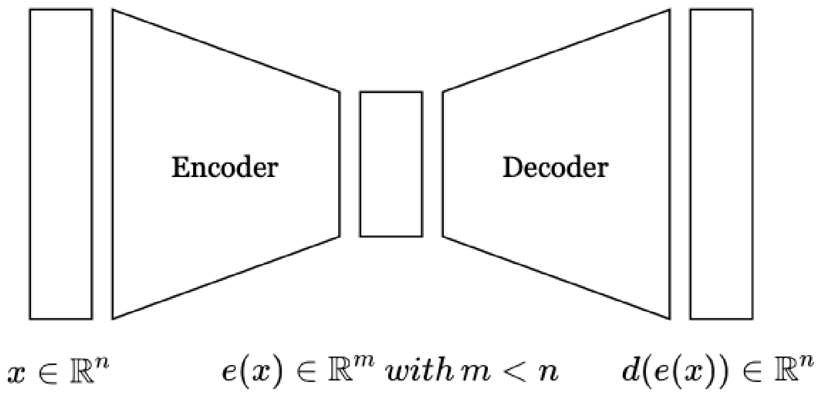

Encoder
Decoder
$x \in \mathbb{R}^n$
$e(x) \in \mathbb{R}^m \; with \, m < n$
$d(e(x)) \in \mathbb{R}^n$

FIGURE 2.1: Illustration of the general encoder-decoder framework

t-SNE One particular technique for dimensionality reduction is t-distributed Stochastic Neighbor Embedding (t-SNE) [6]. It is primarily used as a way to visualize high-dimensional data, which is where we will use it for when we visualize our player representations. Unlike another popular dimensionality reduction technique Principal Components Analysis (PCA), t-SNE is non-linear, so it can separate data which can not be separated by straight lines. t-SNE is capable of preserving the local structure of the data, i.e. it preserves small pairwise distances or local similarities, while also providing some insight in global structures (thus, preserving larger distances) like groupings in clusters. Because PCA is only good at preserving global structure, we use t-SNE for our visualizations later.

2.2 Neural Networks and RNNs

Neural networks or artificial neural networks (ANNs) are a set of machine learning models [7]. They consist of nodes (also called neurons) which are connected to each other in various possible ways. The connections make up layers of nodes and data is sent through these connections. The data is typically put into the network in one or more input nodes, and it is later sent to the deeper layers before arriving at the output layer. With each connection in the network, there is an associated weight and bias. These are used to transform the data when it goes through the network, by applying linear or non-linear operations on it. The main goal of a neural network is to learn from the data it is given and give back useful outputs at the output layer. The system can "learn" what to output by adjusting the weights and biases of the connections, based on the error between the actual output of the network and the expected output (the true output in supervised learning). A commonly used algorithm for adjusting the weights during learning is gradient descent with backpropagation, where each time an error is made with an output, the error is sent backward and the weights and biases are updated.

Networks where the connections between nodes do not form cycles are called feed forward neural networks. A special kind of neural network where cycles do occur and which is used later in this thesis is a Recurrent Neural Network (RNN). RNNs take sequential data as input, and they "memorize" information of prior inputs by compressing it into a hidden state vector. These kinds of networks are useful when data points are dependent on previous data points in the sequence, as the dependencies are explicitly modelled in the architecture of RNNs.

2.3 Autoencoder

An autoencoder is a type of neural network consisting of two components: an encoder and a decoder. Autoencoders can be seen as the general encoder-decoder model, explained in Section 2.1, where the encoder and decoder are neural networks. As previously stated, the goal of these kinds of models is to minimize the reconstruction error between the encoder input and the decoder output, while simultaneously learning latent space representations (compressed representations) of the data. In the case of autoencoders, this is done by learning the parameters of both components, i.e. its weights and biases, using gradient descent. Each time the model is given some data, the reconstruction error between input and decoder output can be calculated, and this error is backpropagated through the network, resulting in an optimal update of these parameters. Figure 2.2 shows an illustration of a possible autoencoder architecture.

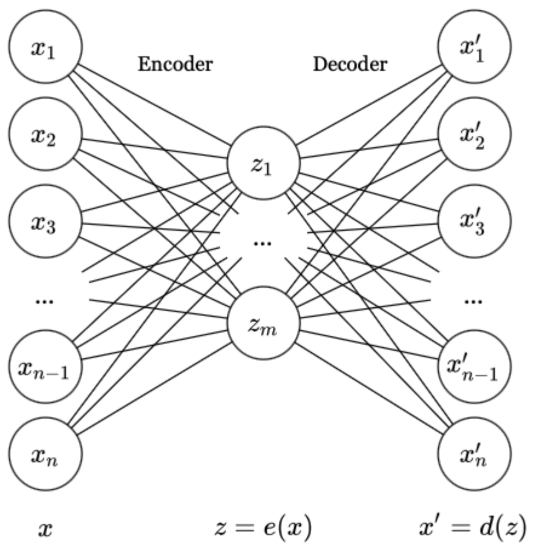


FIGURE 2.2: Illustration of an autoencoder

The architectures of the encoder and decoder neural networks can vary from simple feed forward neural nets with a limited number of layers to deep neural nets with a large number of (nonlinear) layers. With complex architectures, one could theoretically do a dimensionality reduction from any dimension to a dimension of one, by encoding each datapoint from a total of N datapoints as integers 1 to N. The decoder could then do the reverse process, and the reconstruction error would be zero. However, just reproducing the output as good as possible is not why autoencoders are used. They are used to extract meaningful representations from the data that can be used in other tasks. For the latent representations to be meaningful and interpretable, the latent space dimension and network architecture should be chosen well. To prevent autoencoders from learning the identity function and overfitting on the training data, variations on the basic model were introduced. One way to achieve this is to change the objective function. Sparse autoencoders (SAEs) [8] add a sparsity penalty (forcing some hidden units to become inactive) to the original loss function and contractive autoencoders (CAEs) [9] add a regularization term to make the model more robust to slight variations of inputs. Other variations include denoising autoencoders (DAEs) [10], where the input is partially changed by adding noise to it and recovering the cleaned or denoised data, and variational autoencoders (VAEs) [11], which is the focus of the following section.

2.4 Variational Autoencoder

Autoencoders are frequently used as generative models. Once an autoencoder is trained, one could randomly take a point from the latent space, put this into the decoder, and receive new output that was not in the original training data. Autoencoders could thus generate new data that is similar to the training data. For this to work, the latent space should be regular enough. It became clear in the previous section that this is not self-evident. Take the autoencoder which mapped each data point to an integer from 1 to N again. Taking some points from this latent space (points that are not in $[1..N]$) will result in meaningless outputs at the decoder. An architecture that tries to solve this issue and is often used for generative purposes is the Variational Autoencoder (VAE) [11]. The regularization that is present in VAEs is useful for learning player representations because it prevents overfitting to players with many observations, and thus generalizes to more players with fewer observations. VAEs have been used for many different applications, such as forecasting action trajectories [12], image (re)synthesis [13] and chemical design [14].

2.4.1 Architecture

In terms of architecture, a VAE looks similar to a standard autoencoder. The big difference lies in how the latent space is defined. The latent representations in standard encoders are fixed vectors. With variational autoencoders, the latent space is now a mixture of distributions. When training a VAE, the input is first encoded as a distribution over the latent space. Then, a point (often represented by the letter z) is sampled from this distribution and is put into the decoder. Finally, like in standard autoencoders, the reconstruction error is computed and is backpropagated through the network. Figure 2.3 shows a schematic illustrating this process for a VAE and standard autoencoder.

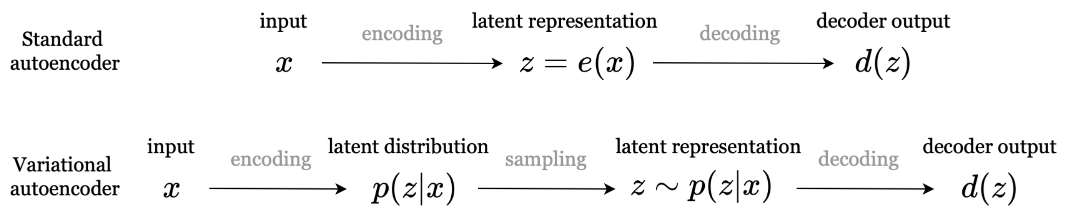


FIGURE 2.3: Illustration of the difference between a standard autoencoder (Top) and a variational autoencoder (Bottom)

The fact that the latent representations are no longer fixed vectors but distributions, allows for regularization of the latent space. These distributions are forced to lie close to another distribution called the prior distribution $p_\theta(\mathbf{z})$ with parameters θ, which is the distribution of the latent space, by adding a regularization term to the loss function. How this loss function is derived, is explained in the next section.

2.4.2 Mathematical formulation

The input data $\mathbf{x}$ can be characterized by a distribution $p_\theta(\mathbf{x})$ with parameters θ. The assumption is that the data is generated from the latent variables $\mathbf{z}$. This happens in two steps: $\mathbf{z}$ is first sampled from the prior distribution $p_\theta(\mathbf{z})$, after which

$\mathbf{x}$ is sampled from the conditional likelihood distribution $p_\theta(\mathbf{x}|\mathbf{z})$. Figure 2.4 shows this process as a probabilistic graphical model [1].

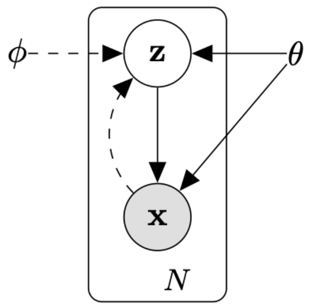


FIGURE 2.4: Illustration of a probabilistic graphical model describing the generative process, showing the dependencies between input data $\mathbf{x}$, latent variables $\mathbf{z}$ and distribution parameters θ and ϕ. From *Auto-Encoding Variational Bayes* [11]

The encoder and decoder components can now be described as follows. Instead of being deterministic, the decoder is now a probabilistic decoder, defined by the distribution $p_\theta(\mathbf{x}|\mathbf{z})$. The probabilistic encoder is similarly defined by the distribution $p_\theta(\mathbf{z}|\mathbf{x})$. Using the same notations as in probability theory, $p_\theta(\mathbf{z})$ is called the prior, $p_\theta(\mathbf{x}|\mathbf{z})$ the likelihood and $p_\theta(\mathbf{z}|\mathbf{x})$ the posterior. The relation between these distributions comes from Bayes' theorem:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}$$

In practice, computing $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ is an expensive operation and most of the time intractable. The true posterior will thus have to be approximated by an approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$, with ϕ being the parameters of that distribution. It is thus $q_\phi(\mathbf{z}|\mathbf{x})$ that is actually modeled by the encoder. One of the goals now is to have $q_\phi(\mathbf{z}|\mathbf{x})$ as close as possible to $p_\theta(\mathbf{z}|\mathbf{x})$, which is done by minimizing the Kullback-Leibler divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ between the two distributions. This term is defined as follows [15]:

[1] https://en.wikipedia.org/wiki/Graphical_model

$$\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \left(log(p_\theta(\mathbf{x})) + log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= log(p_\theta(\mathbf{x})) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \\
&= log(p_\theta(\mathbf{x})) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \\
&= log(p_\theta(\mathbf{x})) + E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left(log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - log(p_\theta(\mathbf{x}|\mathbf{z})) \right) \\
&= log(p_\theta(\mathbf{x})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) - E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}(log(p_\theta(\mathbf{x}|\mathbf{z})))
\end{aligned}$$

The equation can now be rewritten as:

$$log(p_\theta(\mathbf{x})) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}(log(p_\theta(\mathbf{x}|\mathbf{z}))) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}))$$

During training of a VAE, the goal is to maximize the left-hand side of this equation. Because the KL divergence is always non-negative, the following can be written:

$$log(p_\theta(\mathbf{x})) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) \leq log(p_\theta(\mathbf{x}))$$

The objective function that is obtained from this is called the evidence lower bound (ELBO):

$$-L_{\theta,\phi} = E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}(log(p_\theta(\mathbf{x}|\mathbf{z}))) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) \leq log(p_\theta(\mathbf{x})) \tag{2.1}$$

The ELBO objective is thus a sum of two terms. The first term indicates how well $\mathbf{x}$ is reconstructed from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Maximizing this term thus corresponds to minimizing the reconstruction error between input and output of the network. The second term is a distance measure between the approximate posterior and the prior. When minimizing this term, the posterior distribution will be forced to lie close to the prior, which is the regularization we discussed previously. The optimal parameters θ^*, ϕ^* are the ones that maximize the evidence lower bound, or equivalently minimize $L_{\theta,\phi}$. In the traditional VAE design, the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are the outputs of the encoder neural network.

Reparametrization trick Sampling the latent variables $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$ after the encoding is a non-differentiable operation, which is an issue when we want to use backpropagation and make gradient descent possible on this architecture. The so-called reparametrization trick [11] solves this issue. This simple trick modifies the equation $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ into $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \varepsilon$ with $\odot$ the element-wise product and ε a random vector sampled from the standard normal distribution: $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$. Training is now possible because ε isn't involved in the backpropagation process.

Chapter 3

Related work

In this chapter, previous work about representing soccer players and teams is explored. Two approaches in particular, that also learn soccer player representations from event stream data, are explained in more detail. The major differences with our approach are briefly discussed as well. Then, an approach for learning contextualized player representations in ice hockey is discussed, which was the main influence for the models described in this work. Finally, the VAEP framework for valuing on-the-ball actions in soccer is explained, as we will aim to improve its performance using our learned player representations in Chapter 6.

3.1 Characterizing playing style

Various approaches exist that try to characterize playing style of sports players by learning player representations that "describe" these players in a fixed size vector. Sections 3.1.1 and 3.1.2 explain two action-based approaches for soccer player representations, one of which is used as a baseline for comparison against our model's performance on a player de-anonimization task. Section 3.1.3 shortly describes an approach for learning ice hockey player representations that relies heavily on the match context.

3.1.1 Player Vectors

Decroos et al. [1] introduced Player Vectors, an approach for characterizing the playing style of soccer players based on event stream data. Their goal is to learn a fixed size player vector for a player based on the event stream data describing the actions of that player. Playing style is defined here as a concept that is characterized by a player's preferred area(s) on the pitch and the actions he or she tends to perform in these areas.

To achieve this, the relevant action types for this task are selected first. Decroos et al. argue that the only relevant actions for this task are offensive on-the-ball actions that come from open play. Only offensive actions are chosen because defensive playing style is mainly characterized by the positioning of players rather than which defensive actions they perform with the ball. The actions have to be on-the-ball, because that is the only data available when working with event stream data. Defensive on-the-ball actions like tackles and interceptions are omitted, as they are usually carried out out of necessity rather than being a playing style characteristic of some player. Actions that are also omitted are set pieces like free kicks or throw ins, because they are viewed as actions that are performed by specialists (e.g. free kicks) or by players on a certain position (e.g. throw ins are usually done by a fullback or winger). The actions that remain then are passes, dribbles, crosses and shots. In this thesis, no selection of relevant action types is done, as we believe that every action can be relevant in constructing player representations.

The following step consists of dividing the pitch into zones and counting how many times a certain relevant action was performed by a certain player. This results in a fixed sized matrix per player per action type, which is reshaped into a vector and then grouped together with vectors of the same action type to form a big matrix per action type. Then, non-negative matrix factorization (NMF) is performed on these matrices to reduce their dimensionality. The final player vector is then constructed by concatenating the players' compressed vectors of each action type. The quality of the system was assessed by predicting the identity of players based on anonymized event stream data. The results showed that the system was able to make very good predictions for offensive players and less good for defensive players.

In the NMF step, a parameter k_t needs to be chosen for each action type t. This parameter refers to the number of principal components of that action type. These principal components are human-interpretable, as they can be seen as variations of a certain action type (e.g. close- and far shot for the shot action). The obtained player vectors can thus be interpreted in a way that is useful for certain tasks like scouting and monitoring player development. This differs from the approach that is taken in this thesis, as the obtained player representations are less intuitive. However, this is not one of the goals we want to achieve, as we mainly want to use the representations in different tasks that do not require the representations to be easily interpretable.

A downside of Player Vectors is that the context of the actions that are performed is not fully taken into account. It is primarily based on the type of action and the zone on the field where the action took place. Other information like the result of the action, the time that was played, etc. could also be useful in characterizing a player's playing style.

3.1.2 SoccerMix

SoccerMix [2] is another approach for capturing playing style of both soccer players and teams. Grid-based approaches like Player Vectors [1], where a grid is placed over the field to divide the pitch in zones and where actions are counted in these grid cells, have some downsides. The grid cell boundaries are user-defined, so they are somewhat arbitrary, and this can also result in having actions that are spatially close but are deemed dissimilar by the model because they appear in different cells. Most approaches include only a few (one or two) attributes to group the actions, because splitting on more attributes would increase the sparsity of the data. To solve this problem, Decroos et al. use mixture models to better group actions together. They also include the direction in which the ball moves for each action, which is a feature that is also considered in the work of this thesis.

Their approach is as follows. For each action type, a mixture model is fitted to the locations of actions of that type. Then, a new mixture model is fitted to the directions of the action for each of the previously obtained mixture models. Each group in the mixture models corresponds to a prototypical action of a certain type, location and direction. Each action can then be seen as a probability distribution over these prototypical actions. This is then encoded in a weight vector, and the style vectors are obtained by summing the weight vectors of all actions that are performed by a player in some time span. These vectors are also interpretable, as the weight of an action group can be seen as an indication of how often that sort of action is performed by a player. In the same experiment for de-anonymizing players as conducted in the Player Vectors paper [1], SoccerMix achieved better results on almost all metrics.

Once again, match context in which the actions are performed is not taken into account in this approach. Similar to Player Vectors, an interpretable vector is obtained that can be used for e.g. comparing playing style of players, which is different from the player representations that are learned by the models we introduce later. The shortcomings that they try to address here that occurred in grid-based approaches do not appear in neural network based approaches liked the one introduced in this thesis.

3.1.3 Learning Agent Representations for Ice Hockey

Liu et al. [3] introduce a Variational Recurrent Ladder Agent Encoder (VaRLAE), which learns contextualized player representations of ice hockey players conditioned on the game history. VaRLAE combines variational autoencoders, RNNs and hierarchical latent variables instead of a single layer of latent variables, resembling ladder networks [16], in its architecture. The latent variables are hierarchically structured as in a ladder network, and their dependence follows a Markov Game Model [17]. This paper is the greatest influence on the work conducted in this thesis, as the model they introduce is largely the same as the final model that is explained in Section 4.4 and is used in later experiments. The biggest differences lie in the data that is used (different features) and the tasks for which they used their player representations. A more detailed explanation of the models follows in Chapter 4.

3.2 Valuing actions using VAEP

An important task in soccer analytics is measuring the quality of actions that soccer players perform by assessing which impact they have on the game. Knowing how valuable the actions of a player are is useful in player scouting and player development, because they are an indication of the player's quality or progress. Valuing players is traditionally done by focusing on specific actions like shots or tackles, and the match context in which the actions occur is often overlooked. One approach for valuing player actions that addresses the shortcomings of traditional player performance metrics is VAEP (Valuing Actions by Estimating Probabilities). The player representations that are learned by the models introduced in this thesis are later used as an extra input feature in the VAEP framework, in an attempt to improve its performance.

VAEP VAEP was introduced by Decroos et al. [4] as a framework for giving ratings to players and giving value to the actions those players perform. VAEP can give value to all types of actions (offensive and defensive) where the match context as well as the possible long-term effects are taken into account.

A value of an action can be seen as a measure on how much the action is expected to influence the scoreline. Actions will get a positive value when it is expected to increase the chance of scoring a goal for the team performing the action, and a negative value when the chances of the opponent for scoring a goal are increased. In their framework, a soccer game is presented as a series of actions $[a_1, ..., a_n]$ where each action a_i changes the game state from state S_{i-1} to S_i, where the state S_i can be represented by the actions that were performed up to that point in time ($[a_1, ..., a_i]$).

To give value to an action, the change in probability of scoring and conceding needs to be assessed. Let $P_{scores}(S_i, x)$ be the probability that team x (home or away) scores a goal in the near feature, while being in state S_i. The change in probability for scoring a goal when performing action a_i is then $\Delta P_{scores}(a_i, x) = P_{scores}(S_i, x) - P_{scores}(S_{i-1}, x)$. The authors call this value the offensive value of action a_i for team x. The probability of conceding a goal is similarly represented as $P_{conceded}(S_i, x)$, and the change in probability when performing action a_i is thus $\Delta P_{concedes}(a_i, x) = P_{concedes}(S_i, x) - P_{concedes}(S_{i-1}, x)$. The offensive value is positive if the chances of scoring increase. A team should always strive to reduce the likelihood of an opposing goal, so the defensive value is defined as $-\Delta P_{concedes}(a_i, x)$. These values are combined into the total VAEP value of an action: $V(a_i, x) = \Delta P_{scores}(a_i, x) + (-\Delta P_{concedes}(a_i, x))$.

The scoring and conceding probabilities needed to obtain the VAEP values are calculated by looking k steps ahead (where k is a user-defined parameter), and computing the probability that a goal is scored in one of these following actions. To compute these probabilities, binary probabilistic classification was performed with as input a number of features that describe the game state and as output a binary label indicating whether a goal was scored in the subsequent k actions. For the input, only the three most recent actions are considered and a number of features of these actions are used. The features consist of simple features (like location, action type, time played, etc.), complex features which are calculated using the simple features of one action and of consecutive actions, and game context features like the goal difference after action a_i.

The player rating of a player in one game (90 minutes) is calculated as the sum of the VAEP values of actions that a player performed, scaled down by the actual minutes played by that player. Intuitively, these player ratings can be seen as the average net goal difference of the player's team that a player contributed to in a game.

Because no ground-truth for action values exist, the system was evaluated by looking at the underlying scoring probabilities, for which ground-truth labels do exist. In their experiments, Decroos et al. evaluate the system using different classifiers and different feature sets on the Brier-score metric, which measures the calibration of the obtained predictions. They concluded that using all VAEP features gave the best scores, followed by the top 10 most important features of these VAEP features, with only a small difference in the Brier score. Using only action locations as features gave the worst scores. In Section 6.2, we will investigate whether the player representations obtained in this thesis could give an additional performance boost when including these in the feature set.

Chapter 4

Methodology

In this chapter, we start by describing the data that is used for training the different models. Then, the approach for building the VAE architectures is explained along with some theoretical background of these models. The training process and details about the implementation of each model is given as well. Additionally, each model is evaluated on its accuracy and performance on a player de-anonimization task. The first model is a simple extension of the standard VAE, and the later models build further on this with added complexity.

4.1 Data

4.1.1 Data format

One can think of many types of information when describing soccer matches. The most widely used type of soccer data is match sheet data, which is the usual kind of high-level information that the media reports on soccer games like line-ups, goals and substitutions. Currently, many soccer clubs have contracts with companies who collect data that is more extensive and detailed than simple match sheet data. One type of data that is collected is tracking data, which captures the exact position of the ball and of all the players during the whole game. Another type of data that is sold by organizations like StatsBomb [18], Stats Perform [19] and Wyscout [20], is called event stream data. It is obtained by human annotators who record each on-the-ball action in a soccer game. The data consists of information about each action that is performed with the ball, including the type of action, the action's start- and end location, the timing of the action and some additional features.

For the purpose of learning player representations, event stream data is used. More specifically, the data that is used in training the models and for experiments is that of two different competitions: the 2019/2020 La Liga season which includes all games of FC Barcelona and the 2019/2020 and 2020/2021 seasons of the FA Women's Super League (FAWSL). The La Liga data was used to quickly train and test different architectures because it is the smallest dataset with only 33 games. The 2019/2020

FAWSL data contains a lot more games from all teams in that competition (87 games in total) but doesn't cover a whole season because it was stopped early due to the COVID-19 pandemic. The 2020/2021 FAWSL dataset is the most complete dataset, with 131 games. This data is all publicly available and provided by StatsBomb [1].

The raw StatsBomb data contains very detailed information about each event occurring in the game. Events include everything that happens in a game: actions that players perform, as well as general events such as the end of the game or substitutions. For our learning task, only the actions are needed. Events can also contain optional information, causing the data to differ in size and structure, which makes it harder to automatically parse it. For this and other reasons, Decroos et al. [4] proposed SPADL (Soccer Player Action Description Language). SPADL transforms event stream data from different vendors into one format that is human-interpretable and simple. The data in the SPADL format contains only on-the-ball actions, where each action is described by a number of features. We choose the following ten from the total set of features:

- **player_id**: ID of the player who performed the action
- **period_id**: ID of the period of the game in which the action happened (first or second half, or first or second period of extra time if applicable)
- **time_seconds**: Start time of the action in seconds counted from the beginning of the period
- **start_x, start_y, end_x, end_y**: Start- and end coordinates of the action
- **type_id**: Type of action (e.g. pass, dribble...)
- **result_id**: ID of the result of the action (success, failure or others like yellow card)
- **bodypart_id**: ID of the body part used to perform the action

Features like **game_id** were not used because it is irrelevant to know in which game the action was performed for learning player representations. The **team_id** feature was removed as well because the network might give this feature a very high weight while predicting the acting player and this will not contribute to learning useful player representations.

From this basic set of features, an additional nine features were derived and added to the input data because they can be useful for characterizing playing style. The added features include start- and end distance to goal, start- and end- angle to goal, the difference in x- and y coordinates and the total distance covered by the action, which can give an indication whether the player in question plays a lot towards

[1] https://github.com/statsbomb/open-data

the opponent's goal or plays more defensively. The absolute timing of the action is included too (the **time_seconds** feature is relative to the period), as well as the goal-score difference, which is the number of goals that the team of the player on the ball is behind or ahead. This feature may provide useful information, as some players might behave differently than other players when their team is winning or losing.

4.1.2 Data preprocessing

Before the data is ready to be used as input for the different models, it needs to be preprocessed. First of all, some of the integer features (**player_id**, **type_id**, **result_id** and **bodypart_id**) are not really ordered numerical values but serve as categorical features. A neural network might interpret these features as being ordered when putting them into the network like this, so they must be encoded into vectors. All the mentioned categorical features are encoded as one-hot vectors with as size the number of different values possible for that feature. All other features are numerical and are standardized (i.e. the mean of the values in the data is zero and the variance is one). Standardization is widely done in gradient based algorithms when the input data attributes have different scales.

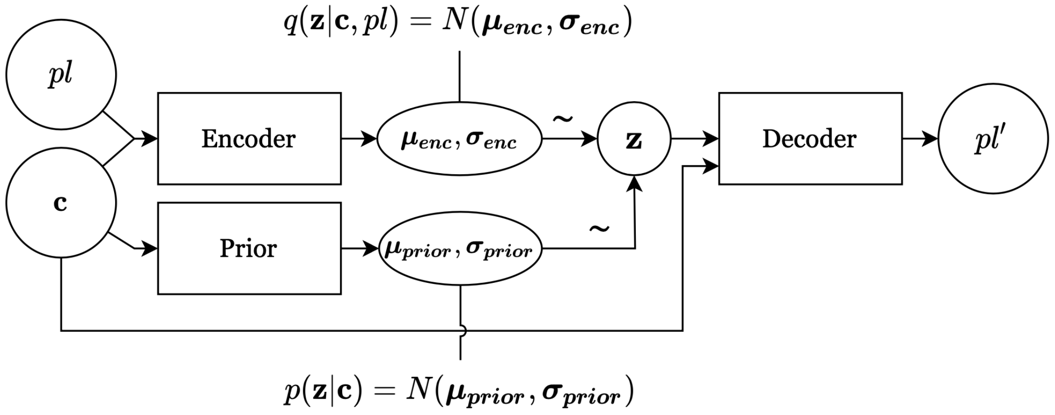
To use the data in the recurrent models, it needs to be transformed from single actions to sequences of actions. How this is done will be explained later in Section 4.3.

4.2 Model 1: CVAE

The first model that was built for learning the player representations is the Conditional Variational Autoencoder (CVAE) [21]. A CVAE is a variant of a variational autoencoder which uses an additional "condition" as input to the encoder and decoder. In this case, the condition will be the match context in which a player performs an action. The distributions that the network now learns (the prior, posterior and likelihood distributions as explained in Section 2.4.2) all condition on this extra input, and the latent variables and outputs that the decoder generates now also depend on this extra condition.

4.2.1 Architecture

To see how this fits into our player representation framework, we will map the data from Section 4.1 to the variables of the variational autoencoder explained in Section 2.4. A schematic of the CVAE architecture can be seen in Figure 4.1. The input data x are the player IDs, which are one-hot vectors representing each player pl from the used dataset. We want the player representations to be dependent on the match context, because a player's style can differ depending on the situational context the player finds his-/herself in, and it is very difficult to describe a player's style under every possible match context. In this case, the match context on a given moment contains all features described earlier except the player IDs pl. All these features are concatenated into a single "context" vector $\mathbf{c}$.



$q(\mathbf{z}|\mathbf{c}, pl) = N(\boldsymbol{\mu_{enc}}, \boldsymbol{\sigma_{enc}})$
pl
$\mathbf{c}$
Encoder
Prior
$\boldsymbol{\mu_{enc}}, \boldsymbol{\sigma_{enc}}$
$\boldsymbol{\mu_{prior}}, \boldsymbol{\sigma_{prior}}$
$\sim$
$\sim$
$\mathbf{z}$
Decoder
pl'
$p(\mathbf{z}|\mathbf{c}) = N(\boldsymbol{\mu_{prior}}, \boldsymbol{\sigma_{prior}})$

FIGURE 4.1: Schematic of the CVAE architecture

During training, the approximate posterior $q_\phi(\mathbf{z}|\ pl, \mathbf{c})$, which now conditions on the match context $\mathbf{c}$, is learned. The parameters $\boldsymbol{\mu_{enc}}$ and $\boldsymbol{\sigma_{enc}}$ of this Gaussian distribution $\mathcal{N}(\boldsymbol{\mu_{enc}}, \boldsymbol{\sigma_{enc}})$ are the outputs of the encoder, which takes the concatenation of the player ID pl and context variables $\mathbf{c}$ of an action as input. $q_\phi(\mathbf{z}|\ pl, \mathbf{c})$ is the contextualized player representation of player pl under match context $\mathbf{c}$, which is later used in experiments. Like before, the decoder takes latent variables $\mathbf{z}$ as input, which is sampled from the approximate posterior at training time: $\mathbf{z} \sim q_\phi(\mathbf{z}|\ pl, \mathbf{c})$. $\mathbf{z}$ is concatenated with $\mathbf{c}$ again before being put into the decoder. The decoder models the likelihood distribution $p_\theta(pl|\ \mathbf{z}, \mathbf{c})$, and its output is compared with the original player input pl. The equations from Section 2.4.2 are still valid when conditioning on $\mathbf{c}$, so the ELBO objective (Equation 2.1) becomes:

$$-L_{\theta,\phi} = E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\ pl,\mathbf{c})}(log(p_\theta(pl|\ \mathbf{z}, \mathbf{c}))) - D_{KL}(q_\phi(\mathbf{z}|\ pl, \mathbf{c})\,||\,p_\theta(\mathbf{z}|\mathbf{c})) \tag{4.1}$$

The first term measures how well the player ID input is reconstructed, and the second term measures the closeness of the approximate posterior and the context-specific prior $p_\theta(\mathbf{z}|\mathbf{c})$, which is again a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu_{prior}}, \boldsymbol{\sigma_{prior}})$. At test time, sampling is done from the prior distribution and the sampled latent variables are concatenated with the context variables to feed into the decoder. The decoder has to predict the acting player solely on the latent variables and context variables, thus without the player ID.

As explained before, the Kullback–Leibler divergence term acts as a regularization term. It creates a shrinkage effect between the individual player representations $q_\phi(\mathbf{z}|\ pl, \mathbf{c})$ and the prior that is conditioned on the context $p_\theta(\mathbf{z}|\mathbf{c})$. This means that the posterior for each player is "shrunk" towards the prior mode (the peak of the probability mass function, or the value that is most likely to be sampled) [2]. This is useful for our player representations because of two reasons. First, it prevents overfitting to players with many observations. Because all representations are drawn towards the prior mode, the model generalizes better. Second, because prior distributions that are conditioned on similar match contexts lie closer together, player representations that condition on similar match contexts will also lie closer together, because they are drawn towards their priors. This has the effect that the player representations of different players who act similarly in similar match contexts, or of the same player in different match contexts, will lie closer together. This property is what is ultimately desired from our player representations, as players with comparable playing styles will often appear and act the same in similar match contexts.

4.2.2 Implementation

The layers of the CVAE along with its dimensions are visualized in Figure 4.2.

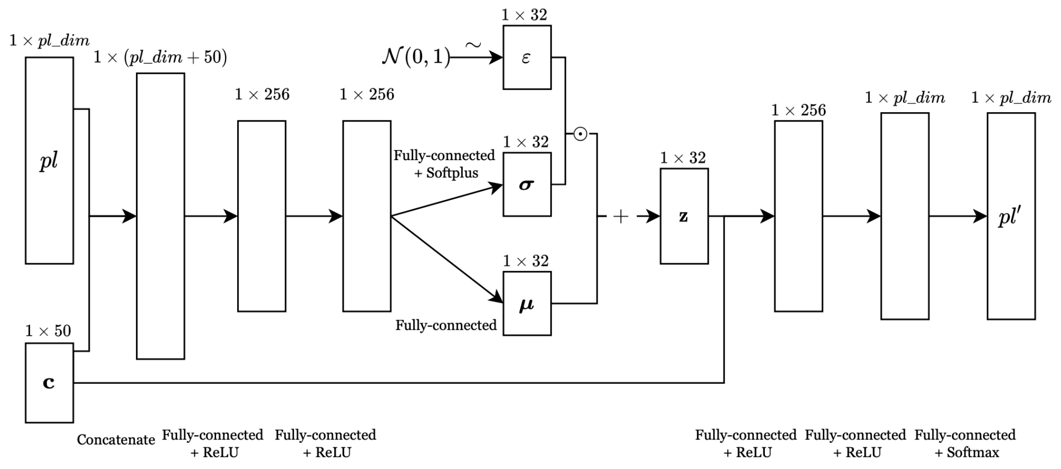


FIGURE 4.2: Layers and dimensions of the CVAE

Inference The input of the encoder is the concatenation of the player ID of the player who performed the action and the context variables of the action. The dimension of this input vector depends on the total number of players and thus on the dataset we are training on. It is of length 370 for the La Liga dataset, 237 for FAWSL 19/20 and 273 for FAWSL 20/21. The dimension of the context vector, i.e. the concatenation of all context variables, is the same in all cases (50). It is the sum of the numeric features (14) and the lengths of the period- (3), action type- (22), result- (7) and body part (4) one-hot vectors.

[2] https://en.wikipedia.org/wiki/Mode_(statistics)

The encoder consists of two fully-connected layers with a hidden dimension of 256, both using a ReLU activation function (with $ReLU(x) = max(0, x)$). The parameters $\boldsymbol{\mu_{enc}}$ and $\boldsymbol{\sigma_{enc}}$ of the approximate posterior distribution are computed by feeding the hidden variables to fully-connected layers with a linear and softplus activation function respectively. A linear activation function essentially means "no activation", so the weighted output is not changed to obtain the mean $\boldsymbol{\mu_{enc}}$. The softplus activation is a differentiable approximation to the ReLU function ($f(x) = log(1 + exp(x))$). It is used because it enforces the variance $\boldsymbol{\sigma_{enc}}^2$ to be positive. The dimension of both parameters is the same as the dimension of the latent space embeddings, and thus of the learned player representations. While experimenting, low dimensions worked better than high dimensional embeddings, so a dimension of 32 is used.

The network that computes the parameters of the prior distribution is exactly the same as the encoder network, except that it only takes the context variables as input. It is therefore not included in Figure 4.2.

Generation $\mathbf{z}$ is calculated by using the reparametrization trick (explained in Section 2.4.2). A random vector ε of length 32 (latent space dimension) is sampled from the standard normal distribution $\mathcal{N}(0, 1)$. During training, $\mathbf{z}$ is calculated by the formula $\mathbf{z} = \boldsymbol{\mu_{enc}} + \boldsymbol{\sigma_{enc}} \odot \varepsilon$. At test time, $\mathbf{z}$ is sampled from the prior distribution, so the formula becomes $\mathbf{z} = \boldsymbol{\mu_{prior}} + \boldsymbol{\sigma_{prior}} \odot \varepsilon$. The context variables $\mathbf{c}$ are concatenated with $\mathbf{z}$ before being fed into the decoder. The first two layers of the decoder are fully-connected layers with ReLU activation functions. The first one transforms the input of dimension 32 to a vector of dimension 256, and the second layer has an output dimension of pl_dim, the same dimension as the player input pl. The final layer is a fully-connected layer with a softmax activation function. The softmax function is defined as

$$\sigma(\mathbf{x})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for\ i = 1, ..., K\ and\ \mathbf{x} = (x_1, ..., x_K) \in \mathbb{R}^K$$

It transforms each component x_i to a value between 0 and 1 and makes the sum of the components add up to 1. The output is thus a vector pl' of dimension pl_dim, where each component pl'_i represents the probability of player i performing the action that was put into the network, where i is the index of the predicted player in the ordered list of player IDs.

Loss function The loss function that is used for training the network is derived from the objective function defined before (Equation 4.1). For the reconstruction term, a categorical cross entropy loss function is used. It is defined as

$$CE(y_i, \hat{y}_i) = -\sum_{i=1}^{K} y_i \cdot log\, \hat{y}_i$$

In our case y_i is equal to pl and $\hat{y}_i$ is pl'. Categorical cross entropy measures the average difference between the actual and predicted discrete probability distributions and is the default loss function to use in multi-class classification problems. Another option that is sometimes used for this kind of task is the Mean Squared Error (MSE). The regularization term is the KL-divergence between the approximate posterior distribution and the context-specific prior. For two Gaussian distributions, this term can be calculated by using the parameters of the distributions [3]:

$$D_{KL}(q_\phi(\mathbf{z}|\, pl, \mathbf{c}) \,||\, p_\theta(\mathbf{z}|\mathbf{c})) = log\frac{\boldsymbol{\sigma_{prior}}}{\boldsymbol{\sigma_{enc}}} + \frac{\boldsymbol{\sigma_{enc}}^2 + (\boldsymbol{\mu_{enc}} - \boldsymbol{\mu_{prior}})^2}{2\boldsymbol{\sigma_{prior}}^2} - \frac{1}{2}$$

The loss function we want to minimize it the opposite of the objective function:

$$L_{\theta,\phi} = CE(pl, pl') + D_{KL}(q_\phi(\mathbf{z}|\, pl, \mathbf{c}) \,||\, p_\theta(\mathbf{z}|\mathbf{c}))$$

At test time, we also measure the accuracy of the player predictions by means of the categorical accuracy metric. This corresponds to the frequency of correct predictions, where the predicted player is the player corresponding to the highest probability in the output vector of the decoder.

4.2.3 Model evaluation

To train the CVAE, we use the Adam optimizer. The Adam algorithm is an extension to stochastic gradient descent, which uses adaptive per-parameter learning rates. The base learning rate that is used in the experiments is 0.001. In the final model from Section 4.4, other learning rates will be experimented with. We use the three competitions mentioned earlier in Section 4.1 to train and test the network. The set of all actions is randomly shuffled and split in a train and test set containing 67% and 33% of the total dataset respectively. The model is trained for 50 epochs, and training is stopped when the loss on the validation set (which is 50% of the test set) has not improved for 5 epochs (early stopping).

Accuracy Table 4.1 below shows the performance in terms of accuracy of the CVAE for three datasets.

La Liga	FAWSL 2019/2020	FAWSL 2020/2021
0.203	0.082	0.073

Table 4.1: Accuracy of the CVAE for three different competitions

A possible explanation why the accuracy for the La Liga season is a lot higher than for the two FAWSL seasons is that the La Liga dataset only contains games of FC Barcelona, so it will be biased towards players from that team. The Barcelona players have many more datapoints than players of other teams, so their predictions will generally be better. The performance can be compared against a baseline, which

[3]https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians

chooses a random player as prediction for each action. The accuracy of the random guessing baseline is around $\frac{1}{pl\_dim}$ for each competition ($\frac{1}{370} \approx 0.0027$ for La Liga, $\frac{1}{237} \approx 0.0042$ for FAWSL 19/20 and $\frac{1}{273} \approx 0.0037$ for FAWSL 20/21), so the CVAE does a lot better. Another baseline is a model that predicts a Barcelona player each time for the La Liga dataset, which achieves an accuracy of 2.3%. For the two FAWSL datasets, we use a model that predicts a random player from Arsenal or Manchester City each time (the two teams with the most actions in both datasets), achieving an accuracy of 0.5%. Both baselines thus achieve accuracies that are a lot lower than the CVAE model. As a comparison, the CVAE model of Liu et al. [3] achieves an accuracy of 11.94%, which is close to what we accomplish.

Visualizations The accuracy metric only measures the predictive performance of the network, not the quality of the obtained player representations. A way to get an idea of the embeddings is to visualize them in 2D space and see how well they are separated. The player representations are 32-dimensional, so a dimensionality reduction technique is needed which maps them to two-dimensional vectors. One of the best methods for this task is t-distributed stochastic neighbor embedding (t-SNE), which was explained earlier in Section 2.1. t-SNE preserves the local structure of the data, so similar embeddings will be clustered together.

Intuitively, player representations of players that act in the same zone should be more alike. Because the start area of an action is no explicit attribute of the data, we divide the pitch in areas and derive the start area from the start location of the action. How the pitch is divided can be seen in Figure 4.3.

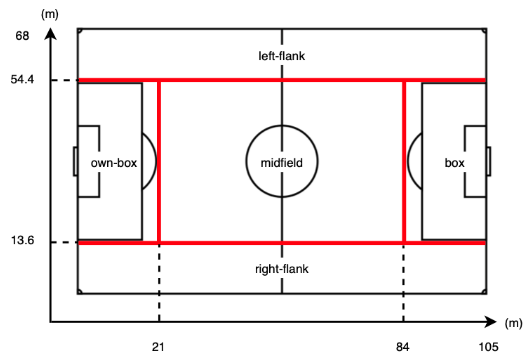
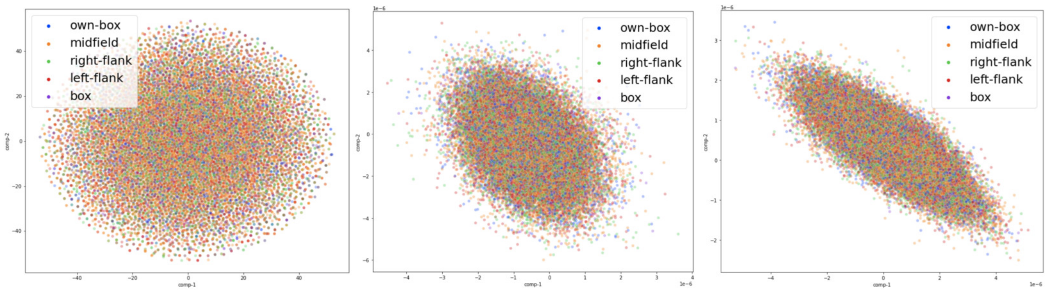


FIGURE 4.3: Starting areas of the pitch

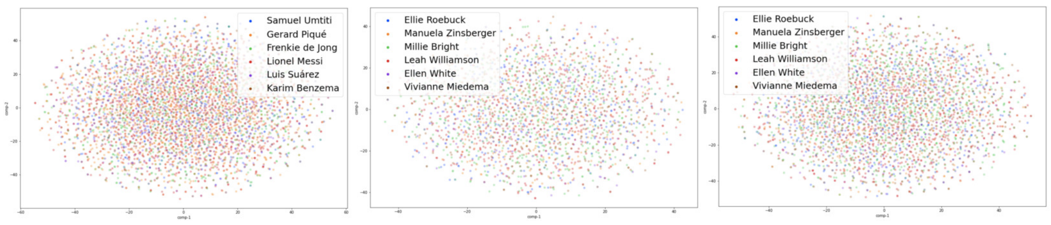
A choice that has to be made is which vector to choose as our player representation. Because the player representations are distributions conditioned on the match context, the vector can be a random sample or the mean vector of this distribution. We first experiment with random samples. Figure 4.4 shows the high-dimensional player representations of the acting player at each event of the test data for the three competitions, visualized in 2D space with t-SNE. The color of the points indicate the starting area of the action. The figure shows one big cloud of points for each competition, indicating that the CVAE can't properly separate all embeddings.



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

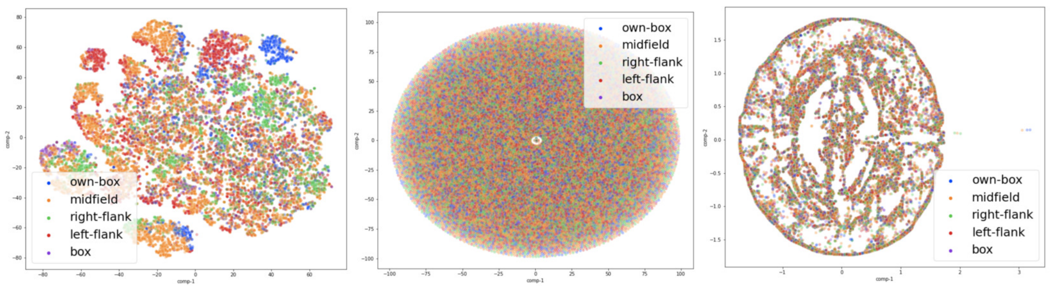
Figure 4.4: Visualization of all player representations labelled by start-area

To see how well individual players are separated, a small set of six players is chosen for each competition (the same players are chosen for FAWSL 19/20 and FAWSL 20/21), and the embeddings are visualized with t-SNE (Figure 4.5). Here too, all the points are scattered around the figure, with no separation possible. When we take the mean vector of the posterior distribution as our embeddings, the t-SNE projections become separable for the La Liga dataset (Figure 4.7). The points of the FAWSL 19/20 and 20/21 players look different than in Figure 4.5, but separation is still not possible. A possible cause for this can be that the variances of the components of the embeddings are very large relative to the means. This is less the case for the models discussed later. Figure 4.6 shows the embeddings of all players, again using the mean vectors. The points no longer form one big cloud (except for the FAWSL 19/20 case), but the clusters contain embeddings from all starting areas, so this figure is not insightful either.



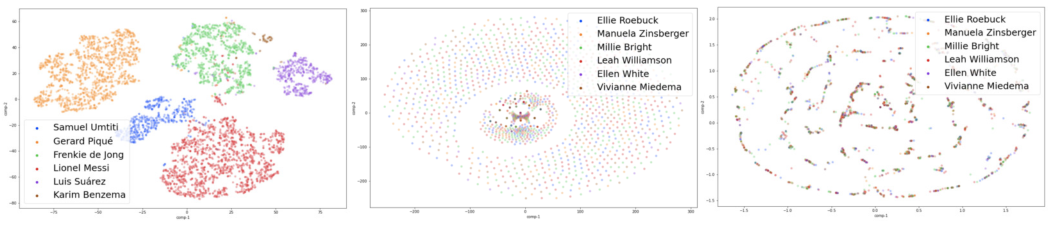
(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

Figure 4.5: Visualization of the player representations of six chosen players



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

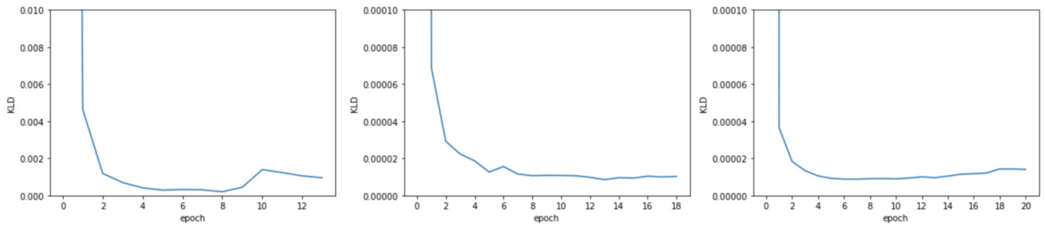
FIGURE 4.6: Visualization of all player representations labelled by start-area, where the player representation is the mean vector of the posterior distribution



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.7: Visualization of the player representations of six chosen players, where the player representation is the mean vector of the posterior distribution

Posterior collapse Many VAE architectures experience a problem called posterior collapse [22]. This problem appears when the weights of the latent space are very small, such that the decoder ignores the latent vectors and only uses the context variables to compute its outputs. It occurs when the KL-divergence term in the loss function drops to almost zero and the latent space becomes over-regularized, causing the latent variables to contain little or no information. When looking at the KL-divergence during training of the CVAE on the FAWSL 19/20 and 20/21 data, we see that it drops to a very small value in the first few epochs (around 10^{-5}) and it does not change much afterwards (Figure 4.8). On the La Liga data, the KLD converges to a much greater value (around 0.001).



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.8: KL-divergence during training of the CVAE on the La Liga (A), FAWSL 19/20 (B) and FAWSL 20/21 (C) data

Several solutions have been proposed to solve this issue, and most of them involve re-weighting the KL-divergence term in the loss function to lessen its importance ([22], [23], [24]). In addition to the ladder structure they use for their VaRLAE model (which is explained later in Section 4.4), Liu et al. [3] use a factor $\beta < 1$ in their loss function to scale the KL-divergence term. β is increased linearly from 0 to 1, so that the model goes from a standard (deterministic) autoencoder to a variational autoencoder. While Liu et al. only use this warm-up period in their final ladder model, we try it for all models.

With the warm-up factor, the loss function becomes:

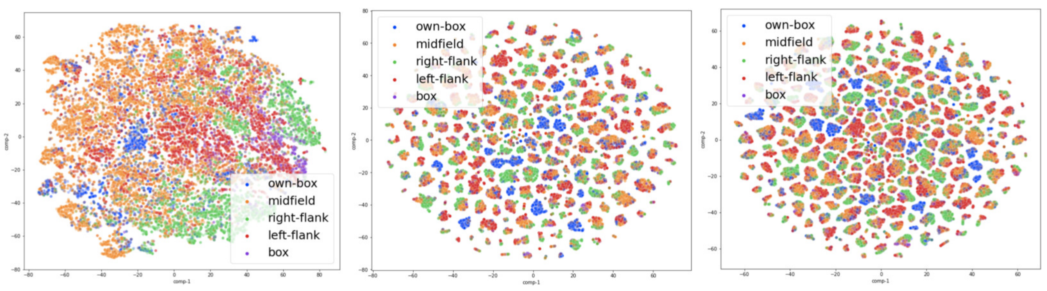
$$L_{\theta,\phi} = CE(pl, pl') + \beta D_{KL}(q_\phi(\mathbf{z}|\, pl, \mathbf{c}) \,||\, p_\theta(\mathbf{z}|\mathbf{c}))$$

Results using a warm-up period In the following experiments, β is increased linearly from 0 to 1 during the first five epochs (of the 50 in total). The accuracy of the CVAE is worse for each competition when applying the warm-up period (Table 4.2).

La Liga	FAWSL 2019/2020	FAWSL 2020/2021
0.167	0.024	0.019

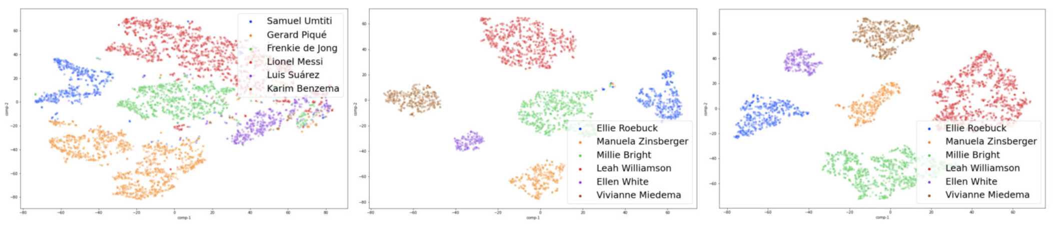
TABLE 4.2: Accuracy of the CVAE with a warm-up period for three different competitions

When a sample from the posterior distribution is chosen as the player representation, the t-SNE visualizations look similar to those of Figure 4.4 and 4.5. Using the mean, the visualizations of the player representations of all players show that the embeddings are better clustered together, close to other embeddings of the same start area (Figure 4.9). The case study of six players show different clusters for each player, especially for the two FAWSL seasons (Figure 4.10). This demonstrates that even though the accuracy is less for all competitions when using the warm-up period, the quality of the embeddings has improved.



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.9: Visualization of all player representations labelled by start-area, where the player representation is the mean vector of the posterior distribution and a warm-up period is used



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.10: Visualization of the player representations of six chosen players, where the player representation is the mean vector of the posterior distribution and a warm-up period is used

4.2.4 Player de-anonimization

As an additional way to test the quality of the player embeddings, we conduct an experiment (also done by Decroos et al. [1]) to test how well players can be identified based on their player representation. This is done for each of the models that follow, such that their performance can be compared.

Experiment setup We do this by first splitting the test set of each competition (33% of the total data) into two separate sets. The set which contains 67% of the testing data is used to compute a single player representation for each player in that set. Because different representations are learned for each action that a player performs, we get multiple representations per player. The mean of these representations per player is taken as the labelled player representation of that player. Then, the (anonymous) player representations are computed using the remaining actions of the test set (33% of the total testing data). Next, we compare each embedding that is obtained against the labelled player representations. A ranking is then constructed where the labelled embeddings are sorted by increasing Manhattan distance, i.e. the labelled player embedding with the smallest Manhattan distance to the given anonymous embedding appears at the top. Another similarity metric that was tried out is the cosine similarity, but it was left out in the experiments because the results were very similar. We can then count how many times the unknown player appears at a certain position in the ranking. It is of course desired that the unknown players appear many times at the top of their own rankings.

Five metrics are reported that reflect the quality of the player embeddings: the percentage of how many times a player is in the top-k of their own ranking (with $k \in \{1, 3, 5, 10\}$), and the mean reciprocal rank (MRR), which is defined as follows:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

$|Q|$ is the amount of rankings that are evaluated and $rank_i$ is the place of the correct player in ranking i. Higher MRR values mean that the players appear higher in their own rankings on average.

Results The results of this experiment for the CVAE model on the three datasets are shown in Table 4.3 below. A warm-up period is used because the results are better than without it. For each competition and on each metric, the scores are very low. This is due to the posterior collapse problem explained earlier, which renders the learned embeddings useless for downstream tasks.

	Top-1	Top-3	Top-5	Top-10	MRR
La Liga	0	0.003	0.005	0.010	0.014
FAWSL 19/20	0.002	0.005	0.012	0.031	0.022
FAWSL 20/21	0	0.004	0.009	0.026	0.019

TABLE 4.3: Top-k percentage and MRR of the CVAE model on the three datasets

4.3 Model 2: CVRNN

The next model that is discussed is the Conditional Variational Recurrent Neural Network (CVRNN). A CVRNN is an extension of a VRNN [25], adding context variables to it, like the CVAE extended the VAE. This model incorporates the play history by extending the CVAE into a recurrent framework.

4.3.1 Additional data preprocessing

Because RNN models expect sequential data as input, our data needs to be adapted to fit into the model. In the CVAE model, each input was a single action consisting of the player ID pl and the context variables $\mathbf{c}$. This has to be transformed to sequences of actions $[a_1, ..., a_t, ..., a_N]$, with a_t the action at timestep t in the sequence. To effectively incorporate play history, the actions need to logically follow one another, i.e. they can not be shuffled. A single sequence also can not contain actions from different games, because they did not occur consecutively in real life either. Our sequences consist of ten actions each, where a value of ten is chosen as a balance between the fact that actions ten or more steps ago can have an influence on the current action, and a sequence length that is not too large. To obtain the sequences, the data is first split up game per game, after which it is preprocessed like in Section 4.1.2. Then it is made into sequences by taking the first ten actions of the game and then shifting one action at a time to get a new sequence each time there is a shift.

4.3.2 Architecture

The CVRNN architecture can be seen as a CVAE embedded into an LSTM, which is a special kind of RNN that can maintain information for longer time periods. An LSTM can be "unrolled", so that it looks like a chain of cells (with the same length as the input sequences) which hold copies of the same network, a CVAE in our case. One such cell at timestep t of the CVRNN is shown in Figure 4.11.

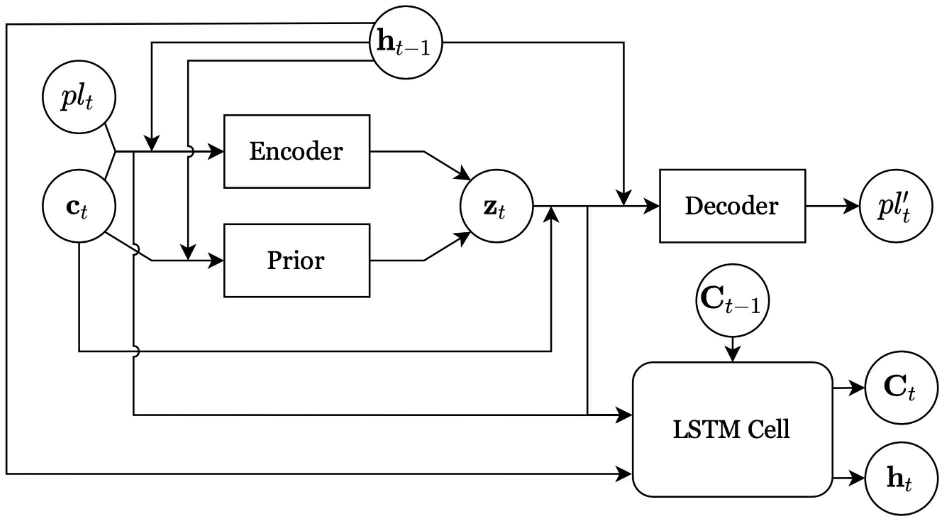


FIGURE 4.11: Schematic of one CVRNN cell

At each timestep t, player ID pl_t and context variables $\mathbf{c}_t$ are put into the network and the output is a player prediction pl'_t, just like in the CVAE case. The difference is that information from the beginning of the sequence is passed through the network to later timesteps via the hidden states $\mathbf{h}_t$ and cell states $\mathbf{C}_t$. These states are updated each time step by means of an LSTM cell. This cell returns a new cell- and hidden state, which are used again in the next timestep for the computation of the next player prediction, hidden state and cell state.

Using play history as an extra information source to predict the acting player should result in a performance boost of the model. Knowing which actions were performed before the current action and by whom can give lots of extra information. Although team information is left out, the model can learn which players play in the same team, because it is likely that players in the same sequence belong to the same team. If the ball is passed around nine times by players of the same team, it is very likely that the tenth action will be performed by another player of that team who operates in the area where the action took place.

4.3.3 Implementation

The implementation of the CVRNN is a bit different from that of the CVAE (Figure 4.12):

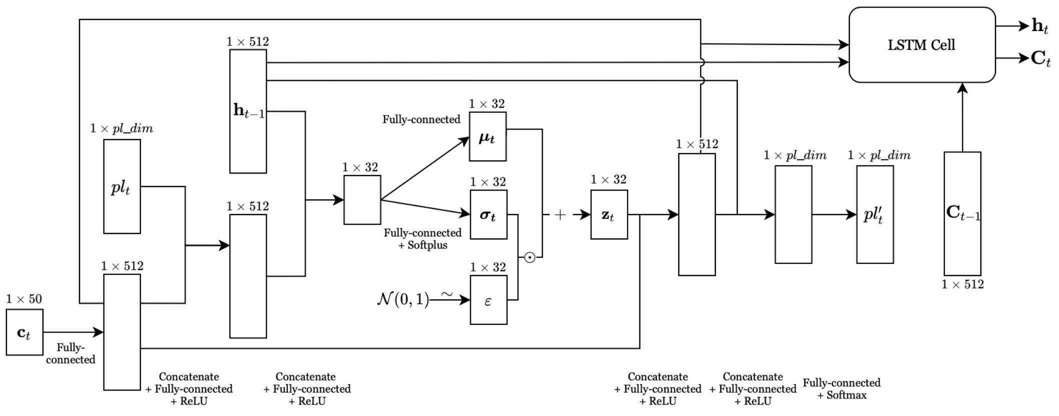


FIGURE 4.12: Layers and dimensions of one CVRNN cell

Inference In the encoder, the context variables $\mathbf{c}_t$ first go through a fully-connected layer with linear activation before being concatenated with the player IDs pl_t. This is followed by two fully-connected layers with ReLU activation, where in between the two layers the output of the first layer is concatenated with the hidden state of the previous time step $\mathbf{h}_{t-1}$. The parameters $\boldsymbol{\mu_{enc,t}}$ and $\boldsymbol{\sigma_{enc,t}}$ are computed as in the CVAE. From these parameters, the player representation of player pl_t given match context $\mathbf{c}_t$ $(q_\phi(\mathbf{z}_t|\, pl_t, \mathbf{c}_t))$ can be obtained. Like in the CVAE, the mean vector or a random sample from the distribution defined by the parameters can be chosen as the final player representation. Important to note is that only the representations of the last timestep are used in measuring the accuracy and for downstream tasks, because only this representation has the information from all other actions in the sequence (because the RNN is unidirectional). The prior network is again the same as the encoder network, except for one less fully-connected layer before the concatenation with the hidden state variable.

Generation The latent variables $\mathbf{z}_t$ are calculated as before by using the reparametrization trick. The decoder is very similar to the encoder except for the final layer: $\mathbf{c}_t$ first goes through a fully-connected layer with linear activation, after which it is concatenated with $\mathbf{z}_t$. Two fully-connected layers with ReLU activation follow, where the output of the first layer is concatenated with $\mathbf{h}_{t-1}$. The final layer is a fully-connected layer with softmax activation, and the output is once again a vector with the probabilities of each player performing the action.

LSTM states update The new hidden- and cell state are computed as follows. The output of the first fully-connected layer with ReLU activation at both the encoder and decoder are concatenated and fed into an LSTM cell, along with the hidden- and cell state from the previous timestep ($\mathbf{h}_{t-1}$ and $\mathbf{C}_{t-1}$). The LSTM cell then outputs the new states ($\mathbf{h}_t$ and $\mathbf{C}_t$).

The total loss is now the sum of the loss at each timestep:

$$L_{\theta,\phi} = \sum_{t=1}^{10} CE(pl_t, pl'_t) + \beta D_{KL}(q_\phi(\mathbf{z}_t|\, pl_t, \mathbf{c}_t) \,||\, p_\theta(\mathbf{z}_t|\mathbf{c}_t))$$

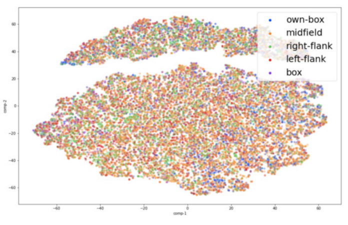
4.3.4 Model evaluation

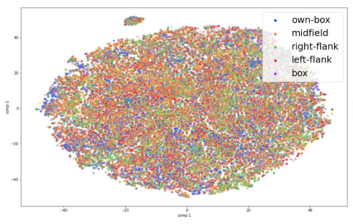
Accuracy For training the CVRNN model, the same configurations as for the CVAE model are used: it is trained for 50 epochs with early stopping using a learning rate of 0.001 and the Adam optimizer. Table 4.4 shows the accuracy for the three datasets with and without a warm-up period. The accuracies are a lot higher than those of the CVAE model, especially for the two FAWSL seasons. The warm-up period only slightly improves the performance for La Liga and FAWSL 20/21, and the accuracy for FAWSL 19/20 decreases by a very small amount when using warm-up. We can conclude that adding play history significantly improves identifying the correct players. Liu et al. [3] do not use a warm-up period in their CVRNN model, which has an accuracy that lies around 46%.

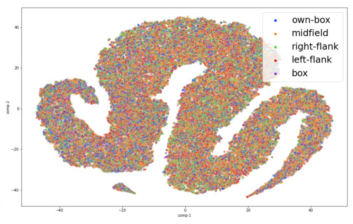
	La Liga	FAWSL 2019/2020	FAWSL 2020/2021
No warm-up	0.443	0.534	0.486
Warm-up	0.444	0.532	0.493

Table 4.4: Accuracy of the CVRNN for three different competitions

Visualizations While the accuracy of the CVRNN improved a lot over the CVAE, the quality of the player representations did not. The t-SNE visualizations of all players and of six chosen players when a random sample from the posterior distribution is chosen as the embeddings are similar to those of Figure 4.4 and 4.5 both with- and without warm-up, where the embeddings form one big cloud of points (which is why they are not shown here). The visualizations using the mean vector as embedding differ in shape, but still lack interpretable clusters in most cases. Figures 4.13 and 4.14 show the embeddings of all players, and Figures 4.15 and 4.16 the embeddings of six chosen players with- and without warm-up respectively.

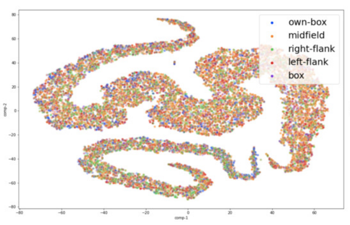


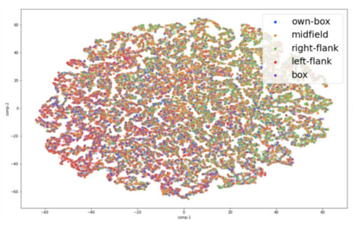


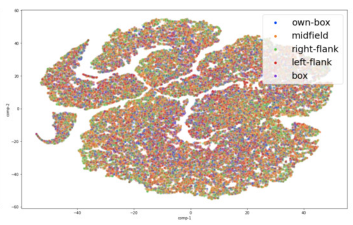


(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.13: Visualization of all player representations labelled by start-area using the mean vector as player embedding

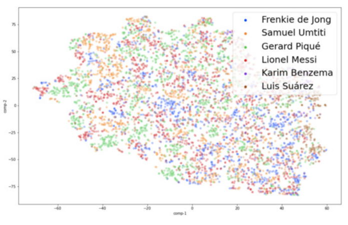


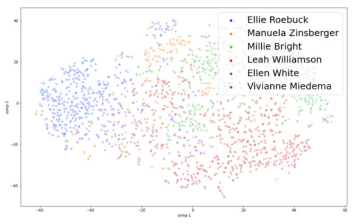


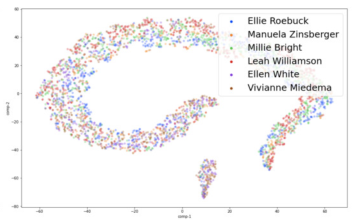


(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.14: Visualization of all player representations labelled by start-area using the mean vector as player embedding and a warm-up period

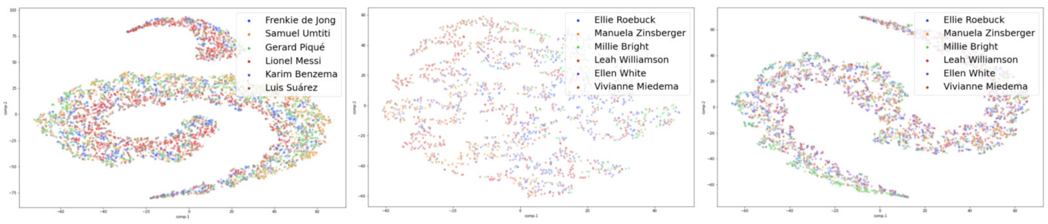






(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.15: Visualization of the player representations of six chosen players using the mean vector as player embedding

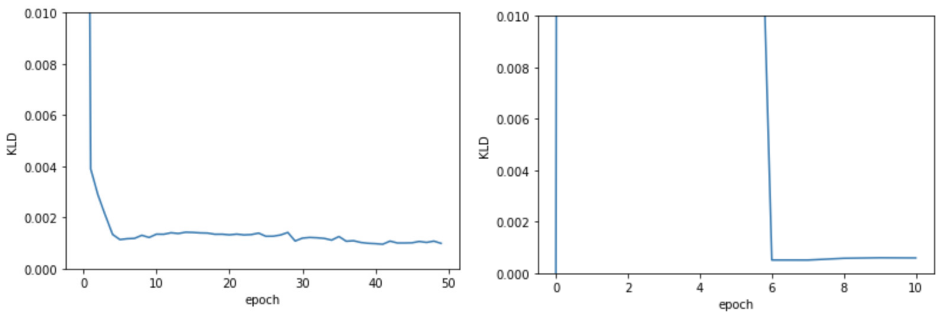


(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.16: Visualization of the player representations of six chosen players using the mean vector as player embedding and a warm-up period

Only for the plot of six players of the FAWSL 19/20 season without warm-up (and a bit for the La Liga players too) are the embeddings placed somewhat near to embeddings of the same player. Compared to the CVAE, using a warm-up period did not help in improving the quality of the learned embeddings, as the figures show.

Posterior collapse The CVRNN experiences the same problem of posterior collapse as the CVAE: the KL-divergence drops to a small value (around 0.001 for the FAWSL 20/21 data) after only two training epochs (see Figure 4.17 (A)). When the warm-up period is used, the KL-divergence first rises (because β is increased) and then drops to a small value again (around 0.0005) when β has increased to one (Figure 4.17 (B)). We can thus conclude that only using the warm-up does not solve the issue of posterior collapse for the CVRNN.



(A) No warm-up (B) Warm-up

FIGURE 4.17: KL-divergence during training of the CVRNN on the FAWSL 20/21 data

Player de-anonimization We now conduct the player de-anonimization experiment that was introduced in Section 4.2.4 on the CVRNN model with a warm-up period (Table 4.5). Although the scores are slightly better than those of the CVAE model, they are still very low. Just like with the CVAE (Table 4.3), the reason for the poor performance is the posterior collapse problem that the model suffers from.

	Top-1	Top-3	Top-5	Top-10	MRR
La Liga	0.002	0.006	0.010	0.025	0.026
FAWSL 19/20	0	0.004	0.009	0.024	0.020
FAWSL 20/21	0.001	0.006	0.013	0.036	0.023

TABLE 4.5: Top-k percentage and MRR of the CVRNN model on the three datasets

4.4 Model 3: VaRLAE

The third and final model is called the Variational Recurrent Ladder Agent Encoder (VaRLAE) [3]. It is another recurrent model like the CVRNN, with the addition of a hierarchical latent space. Every RNN cell contains a Ladder-VAE (LVAE) [26], where three latent variables are used in a hierarchical structure instead of one. The three latent variables relate to different features of the context variables, who are first split up to form three different inputs instead of one $\mathbf{c}_t$.

4.4.1 Additional data preprocessing

The context variables $\mathbf{c}_t$ are split into three: $\mathbf{s}_t$, $\mathbf{a}_t$, $\mathbf{r}_t$. These components follow a Markov Game Model [17], where an agent at game state $\mathbf{s}_t$ performs an action $\mathbf{a}_t$ and receives a reward $\mathbf{r}_t$ ($\mathbf{s}_t \to \mathbf{a}_t \to \mathbf{r}_t$). Liu et al. [3] choose to let $\mathbf{a}_t$ contain only the action, $\mathbf{r}_t$ the reward and $\mathbf{s}_t$ the rest of the features. Our approach is similar: $\mathbf{r}_t$ contains the result_id, $\mathbf{a}_t$ the type_id and bodypart_id, and $\mathbf{s}_t$ all other features.

4.4.2 Architecture

One cell of the VaRLAE network is shown in Figure 4.18. There are now four inputs: the player at time t, pl_t, and the context variables $\mathbf{s}_t$, $\mathbf{a}_t$ and $\mathbf{r}_t$. The inference part of the network consists of multiple layers following a dependency structure. Deterministic variables d_t are calculated from the input variables (player and context) and the variables from lower layers. The information flow is thus bottom-up. The direction of information flow is reversed when obtaining the approximate posterior distributions. The posterior distributions are again Gaussians and their parameters are computed by a weighted combination of the upper layers' prior distribution parameters ($\boldsymbol{\mu}_t^{prior}$ and $\boldsymbol{\sigma}_t^{prior}$) and the deterministic variables d_t from the same layer. The parameters of the prior distribution come from the generative process (the decoder), so information is being shared between the generative model and the inference model (hence the double lines between the latent variables in Figure 4.18). The bottom posterior distribution $q_\phi(\mathbf{z}_{r,t}|\, pl_t, \mathbf{r}_t, \mathbf{z}_{a,t})$ captures information from upper layers and serves as the contextualized player representation of player pl_t under the given match context. The priors are computed from the context variables and from the latent variables of upper layers. It is from here that the latent variables are sampled to predict the acting player at the output layer.

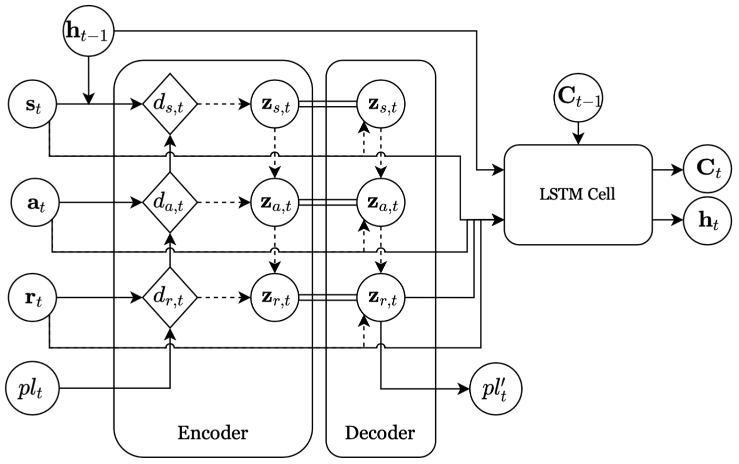


FIGURE 4.18: Schematic of the VaRLAE architecture

For the computation of the deterministic variables d_t and for the prior parameters $\boldsymbol{\mu}_{\boldsymbol{t}}^{\boldsymbol{prior}}$ and $\boldsymbol{\sigma}_{\boldsymbol{t}}^{\boldsymbol{prior}}$, batch normalization is used at each layer. Batch normalization [27] aims to stabilize neural networks through normalization of each batch of input data by subtracting the batch mean (re-centering) and dividing by the batch standard deviation (re-scaling). Its initial goal was to reduce the shift of the distributions (means and variances) of the networks' hidden layers, called the internal covariate shift. The usage of batch normalization has some additional benefits. It allows the usage of larger learning rates with a smaller chance of exploding or vanishing gradients, and it is believed to reduce chances of overfitting as it adds slight regularization. It also makes the network more robust to different learning rates and less dependent on the initialization.

Liu et al. [3] report that the hierarchical latent variables prevent posterior collapse, which is often an issue when using high capacity decoders. The ladder structure should ensure that the decoder uses the latent variables during the generation process, and so that the latent variables contain meaningful information about the acting player in the given context. Because of the top-down information flow, the decoder is forced to use the latent variables from upper layers in order to make use of the context variables.

4.4.3 Implementation

At first, the deterministic variables d_t are computed using two inputs at each layer starting at the bottom with $d_{r,t}$ which uses pl_t and $\mathbf{r}_t$ (Figure 4.19). The context input $\mathbf{r}_t$ first goes through a fully-connected layer (denoted as F-C in the figures) after which it is concatenated with the other input, pl_t in this case. It is then fed into another fully-connected layer, followed by batch normalization and ReLU activation to obtain $d_{r,t}$. From $d_{r,t}$, we obtain the Gaussian parameters $\hat{\boldsymbol{\mu}}_{\boldsymbol{r,t}}^{\boldsymbol{enc}}$ and $\hat{\boldsymbol{\sigma}}_{\boldsymbol{r,t}}^{\boldsymbol{enc}}$ from fully-connected layers with linear and softplus activation respectively. These will be used later in the computation of the posterior parameters. At the higher layers, the same functions are applied using different inputs. Instead of pl_t, the deterministic variable from one layer lower is used as the first input and the second input becomes the corresponding context variable ($\mathbf{a}_t$ at the middle layer and $\mathbf{s}_t$ at the top layer). $\mathbf{s}_t$ is first concatenated with $\mathbf{h}_{t-1}$ for the calculation of $d_{s,t}$.

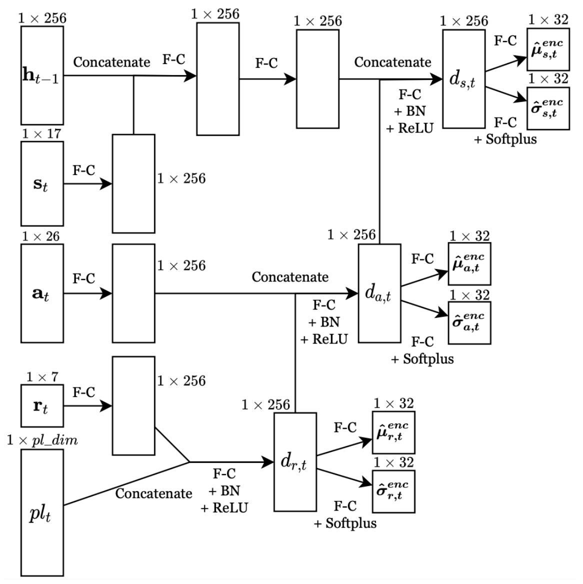


FIGURE 4.19: Layers and dimensions of the VaRLAE upward pass

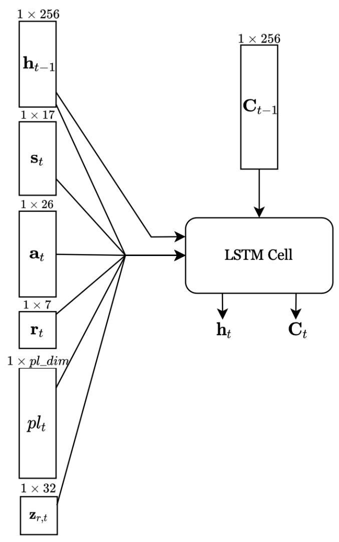


FIGURE 4.20: Layers and dimensions of the VaRLAE LSTM hidden state update

Then, the parameters of the approximate posterior- and prior distributions are computed in a stochastic downward pass (Figure 4.21). For the prior distributions, we concatenate at each layer the context variable and latent variable $\mathbf{z}_{(+),t}$ from the upper layer after being processed by fully-connected layers (at the top layer, $\mathbf{z}_{(+),t}$ is replaced by $\mathbf{h}_{t-1}$). The Gaussian parameters $\boldsymbol{\mu}_t^{\boldsymbol{prior}}$ and $\boldsymbol{\sigma}_t^{\boldsymbol{prior}}$ are obtained by feeding this into the same kind of layers used for calculating $\hat{\boldsymbol{\mu}}_t^{\boldsymbol{enc}}$ and $\hat{\boldsymbol{\sigma}}_t^{\boldsymbol{enc}}$. At the top layer, the parameters of the posterior distribution $\boldsymbol{\mu}_{\boldsymbol{s,t}}^{\boldsymbol{enc}}$ and $\boldsymbol{\sigma}_{\boldsymbol{s,t}}^{\boldsymbol{enc}}$ are equal to $\hat{\boldsymbol{\mu}}_{\boldsymbol{s,t}}^{\boldsymbol{enc}}$ and $\hat{\boldsymbol{\sigma}}_{\boldsymbol{s,t}}^{\boldsymbol{enc}}$ calculated earlier. At the lower layers, the parameters are obtained by a weighted combination of ($\hat{\boldsymbol{\mu}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}}$, $\hat{\boldsymbol{\sigma}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}}$) and ($\boldsymbol{\mu}_{\boldsymbol{c,t}}^{\boldsymbol{prior}}$, $\boldsymbol{\sigma}_{\boldsymbol{c,t}}^{\boldsymbol{prior}}$) (with $c \in \{\mathbf{a}, \mathbf{r}\}$):

$$\boldsymbol{\mu}_{\boldsymbol{c,t}}^{\boldsymbol{enc}} = \frac{\hat{\boldsymbol{\mu}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}}(\hat{\boldsymbol{\sigma}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}})^{-2} + \boldsymbol{\mu}_{\boldsymbol{c,t}}^{\boldsymbol{prior}}(\boldsymbol{\sigma}_{\boldsymbol{c,t}}^{\boldsymbol{prior}})^{-2}}{(\hat{\boldsymbol{\sigma}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}})^{-2} + (\boldsymbol{\sigma}_{\boldsymbol{c,t}}^{\boldsymbol{prior}})^{-2}} \; and \; \boldsymbol{\sigma}_{\boldsymbol{c,t}}^{\boldsymbol{enc}} = \frac{1}{(\hat{\boldsymbol{\sigma}}_{\boldsymbol{c,t}}^{\boldsymbol{enc}})^{-2} + (\boldsymbol{\sigma}_{\boldsymbol{c,t}}^{\boldsymbol{prior}})^{-2}}$$

The player output pl'_t is computed by feeding $\mathbf{z}_{r,t}$ into two consecutive fully-connected layers with ReLU and softmax activation. The next hidden state is computed by feeding $\psi([\mathbf{z}_{r,t}, \mathbf{r}_t, \mathbf{a}_t, \mathbf{s}_t, pl_t])$ into an LSTM cell (Figure 4.20), where $\psi(.)$ is a fully connected layer with ReLU activation and [.] represents a concatenation. Lastly, the

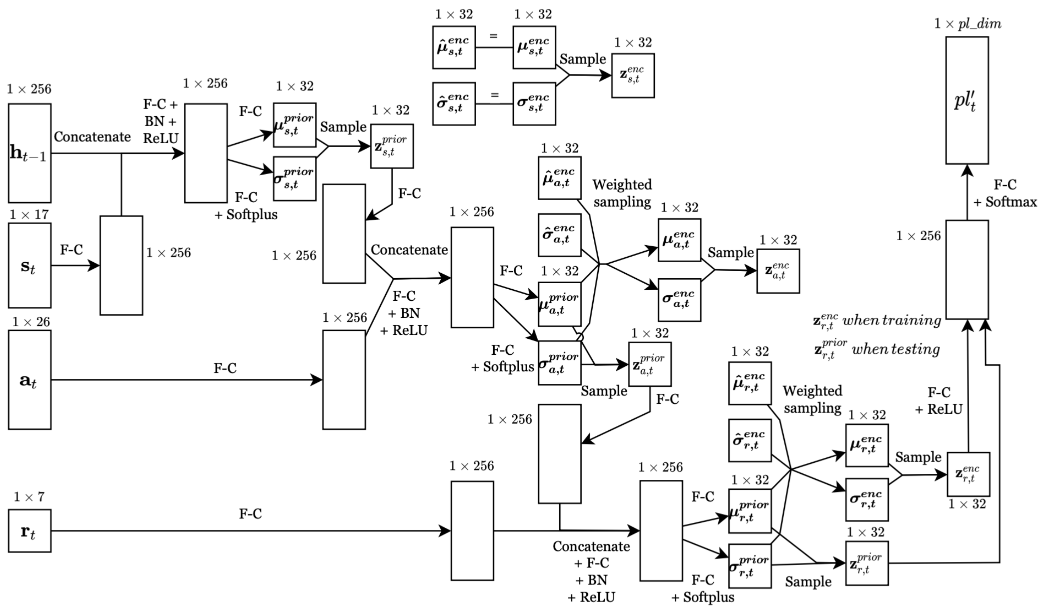


FIGURE 4.21: Layers and dimensions of the VaRLAE downward pass

loss function becomes:

$$L_{\theta,\phi} = \sum_{t=1}^{10} \left[\sum_{c\in\{\mathbf{s},\mathbf{a},\mathbf{r}\}} CE(pl_t, pl'_t) + \beta D_{KL}(q_\phi(\mathbf{z}_{c,t}|\, pl_t, c_t, \mathbf{z}_{(+),t}) \,||\, p_\theta(\mathbf{z}_{c,t}|c_t, \mathbf{z}_{(+),t})) \right]$$

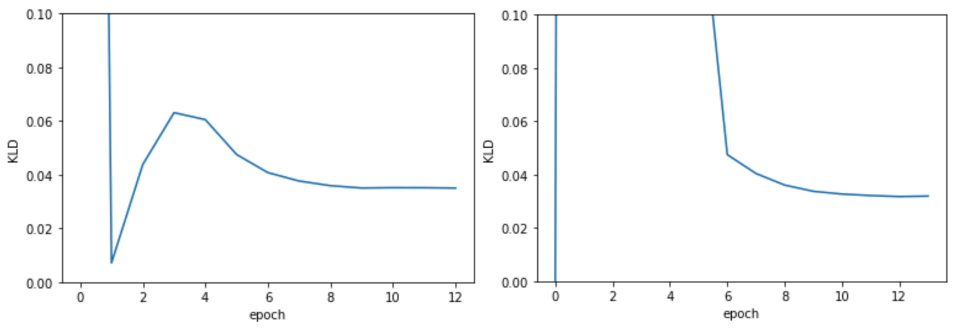
4.4.4 Model evaluation

Accuracy Table 4.6 shows the accuracies of the VaRLAE model on the three competitions with- and without using a warm-up period. The Adam optimizer and a learning rate of 0.01 were used for training the model. The learning rate is larger than in previous experiments because the VaRLAE model tends to be unstable for smaller learning rates when a warm-up period is used. Some of the weights get NaN values, rendering the model and the embeddings useless. The accuracies of the VaRLAE model are higher for all competitions than those of the CVRNN, and the warm-up period also gives a small improvement in accuracy. Once again, our accuracies are similar to those of Liu et al. [3] for the ice hockey case, whose VaRLAE model achieves an accuracy around 50%.

	La Liga	FAWSL 2019/2020	FAWSL 2020/2021
No warm-up	0.446	0.550	0.537
Warm-up	0.451	0.579	0.538

TABLE 4.6: Accuracy of the CVRNN for three different competitions

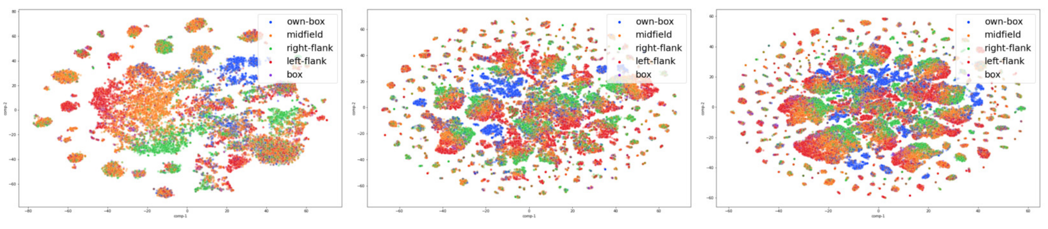
KL-divergence When looking at the KL-divergence during training (Figure 4.22), we see that it converges to a value of around 0.04 after six training epochs for the models with- and without warm-up. This is a lot higher than in the CVRNN case, where the KLD dropped to values around 0.001 very quickly. This suggests that the VaRLAE model does not experience the posterior collapse problem like previous models.



(A) No warm-up (B) Warm-up

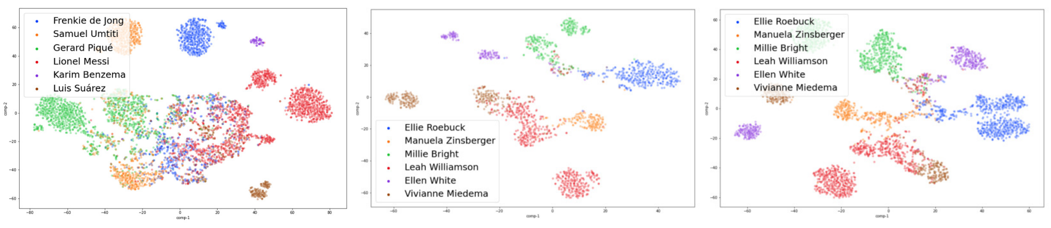
FIGURE 4.22: KL-divergence during training of the VaRLAE on the FAWSL 20/21 data

Visualizations Figure 4.23 and 4.24 show the t-SNE visualizations of all player embeddings and the embeddings of six players respectively. The embeddings in these figures are random samples from the posterior distributions $q_\phi(\mathbf{z}_{r,t}|\, pl_t, \mathbf{r}_t, \mathbf{z}_{a,t})$. In previous models, choosing the mean vector as the player embedding resulted in better looking figures. The variances were very high relative to the means in those models, but this is not the case anymore for the VaRLAE model. The plots using the mean vector as embedding look almost the same now as the plots when using random samples, so only the latter are shown. The plots of the models that use a warm-up period also look very similar to the ones who do not, so they are also not shown here.



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.23: Visualization of all player representations labelled by start-area



(A) La Liga (B) FAWSL 19/20 (C) FAWSL 20/21

FIGURE 4.24: Visualization of the player representations of six chosen players

The plots demonstrate that the quality of the embeddings have improved a lot. In Figure 4.23, many clusters of the same color are visible, so the embeddings of players who perform actions in the same area are similar. The clusters of embeddings in Figure 4.24 are nicely separated, and embeddings of the same players appear in the same clusters. Some clusters lie close together or have overlapping areas, which make sense intuitively. In the FAWSL 19/20 and 20/21 figures, we see that the embeddings of Manuela Zinsberger, Leah Williamson and Vivianne Miedema appear close together. These are all players from the same team (Arsenal WFC). Other players from the same team (Manchester City WFC) are Ellie Roebuck and Ellen White, whose embeddings also lie close together in the FAWSL 20/21 plot. The embeddings of players in the same position are also similar: Ellie Roebuck and Manuela Zinsberger are both goalkeepers, and Ellen White and Vivianne Miedema are both strikers. The resemblance is less clear for the two centre-backs Leah Williamson and Millie Bright.

Player de-anonimization Table 4.7 shows the results of the player de-anonimization task for the VaRLAE model. The scores are a lot higher than those of the CVAE and CVRNN models, proving that the embeddings can indeed be useful in downstream tasks. We also make a comparison with the player representations learned by SoccerMix (Section 3.1.2), trained on the three datasets (Table 4.8). While the scores of that model are not bad at all, the VaRLAE model achieves higher scores on all metrics. What is also interesting is to examine the individual rankings and see which players other than the player for whom the ranking was established appear in the list. This is done for the VaRLAE model later in Section 6.1. In the next chapter, we explore different configurations of the VaRLAE model, to see if the performance can be improved even more.

	Top-1	Top-3	Top-5	Top-10	MRR
La Liga	0.418	0.546	0.610	0.699	0.511
FAWSL 19/20	0.515	0.665	0.729	0.812	0.614
FAWSL 20/21	0.478	0.642	0.713	0.810	0.587

TABLE 4.7: Top-k percentage and MRR of the VaRLAE model on the three datasets

	Top-1	Top-3	Top-5	Top-10	MRR
La Liga	0.255	0.415	0.479	0.606	0.371
FAWSL 19/20	0.301	0.485	0.578	0.757	0.435
FAWSL 20/21	0.361	0.530	0.602	0.703	0.477

Table 4.8: Top-k percentage and MRR of the SoccerMix model on the three datasets

Chapter 5

Experiments

In this chapter, the outcomes of experiments on the models from Section 4 are described. First, we test the VaRLAE model on multiple configurations, in which we make use of the MRR metric introduced in the previous chapter to test their performance. Then, we explore the player representations more thoroughly by examining their component values and looking at the effect of changing those values.

5.1 Experiments on the VaRLAE model

In this section, we experiment with various learning rates as well as different latent space dimensions for the VaRLAE model, in order to find out which configuration has the best performance. Then, the importance of some components of the model is assessed. We look at the performance when we leave out batch normalization, as well as when we leave out some information of the context variables by using the intermediate representations $\mathbf{z}_{a,t}$ and $\mathbf{z}_{s,t}$ as our player representations.

5.1.1 Experiments on the learning rate

We first consider the case where the dimensions of the latent variables $\mathbf{z}_{s,t}$, $\mathbf{z}_{a,t}$ and $\mathbf{z}_{r,t}$ are all equal to 32. The accuracy as well as the MRR metric is reported for each competition using learning rates of 0.01, 0.001 and 0.0001 on the VaRLAE model with- and without warm-up (Table 5.1 and 5.2). The MRR of the player de-anonimization task introduced in Section 4.2.4 is included because the accuracy metric does not provide insight on the quality of the player representations.

Learning rate		La Liga	FAWSL 19/20	FAWSL 20/21
0.01	No warm-up	0.446	0.550	0.537
	Warm-up	0.451	**0.579**	0.538
0.001	No warm-up	**0.453**	0.567	0.527
	Warm-up	/	0.557	/
0.0001	No warm-up	0.435	0.569	**0.541**
	Warm-up	/	0.566	0.533

Table 5.1: Accuracy of the VaRLAE model for different learning rates on the three competitions, where $dim_{\mathbf{z}_{s,t}} = dim_{\mathbf{z}_{a,t}} = dim_{\mathbf{z}_{r,t}} = 32$

Learning rate		La Liga	FAWSL 19/20	FAWSL 20/21
0.01	No warm-up	0.511	0.611	**0.587**
	Warm-up	**0.512**	**0.627**	0.578
0.001	No warm-up	0.510	0.620	0.585
	Warm-up	/	0.578	/
0.0001	No warm-up	0.482	0.622	0.587
	Warm-up	/	0.610	0.545

Table 5.2: MRR of the VaRLAE model for different learning rates in the three competitions, where $dim_{\mathbf{z}_{s,t}} = dim_{\mathbf{z}_{a,t}} = dim_{\mathbf{z}_{r,t}} = 32$

The accuracies only vary slightly, and for each competition a different learning rate gives the best result. For the MRR, a learning rate of 0.01 resulted in the highest scores. The model can be unstable when the warm-up period is used, especially when smaller learning rates are used. Some of the weights would get NaN values so that the accuracy and MRR could not be measured (indicated with '/' in the tables).

5.1.2 Experiments on the latent space dimensions

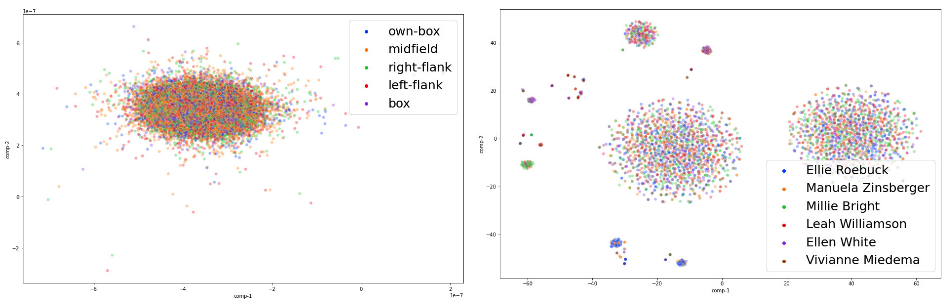
Table 5.3 shows the MRR metric of the VaRLAE model for different learning rates, now using a dimension of 64 for $\mathbf{z}_{s,t}$ and $\mathbf{z}_{a,t}$. The dimension of $\mathbf{z}_{r,t}$ (dimension of the player representations) is kept at 32 to allow fair comparison between the different models. The best scores now appear at different learning rates, and the models seem somewhat more stable because fewer results are undefined. For the La Liga and FAWSL 20/21 datasets, the highest MRR scores across all learning rates are better than those of the model with $dim_{\mathbf{z}_{s,t}} = dim_{\mathbf{z}_{a,t}} = dim_{\mathbf{z}_{r,t}} = 32$, although the differences are small. Achieving the best results thus depends on the dataset, as there is no clear "best" learning rate or latent space dimensions to use.

Learning rate		La Liga	FAWSL 19/20	FAWSL 20/21
0.01	No warm-up	0.507	0.601	0.591
	Warm-up	**0.517**	0.610	/
0.001	No warm-up	0.486	0.610	0.584
	Warm-up	0.512	**0.617**	/
0.0001	No warm-up	0.495	0.576	**0.604**
	Warm-up	0.494	0.525	0.537

TABLE 5.3: MRR of the VaRLAE model for different learning rates on the three competitions, where $dim_{\mathbf{z}_{s,t}} = dim_{\mathbf{z}_{a,t}} = 64$ and $dim_{\mathbf{z}_{r,t}} = 32$

5.1.3 Experiment on batch normalization

Many processing steps in the VaRLAE network involve feeding an input to a fully-connected layer, followed by batch normalization and ReLU activation. We now train a VaRLAE model without batch normalization to investigate the effect it has on the model's performance. Figure 5.1 shows the t-SNE visualizations of all players labelled by starting area of the action and the case study of six chosen players for the FAWSL 20/21 data. Both figures show clouds of points with all mixed labels, so no separated clusters containing points with the same label as when batch normalization was used (Figure 4.23 and 4.24).



(A) All players (B) Six chosen players

FIGURE 5.1: t-SNE visualizations of (A) all players labelled by start-area of the action and (B) six chosen players of the FAWSL 20/21 data

Table 5.4 shows the top-k results and MRR from the player de-anonimization task on the FAWSL 20/21 data for the VaRLAE model with- and without batch normalization. The model without batch normalization achieves scores that are a lot lower than the standard model. This proves that batch normalization is crucial for the VaRLAE model to learn quality player representations.

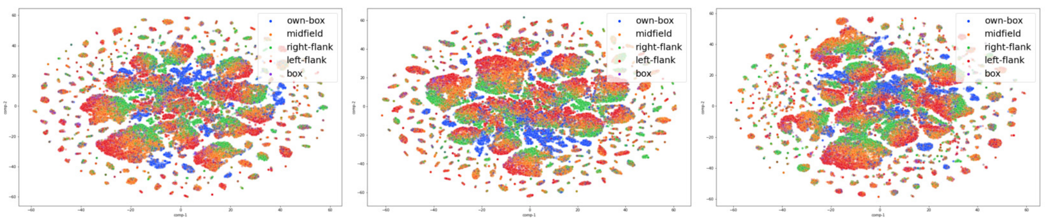
	Top-1	Top-3	Top-5	Top-10	MRR
With batch normalization	0.478	0.642	0.713	0.810	0.587
Without batch normalization	0.006	0.022	0.039	0.081	0.039

TABLE 5.4: Top-k percentage and MRR for the VaRLAE model with- and without batch normalization on the FAWSL 20/21 data

5.1.4 Analysis of the intermediate latent variables

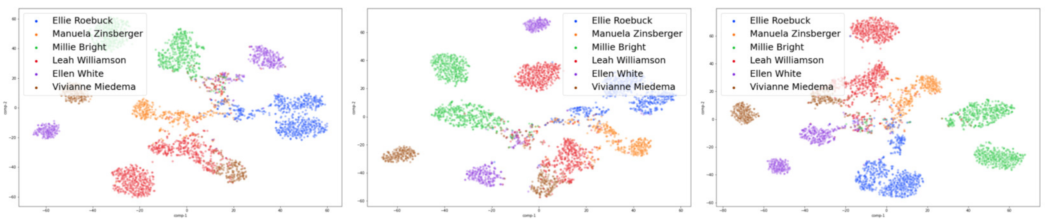
In our VaRLAE model, the latent variables $\mathbf{z}_{r,t}$ are used as the player representations. It has the most complete information about the players because it conditions on all context variables and receives all information from higher layers. We now experiment with the other latent variables $\mathbf{z}_{a,t}$ and $\mathbf{z}_{s,t}$. These variables have access to less information than those at the lowest layer, i.e. $\mathbf{z}_{a,t}$ has access to only $\mathbf{a}_t$ and $\mathbf{s}_t$ and $\mathbf{z}_{a,t}$ only to $\mathbf{s}_t$. The embeddings are visualized using t-SNE as was done for $\mathbf{z}_{r,t}$ in Section 4.4.4, and we compare their performance on the player de-anonimization task.

t-SNE visualizations Figure 5.2 shows the t-SNE visualizations of (A) $\mathbf{z}_{r,t}$, (B) $\mathbf{z}_{a,t}$ and (C) $\mathbf{z}_{s,t}$ of all players in the FAWSL 20/21 dataset, labelled by start-area. The visualizations look very similar, suggesting that the embeddings will also be very similar. The case study for six players is shown in Figure 5.3. We observe the same as in the previous figure, namely that the plots are very much alike. The same is true for the other datasets, which is why they are not shown here.



(A) $\mathbf{z}_{r,t}$ (B) $\mathbf{z}_{a,t}$ (C) $\mathbf{z}_{s,t}$

FIGURE 5.2: Visualization of all latent space embeddings $\mathbf{z}_{r,t}$, $\mathbf{z}_{a,t}$ and $\mathbf{z}_{s,t}$ of the FAWSL 20/21 data, labelled by start-area



(A) $\mathbf{z}_{r,t}$ (B) $\mathbf{z}_{a,t}$ (C) $\mathbf{z}_{s,t}$

FIGURE 5.3: Visualization of the latent space embeddings $\mathbf{z}_{r,t}$, $\mathbf{z}_{a,t}$ and $\mathbf{z}_{s,t}$ of six chosen players of the FAWSL 20/21 data

Player de-anonimization We now look at the performance of the intermediate embeddings when they are used in the player de-anonimization task from Section 4.2.4. Table 5.5 shows the top-k results and the MRR for the three latent variables learned by training on the FAWSL 20/21 data.

Latent variable	Top-1	Top-3	Top-5	Top-10	MRR
$\mathbf{z}_{r,t}$	0.478	0.642	0.713	0.810	0.587
$\mathbf{z}_{a,t}$	0.439	0.616	0.683	0.770	0.554
$\mathbf{z}_{s,t}$	0.430	0.616	0.688	0.776	0.551

TABLE 5.5: Top-k percentage and MRR when using different latent variables as player embeddings on the FAWSL 20/21 data

As expected, the embeddings that contain the most information ($\mathbf{z}_{r,t}$) achieve the best scores on each metric. The embeddings who have access to the least amount of information ($\mathbf{z}_{s,t}$) have the worst MRR score, although doing better on the top-5 and top-10 metrics than $\mathbf{z}_{a,t}$. However, the performance of the intermediate embeddings is only slightly worse than our usual player representations $\mathbf{z}_{r,t}$. This suggests that most information used for distinguishing players is contained in the $\mathbf{z}_{s,t}$ latent variables, and that the action- and result context variables only add a small contribution.

5.2 Analysis of the player representations

We now further analyze the player representations learned by the VaRLAE model. We first examine specific components of the latent variables and their importance to the decoder prediction. Then, we verify what happens at the decoder output when a component of the latent variable is changed. In this way, we can find out which components of our player representations are important in order to distinguish them.

5.2.1 Analysis of the latent variable components

In previous work on soccer player representations, the components of the vectors that are obtained are human-interpretable. In Player Vectors (Section 3.1.1), each component stands for a variation of an action, and the value of that component represents how often a player performs that action. The same idea is used in SoccerMix (Section 3.1.2), where the components represent prototypical actions. We will now investigate whether some of the 32 components of the embeddings learned by the VaRLAE model represent meaningful entities.

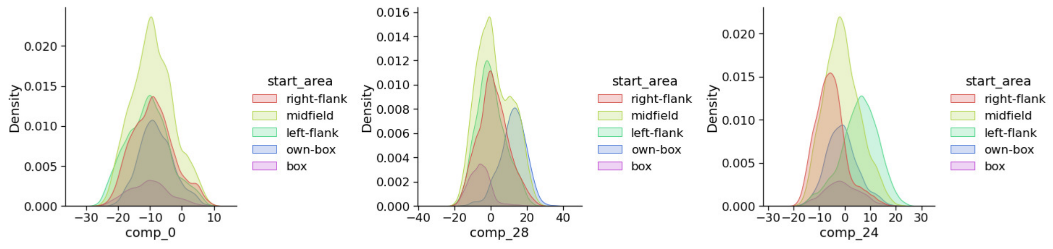
With entities, we mean things by which players in a certain match context can be described and distinguished from one another. A number of possible entities that could be important for distinguishing players are the player's team, the area where the player tends to perform most of his/her actions, the type of actions that the player likes to perform, ... Intuitively, when one of the representation vector's components represents a certain entity (which we don't know yet), players that differ on that entity will also have a different value for that component. For example, say that the first component of our player representations represents the team of that player. Players from different teams would then have different values for that component. This also means that for similar players in similar match contexts, the components (and thus the entities represented by that component) would be more alike. We can now choose an entity and look for each component how its values are distributed. When we take for example the team of a player as an entity, we can examine the distribution of a component's values for each team, i.e. we get one distribution per team. When we do this for each component, we can see for which components the values of different teams differ the most.

Approach We do this by first computing the player representations of the players performing the action at each event of the test set. Then, we take each component of the representations separately. The dimension of our player representations is 32, so there are 32 components in total. For a given component, we then obtain different distributions of the component's value for each value of the chosen entity. Take for example the type of action as entity. For each type of action, we can obtain the distribution of a certain component's value. The easiest way to examine these distributions is by visualizing them. A kernel density estimate (KDE) plot [1] is a way to visualize continuous probability density curves of a distribution, which is similar to a histogram, but smoothened. Each distribution can then be given a color representing the value (or type) of the entity. If these different distributions for a certain component do not have much overlap and their means lie far enough apart, we can assume that the component models (a part of) that entity. We show this idea for different entities.

Experiment 1: start-area as entity Figure 5.4 shows the distributions of three components of the FAWSL 20/21 player embeddings using the start-area attribute as entity. Many of the components look similar to that of component 0 in Figure 5.4 (A): the different distributions have lots of overlap and their peaks are located

[1] https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot

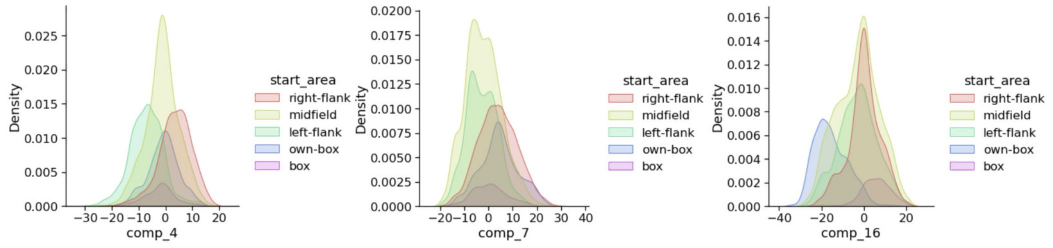
around the same value. This means that, regardless of the start-area of the action, most embeddings have a value of around -10 in their first component.



(A) (B) (C)

FIGURE 5.4: Visualization of the distributions of the value of three components from the FAWSL 20/21 player embeddings, labelled by start-area of the action

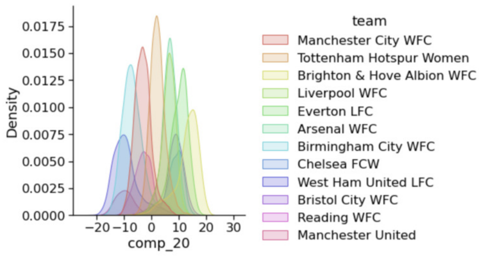
In Figure 5.4 (B), we see that there is less overlap in the distributions of the own-box- and box start-area and that they peak around different values for component 28 (the own-box distribution peaks around -5 and the box distribution around 15). This could mean that this component is (partly) responsible for modeling the y-coordinate of the embeddings' actions, because the box and own-box areas clearly have different y-coordinates, while the other areas not necessarily do. The same can be said about the x-coordinate of the action in component 24 (Figure 5.4 (C)). The distributions of the left- and right-flank areas also show little overlap, and actions in these areas differ most in their x-coordinate by definition. For the FAWSL 19/20 data, the same phenomenon occurs, but at different components. As we can see in Figure 5.5 (A) and (B), the left- and right-flank distributions are partially separated, meaning that multiple components can have a share in modelling a certain attribute (y-coordinate in this case). The figures for the La Liga data are similar and thus not shown here.



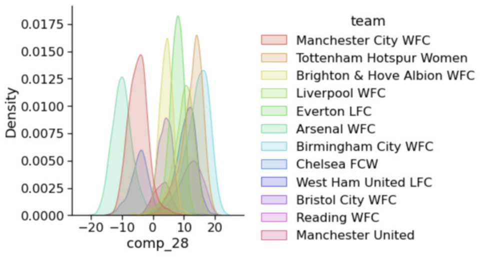
(A) (B) (C)

FIGURE 5.5: Visualization of the distributions of the value of three components from the FAWSL 19/20 player embeddings, labelled by start-area of the action

Experiment 2: team as entity As the next entity to label the distributions, the team of the player performing the action is used. Although the team_id was left out during training of the models (see Section 4.1), it was present in the original SPADL data, so each action can be mapped to the team of the player who performed the action. Figure 5.6 shows two components with a distribution for each team. Some teams are distributed around different values (e.g. the West Ham distribution reaches its peak around a value of -10 for component 20 and the one of Brighton around a value of 15), while the distributions of some other teams show more overlap. For the FAWSL 20/21 data, the observations are similar (see Figure 5.7), but less obvious than the FAWSL 19/20 case. We do not show the La Liga figure because of its bias for data of FC Barcelona, resulting in very high peaks for that team and smaller ones for all other teams.

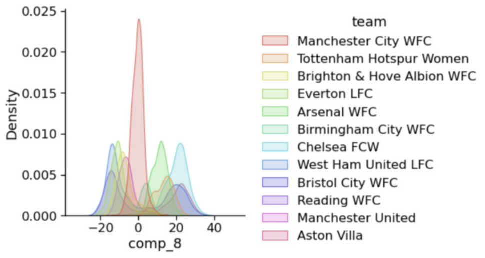


(A)

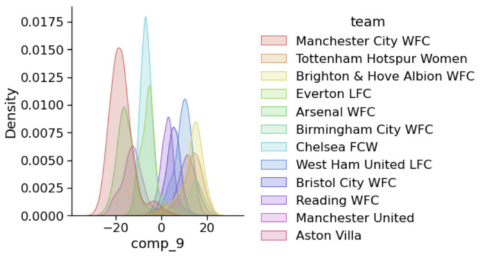


(B)

Figure 5.6: Visualization of the distributions of the value of two components from the FAWSL 19/20 player embeddings labelled by team



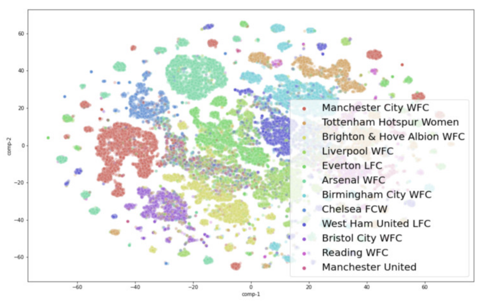
(A)



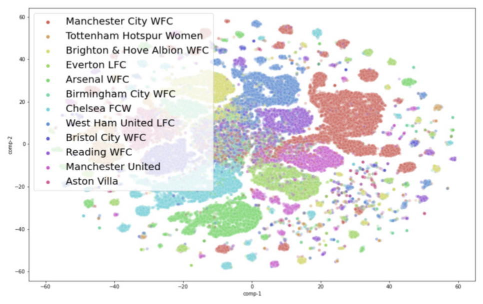
(B)

Figure 5.7: Visualization of the distributions of the value of two components from the FAWSL 20/21 player embeddings labelled by team

The t-SNE visualizations of all players labelled by the team of the acting player also clearly shows clusters with embeddings of players of the same team (see Figure 5.8 for the FAWSL 19/20 and 20/21 data). It is thus clear that the player embeddings contain some team information, even while the team labels were not used during training. A reason for this could be the recurrence of the VaRLAE model. The model might learn similar embeddings for players in the same sequence, and players that appear in the same sequence often play for the same team.



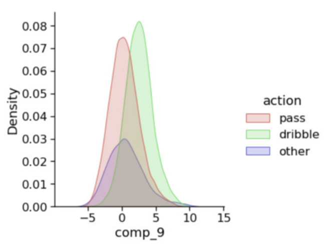
(A) FAWSL 19/20



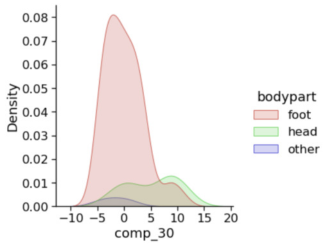
(B) FAWSL 20/21

FIGURE 5.8: Visualization of all player representations of the two FAWSL seasons, labelled by team

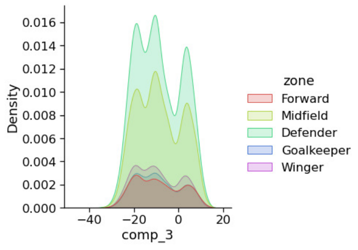
Experiment 3: other entities Other entities that were experimented with are action type, starting position, starting zone (a more general starting position, e.g. midfield, defense, ...), starting side (side of the starting position: left, right or center) and the body part type of the action. Many of the distributions in these experiments showed much overlap, so likely none of these attributes is clearly defined by a component of the embeddings. Figure 5.9 (A) shows the components with the least overlapping distributions for the action-type entity. To not overload the figure, all action types other than pass and dribble are given a ‘other’ label. This is fine for the experiment because only the distributions of the pass and dribble action types had some none-overlapping areas. In Figure 5.9 (B), we see the same thing using the body part type as the entity. The only actions that were considered here are shots, because most other actions are rarely done with body parts other than the foot. A random component is chosen for the starting zone as entity (Figure 5.9 (C)), where all distributions show much overlap (using FAWSL 19/20 data).



(A) action type



(B) bodypart



(C) starting zone

FIGURE 5.9: Visualization of the distributions of one component’s value labelled by (A) action type, (B) body part and (C) starting zone

5.2.2 Making changes to player representations

In this section, we will conduct an experiment where a value of the player embedding is changed. The new output of the decoder is then analyzed further, allowing us to evaluate the importance of certain components in the decoder predictions.

Changing component values of the player representations The following experiment was conducted to see what happens when we change the value of one or more components in a player representation. We want to see which players the decoder predicts when the value of the components that were correlated with the player's team is changed. For this, we select components 20 and 28 of the FAWSL 19/20 representations (see Figure 5.6). First, a 'shot' action that resulted in a goal is selected. More specifically, we select a goal of Adriana Leon from West Ham United, of which we compute the player representation. The distributions of West Ham reach their peaks at values around -10 and 10 for components 20 and 28 respectively. We now want to change these values to the ones of a distribution far away. The distributions of Brighton & Hove Albion are chosen because its values lie the furthest away from the West Ham distribution for component 20. The values of components 20 and 28 are set to 15 and 5 respectively, which is around where the distributions of Brighton reach their peaks. The altered representation is then put into the decoder, whose softmax output represents the probability of a player performing the action. The output is ordered by decreasing probability, and the top-5 of predicted players along with their team is shown in Table 5.6. The predictions from the original player representation are shown for comparison in Table 5.7.

Rank	Name	Team
1	Lisa Evans	Arsenal
2	Kayleigh Green	Brighton
3	Rebecca Jane	Liverpool
4	Ellie Brazil	Brighton
5	Loren Dykes	Bristol City

TABLE 5.6: Top-5 predictions of the changed player representation

Rank	Name	Team
1	Adriana Leon	West Ham
2	Cecilie Kvamme	West Ham
3	Angharad James	Reading
4	Alisha Lehmann	West Ham
5	Rachel Rowe	Reading

TABLE 5.7: Top-5 predictions of the original player representation

The decoder output from the original player representation ranked Adriana Leon as the player with the highest probability of performing the input action, as desired. Changing values in the player embedding results in a totally different prediction. The top predicted player is Lisa Evans, a forward at Arsenal, followed by Kayleigh Green, who is a forward at Brighton & Hove Albion. Interestingly, Brighton was also the team from which the distributions were chosen to base the new values on. Another Brighton forward, Ellie Brazil, appears in fourth position in the list. This is again an indication that there is team information embedded in the player representations.

Chapter 6

Using the player representations in downstream tasks

We now investigate whether our player representations can be useful in two downstream tasks. We first check whether the representations can be used to find players that are similar to a target player. Lastly, we test their applicability in the VAEP framework.

6.1 Finding similar players

To find players that are similar to a target player, we make up a ranking that orders the players based on the similarity (Manhattan distance) of their player representations with the representation of the target player. In Section 4.2.4, the rankings were constructed from the different representations for each action the player performs. Because we want to make one ranking for a player based on all his/her actions, we take the mean of the action-specific embeddings to get one representation per player.

Experiment 1: La Liga players Table 6.1 and 6.2 show the rankings of Marc-André Ter Stegen (goalkeeper at FC Barcelona) and Gerard Piqué (centre-back at FC Barcelona). Ter Stegen appears at the top of his own list and the other players that appear are all goalkeepers, except for Clément Lenglet, who is also a centre-back at FC Barcelona. While it is hard to compare a goalkeeper's style, the ranking shows that the VaRLAE model is able to find similarities between players in the same position. The ranking of similar players to Gerard Piqué also puts Piqué himself at the top, but the rest of the players are not all players who are known to be similar to him. VaRLAE puts many players that played for FC Barcelona that season high in the list. The model can still find similarities based on the position of the players reasonably well, as most of the players in this list are also defenders. Some players that are not really comparable in terms of playing style, but that play in the same team are ranked higher than players from a different team who play in the same position as Piqué, e.g. Ivan Rakitić (a midfielder from FC Barcelona that season) is ranked higher than Kenneth Omeruo (a centre back from CD Leganés).

Rank	Name
1	Marc-André Ter Stegen
2	Pichu
3	Neto
4	Jan Oblak
5	Fernando Pacheco
6	Marko Dmitrović
7	Tomáš Vaclík
8	Diego López
9	Clément Lenglet
10	Joel Robles

TABLE 6.1: Top-10 most similar players to Marc-André Ter Stegen

Rank	Name
1	Gerard Piqué
2	Jean-Clair Todibo
3	Ronald Araújo
4	Sergi Roberto
5	Ivan Rakitić
6	Moussa Wagué
7	Kenneth Omeruo
8	Emerson
9	Ander Iturraspe
10	Sergio Busquets

TABLE 6.2: Top-10 most similar players to Gerard Piqué

Experiment 2: FAWSL players We do the same thing for Vivianne Miedema, striker at Arsenal WFC, using the FAWSL 20/21 data (Table 6.3). Miedema appears at the top of the list, and the other players are mostly Arsenal players. Forwards (Caitlin Foord, Bethany Mead, Chloe Kelly, ...) are ranked higher on average than players from other positions, which is a good sign when comparing to a forward. Because we want to see which players from other teams are similar to our target player, we exclude Arsenal WFC players to create the ranking in Table 6.4. The top-10 consists mostly of strikers and forwards, except for Rosemary Lavelle who is an attacking midfielder. This demonstrates that, when players from the same team are excluded, the learned player embeddings can be used to find similar players.

Rank	Name
1	Vivianne Miedema
2	Caitlin Jade Foord
3	Jill Roord
4	Bethany Mead
5	Danielle van de Donk
6	Chloe Kelly
7	Jordan Nobbs
8	Ellen White
9	Kim Little
10	Leonie Maier

TABLE 6.3: Top-10 most similar players to Vivianne Miedema

Rank	Name
1	Chloe Kelly
2	Ellen White
3	Georgia Stanway
4	Rosemary Kathleen Lavelle
5	Jessica Park
6	Lauren Hemp
7	Francesca Kirby
8	Janine Elizabeth Beckie
9	Bethany England
10	Pernille Mosegaard Harder

TABLE 6.4: Top-10 most similar players to Vivianne Miedema, excluding Arsenal WFC players

6.2 Using the player representations in VAEP

VAEP was introduced in Section 3.2 as a framework for valuing player actions and rating player's performances. Intuitively, knowing which player performs the action can be useful information for valuing the action. It was already mentioned earlier that the underlying scoring probabilities were used as a way to evaluate the model. When the characteristics of a player performing an action are known, the predicted scoring probability can be adjusted to those characteristics. For example, when the action in question is a free-kick shot, it is more likely that the ball goes in when it is taken by Lionel Messi than when some other player takes it. We now evaluate the performance of VAEP when the learned player representations are added to the feature set.

Experiment setup The data format and features used for training a VAEP model are very similar to what was used in training the VAE architectures. It also consists of actions described by a number of features in the SPADL format, where complex- and game context features are added to. There are two options for adding the representations to the VAEP model that are both experimented with:

1. A different player representation is computed for each action, so we add each **action-specific representation** as an extra feature to the data
2. One player representation per player, by taking the mean of all representations for each player

When a different representation is used for each action, some actions will be left out in the training and testing data. This is because of the way the input sequences of the VaRLAE model are defined (i.e. the same as for the CVRNN in Section 4.3.1). For each sequence, a representation is computed only for the final action in the sequence, so we get a representation for each action in a game except for the first nine actions (because of the shifting with which the sequences were obtained). Since VAEP models are trained with an XGBoost classifier [1], which can only handle numerical or boolean features, the player representation vectors are split up into 32 components. Each action thus gets 32 new features, where each of them represents one component of the player representation.

We now train different VAEP models on the La Liga, FAWSL 19/20 and FAWSL 20/21 data. We do this for the original features, as well as the extension with our player representations. Models with one player representation for each player and models with a representation for each action are both examined. Because the models with a representation for each action train on fewer data than the other models, we also train a standard VAEP model (without representations) on the same (reduced) data, to allow for fair comparison. For each configuration, we perform a grid search to tune the parameters of the XGBoost classifiers. A classifier is trained for each combination of hyperparameters, and the classifier with the best results is kept.

[1] `https://xgboost.readthedocs.io/en/stable/python/python_api.html`

Evaluation metrics The scoring function to use for choosing the best classifier is the Brier score in this case. The Brier score measures the calibration of the predictions, so that a lower Brier score means that the predicted probability distribution is closer to the actual one, i.e. that the predictions are better calibrated. Other metrics that are often used in probabilistic classifiers are AUROC and logarithmic loss. AUROC (Area Under the ROC-curve) is useful when classifying or ranking examples, but ignores the values of the predicted probabilities. Logarithmic loss measures the accuracy of the obtained predictions and is similar to the Brier score in the way that they can be minimized (like the Brier score, a lower log-loss is better). Decroos [28] argues that the Brier score is the most important metric to optimize here, because individual prediction errors are summed rather than multiplied like in logarithmic loss, which is appropriate when the resulting probabilities will be summed (e.g. for obtaining player ratings). In Table 6.5, we report all three metrics for comparing the standard and extended VAEP models, but mainly look at the Brier score. The predicted scoring- and conceding probabilities are evaluated for the standard VAEP model and the models extended with player representations. In each column, we compare different models:

(A) Compares the standard VAEP model trained on all data to the VAEP model extended with the mean player representations (option 2)

(B) Compares the standard VAEP model trained on fewer data (as explained earlier) to the VAEP model extended with per-action player representations (option 1)

Results The results for the VAEP model extended with one representation per player are fairly mixed. For the scoring probability models, the extension achieves better Brier scores on the La Liga and FAWSL 19/20 datasets. Except for the logarithmic loss on the FAWSL 19/20 data, the standard VAEP models achieve better scores on the other metrics. Only on the La Liga data, the extension achieves a better Brier score for the conceding probability model. The results on other metrics are mixed. For the VAEP models extended with a different representation per action, the results are already a lot better. The extended version achieves a better or equal Brier score on both the scoring- and conceding models for all competitions, although the differences are small. The results for the other metrics are generally in favor of standard VAEP, except for the conceding model on the FAWSL 20/21 data. A reason for this can be that during the grid search, the configuration with the best Brier score is chosen, so other configurations may still achieve better scores on the other metrics. We can conclude that extending VAEP with per-action player embeddings is an improvement to the standard VAEP model when the Brier score is put forward as the most important metric to optimize. The extension with one player embedding per player only works in some cases, so it is not the preferred option.

La Liga		Standard VAEP	Extended VAEP (one per player)	Standard VAEP	Extended VAEP (one per action)
Scores	Brier score	0.00886	**0.00878**	0.00882	**0.00874**
	Logarithmic loss	**0.04850**	0.04907	0.04827	**0.04818**
	AUROC	**0.74792**	0.72707	**0.75248**	0.74972
Concedes	Brier score	0.00165	**0.00164**	**0.00164**	**0.00164**
	Logarithmic loss	0.01235	**0.01228**	**0.01126**	0.01530
	AUROC	**0.77151**	0.66412	**0.79050**	0.75177
		(A)		(B)	

FAWSL 19/20		Standard VAEP	Extended VAEP (one per player)	Standard VAEP	Extended VAEP (one per action)
Scores	Brier score	0.00796	**0.00790**	0.00797	**0.00795**
	Logarithmic loss	0.04234	**0.04199**	**0.04227**	0.04237
	AUROC	**0.81414**	0.81178	**0.81722**	0.80935
Concedes	Brier score	**0.00267**	0.00270	**0.00268**	**0.00268**
	Logarithmic loss	**0.01795**	0.01851	**0.01801**	0.02287
	AUROC	**0.69456**	0.67261	**0.70166**	0.64346
		(A)		(B)	

FAWSL 20/21		Standard VAEP	Extended VAEP (one per player)	Standard VAEP	Extended VAEP (one per action)
Scores	Brier score	**0.01058**	0.01064	0.01067	**0.01065**
	Logarithmic loss	**0.05300**	0.05373	**0.05345**	0.05364
	AUROC	**0.82110**	0.81025	**0.81966**	0.81526
Concedes	Brier score	**0.00277**	0.00279	0.00280	**0.00277**
	Logarithmic loss	0.01723	**0.01718**	0.01744	**0.01726**
	AUROC	0.78806	**0.81159**	0.78909	**0.79554**
		(A)		(B)	

TABLE 6.5: Brier score, logarithmic loss and AUROC score comparison between (A) the standard VAEP model and the VAEP model extended with one representation per player and (B) the standard VAEP model (trained on fewer data) and the VAEP model extended with a different representation for each action

Rating players The extension of VAEP with player representations can also be evaluated by looking at the player ratings that the VAEP framework produces. An ordered list of the top-10 players of the FAWSL 20/21 season based on the player ratings of the standard- and the extended model is given in Table 6.6 and 6.7 respectively.

Rank	Name	VAEP value
1	Caroline Weir	7.265
2	Samantha Kerr	6.696
3	Lauren Hemp	6.001
4	Alex Greenwood	5.306
5	Stephanie Houghton	5.022
6	Chloe Kelly	4.712
7	Francesca Kirby	4.228
8	Katie McCabe	4.180
9	Caitlin Jade Foord	4.140
10	Vivianne Miedema	3.964

TABLE 6.6: Top-10 players of the FAWSL 20/21 season ranked by VAEP value of the standard model

Rank	Name	VAEP value
1	Samantha Kerr	8.007
2	Caroline Weir	7.013
3	Lauren Hemp	5.983
4	Stephanie Houghton	5.585
5	Alex Greenwood	5.471
6	Francesca Kirby	4.853
7	Vivianne Miedema	4.765
8	Katie McCabe	4.658
9	Caitlin Jade Foord	4.598
10	Chloe Kelly	4.473

TABLE 6.7: Top-10 players of the FAWSL 20/21 season ranked by VAEP value of the model extended with player representations

Both rankings contain exactly the same players, but some appear in a different order. The ranks of Sam Kerr and Caroline Weir are switched in the extended model, with Kerr taking the top spot. Both were included in the FAWSL team of the year [29], but Kerr also finished third in the Ballon d'Or that year behind Alexia Putellas and Jennifer Hermoso of FC Barcelona [30], so giving her the top spot seems justified. Francesca Kirby, who won the PFA Player of the Year [31] as well as the Barclays FA WSL Player of the Season [32] award, is ranked higher by the extended VAEP model. Vivianne Miedema also jumps three places in the second ranking, which is perhaps rightly so, given that she finished second in the top-scorers ranking with 18 goals that season, only three goals less than Sam Kerr.

Chapter 7

Conclusion

7.1 Summary

This thesis discussed an approach for building a model that learns contextualized soccer player representations from event stream data, along with an analysis of these representations and a series of experiments where they are used in down-stream soccer analytics tasks. Three models were built for the task, each one being an improvement on the last.

The first model was an extension to a Variational Autoencoder, the Conditional Variational Autoencoder (CVAE), which takes the match context in which players act as an extra input. The player representations condition on this extra context input, so that we get different representations under different game situations. The representations learned by this model could not be distinguished well based on the intuitions of which players are similar, and its performance on the player de-anonimization task was also insufficient. The second model, the Conditional Variational Recurrent Neural Network (CVRNN), built further on the CVAE by adding play history through the usage of Recurrent Neural Networks (RNNs). While this model achieved a better accuracy, it suffered from the problem of posterior collapse, where the decoder barely uses the latent space representations and thus only the context variables for the prediction of the acting player. This problem rendered the player representations useless, which was confirmed by its poor performance on the player de-anonimization task. The final model, the Variational Recurrent Ladder Agent Encoder (VaRLAE), solved the posterior collapse problem by splitting the latent space into three components, which made sure that the latent space embeddings contain useful information about the acting player. The representations learned by the VaRLAE model perform well on the player de-anonimization task, achieving a larger MRR than the representations of SoccerMix on the same data. Visualizations of the embeddings also made sense intuitively, as embeddings of the same player and of different players in the same position were often clustered together.

After building the models, we experimented with their player representations (especially the ones of the VaRLAE model). Experiments with different hyperparameters and components of the VaRLAE model led us to the conclusion that performance varies only slightly when using different learning rates or latent space dimensions, and that the usage of a warm-up period during training often gave a better performance but could sometimes result in an unstable model. The usage of batch normalization also seemed crucial for the model's performance. Then, the obtained player representations were analyzed further, where it was observed that some components of the representations showed a positive correlation with entities like the player's team and the starting area of his/her actions. The influence of the player's team became even more apparent when lists of the most similar players to a certain player were retrieved, which showed a lot of players from the same team along with players of the same position. Changing certain components of the embeddings that are suspected to be correlated with the team resulted in different players being predicted at the decoder. Finally, we used the player representations as an extra feature in the VAEP framework for player ratings and valuing actions. Including the player representations resulted in better Brier scores of the underlying scoring- and conceding probability model, but the scores of other metrics were worse in some cases.

In general, the player representations learned by the VaRLAE model achieve good results when trying to identify the acting player based on their representations, which shows that the model can distinguish players based on the actions that they perform in which match contexts. Because the representations of players in the same team are similar, finding players with a similar playing style is not straightforward using only the embeddings, although players in the same position are found when we look past that. The improved performance of VAEP extended with the player representations suggests that they could be used in other soccer analytics tasks as well.

7.2 Future work

While we already achieved some satisfactory results, there is a lot left to experiment with and to use the VaRLAE model along with its player representations in. The first thing that can be done is to test the applicability of player representations in other soccer analytics tasks. One such task is expected goals estimation. Expected goals (xG) is a metric that gives a value to a shot based on how likely it is to result in a goal. Classic features on which xG models are trained are match context features like location on the field, actions preceding the shot, etc.[33] Adding player representations as an extra feature to those models could definitely give an improvement, given the intuition that shots coming from certain players have a bigger chance of resulting in a goal than from some other players, which could be captured in the representations. Another task where the player representations could be useful as an additional feature is score difference prediction (as is also done by Liu et al. [3] for ice hockey) or win probability models, where a team's chances at winning is computed at any point in the game.

A logical extension to player representations that was not studied in this thesis are team representations. Instead of learning representations of single players, the model could learn representations that capture the playing style of a whole team. The current player representations learned by the VaRLAE model could also be further improved. While the goal was to capture similarities between players in terms of playing style, the model mainly finds players of the same team to be similar. Representations that depend less on this property would already be more useful.

Finally, it could be investigated how to turn the player representations into more human-interpretable vectors, where the components represent meaningful entities. This would allow the representations to be used in various other tasks like comparing players' styles, monitoring player development (i.e. observing how player representations change over time) or in game analysis, e.g. identifying what went wrong in a defeat by comparing the representations with those of a win. The applicability in these tasks with the current player representations is limited because of the lack of interpretability of the embeddings.

Acknowledgements

We thank StatsBomb for providing the data sets used in this work. We thank the RBFA for the gracious support and feedback.

Bibliography

[1] Tom Decroos and Jesse Davis. Player vectors: Characterizing soccer players' playing style from match event streams. volume 11908. Brefeld, U, Springer, 2020.

[2] Tom Decroos, Maaike Van Roy, and Jesse Davis. Soccermix: Representing soccer actions with mixture models. volume 12461, pages 459–474. Dong, Y, Springer, 2021.

[3] Guiliang Liu, Oliver Schulte, Pascal Poupart, Mike Rudd, and Mehrsan Javan. Learning agent representations for ice hockey. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[4] Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD International Conference on knowledge discovery data mining*, KDD '19, pages 1851–1861. ACM, 2019.

[5] Joseph Rocca. Understanding Variational Autoencoders (VAEs). https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, 2019.

[6] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9:2579–2625, 2008.

[7] IBM Cloud Education. Neural Networks. https://www.ibm.com/cloud/learn/neural-networks, 2020.

[8] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. 2013.

[9] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[10] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

[11] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013.

[12] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. 2016.

[13] Tom White. Sampling generative networks: Notes on a few effective techniques. *CoRR*, abs/1609.04468, 2016.

[14] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. PMID: 29532027.

[15] Weng, Lilian. From Autoencoder to Beta-VAE. https://lilianweng.github.io/posts/2018-08-12-vae/, 2018.

[16] Harri Valpola. From neural pca to deep unsupervised learning. 2014.

[17] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Morgan Kaufmann, San Francisco (CA), 1994.

[18] StatsBomb. https://statsbomb.com. Accessed: 3/5/2022.

[19] Stats Perform. https://www.statsperform.com/opta/. Accessed: 3/5/2022.

[20] Wyscout. https://wyscout.com. Accessed: 3/5/2022.

[21] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *NIPS*, 2015.

[22] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoRR*, 2015.

[23] Chao chao Yan, Sheng Wang, Jinyu Yang, Tingyang Xu, and Junzhou Huang. Re-balancing variational autoencoder loss for molecule sequence generation. *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2020.

[24] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating KL vanishing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 240–250, 2019.

[25] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, volume 28, pages 2980–2988, 2015.

[26] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[28] Tom Decroos. Soccer analytics meets artificial intelligence: Learning value and style from soccer event stream data, 2020.

[29] Sky Sports. Chelsea players, including Fran Kirby and Sam Kerr, dominate PFA WSL Team of the Year. https://www.skysports.com/football/news/28508/12324582/chelsea-players-including-fran-kirby-and-sam-kerr-dominate-pfa-wsl-team-of-the-year, 2021.

[30] BBC. Women's Ballon d'Or: Alexia Putellas wins award for being best female footballer in 2021. https://www.bbc.com/sport/football/59468815, 2021.

[31] BBC. PFA Player of the Year: Chelsea's Fran Kirby wins award for second time. https://www.bbc.com/sport/football/57373958, 2021.

[32] www.standard.co.uk. Chelsea FC pair Fran Kirby and Emma Hayes win WSL Player and Manager of the Season awards. https://www.standard.co.uk/sport/football/chelsea-fc-fran-kirby-emma-hayes-wsl-player-of-the-season-manager-b936957.html, 2021.

[33] Michael Caley. Premier League Projections and New Expected Goals. https://cartilagefreecaptain.sbnation.com/2015/10/19/9295905/premier-league-projections-and-new-expected-goals, 2015.