SciSports: Learning football kinematics through two-dimensional tracking data

Anatoliy Babic, Harshit Bansal, Gianluca Finocchio,
Julian Golak, Mark Peletier, Jim Portegies,
Clara Stegehuis, Anuj Tyagi, Roland Vincze,
William Weimin Yoo

Abstract

SciSports is a Dutch startup company specializing in football analytics. This paper is a joint collaboration with SciSports on learning and predicting football kinematics and trajectories. Using 2-dimensional positional data, we developed methods based on Newtonian mechanics and the Kalman filter, Generative Adversarial Nets and Variational Autoencoders. In addition, we trained a discriminator network to recognize and discern different movement patterns of players. We show that player's movements and velocities can be accurately predicted with quantifiable prediction uncertainties, and state-of-the-art procedures such as deep neural networks can learn movements effectively by discovering important latent features.

KEYWORDS: FOOTBALL, TRAJECTORY, NEWTONIAN MECHANICS, KALMAN FILTER, GENERATIVE ADVERSARIAL NETS, AUTOENCODER, DISCRIMINATOR

1 Introduction

SciSports (`http://www.scisports.com/`) is a Dutch sports analytics company taking a data-driven approach to football. The company conducts scouting activities of football clubs, gives advices to football players about which football club might suit them best, and quantifies the quality of football players through various performance metrics. So far, most of these activities have been supported by either coarse event data, such as outcomes and line-ups of matches, or more fine-grained event data such as completed passes, distances covered by players, yellow cards received and goals scored.

In a recent project called BallJames, SciSports installs specialized cameras and sensors across the football field to create 2-and 3-dimensional virtual rendering of the matches, by recording players' coordinate positions and gait data in high frequency milliseconds time intervals. From these massive amount of digital virtual maps, SciSports is interested in predicting future game course and in learning useful analytics from trajectory data. As these trajectories contain valuable information on the characteristics of the players and on strategies of teams, insights gained from this learning process can then be used as preliminary steps towards determining the quality of football players and searching for the best path towards scoring a goal.

Football kinematics such as accelerations, maximal sprinting speeds and distance covered during a match can be extracted automatically from trajectory data. However, there are also important unobservable factors/features determining the soccer game, e.g., a player can be of enormous value to a game without being anywhere near to the ball. These latent factors are key to understanding the drivers of motion and their roles in predicting future game states. There are in general two basic approaches to uncover these factors: we can either postulate a model or structure to these factors based on physical laws and other domain knowledge (model-based), or we use machine learning techniques and let the data discover these factors on its own (data-driven).

Model-based approaches have been widely used to analyze football trajectories. Examples in the literature include statistical models such as state space models Yu et al. (2003a,b); Ren et al. (2008) and physical models based on equation of motions and aerodynamics Goff and Carré (2009). These methods have the advantage of producing interpretable results and they can give reasonable predictions quickly using relatively few past observations. However as these methods are model-dependent, we might misspecify our modeling assumptions on physical movements, and we are not robust to outliers, i.e., players that have erratic or unexpected behaviors in the field.

Projects such as BallJames introduced above usher in massive amount of data streams, and we would like to take advantage of this through data-driven machine learning algorithms that are more robust to model misspecification. Following this line of thought, one route towards finding hidden factors would be to cast the football game into a reinforcement learning framework, cf. Sutton and Barto (1998). Naively speaking, one could define the state of a football game at some time t to be the collection of trajectories of all the players and the ball up to time t. In reinforcement learning, one would then try to learn a policy, i.e., learning which actions the players should make based on the current state of the game.

However, the state space of all possible player trajectories is enormous: it is a continuous and high-dimensional space. Therefore, it is impossible to try out all actions in all possible game states to derive a good policy. Instead one needs to first reduce the dimensionality of the game state before proceeding further.

One might wonder if instead of taking the whole collection of trajectories *up to* time t as the game state, one could use as game state the positions and velocities of the players *at* time t. However, the reinforcement learning framework is built around the so-called Markov Decision Processes, and it is crucial that the transition to the next state of the game only depends on the current state and the action taken. There is far too much memory in a football game for a state of positions and velocities to suffice. One could take into account variables such as current score, or time into the game, and other qualitative variables such as fatigue which are difficult to quantify for the purpose of numerical analyses.

In his PhD thesis Routley (2015) analyzed for which choice of game state the game of ice hockey can be interpreted as a Markov process based on trajectory data. Le et al. (2017) used a recurrent neural network for the generation and prediction of trajectories of football players, and they hard-coded the variables that are relevant in explaining the data.

In contrast, we prefer to follow an end-to-end approach, by taking the space of trajectories as a state space and let machine learning algorithms find an appropriate reduction of the state space automatically. There is a lot of spatio-temporal structure in the collection of trajectories of football players on the field, i.e., their motions are far from “random". These motions are obviously restricted by physical laws and human capabilities, and by which positions and walking lines make sense in a football game.

To capture the structure and statistics of football players's trajectories, we took a model-based approach using state space models, and data-driven approaches using deep generative networks. For the latter, we implemented a Variational Autoencoder (VAE), as introduced by Kingma and Welling (2013), and a Generative Adversarial Net (GAN) as developed in Goodfellow et al. (2014). Although different in formulation, all these approaches are closely related, as state space models can be viewed as shallow generative networks.

The state space models are built based on Newton's laws of motion, and the underlying latent state vectors are estimated using the Kalman filter. For the deep generative nets approaches, we started by learning the structure of trajectories of a single football player over a timespan of about fifty seconds. A similar approach for generating trajectories of basketball players, albeit using different encoding, was followed by Acuna (2017). After being able to accurately reproduce single trajectories with a VAE, we extended our method to generate trajectories of every player on the field plus the ball.

The VAE generated trajectories approximated real collections of trajectories of football players rather accurately. It was significantly harder to train the GAN, and the final generated trajectories were still easily distinguishable from the real trajectories.

The paper is organized as follows. In the next section, we introduce the 2-dimensional positional data used for our analyses. We present the statistical and machine learning methods we developed to make predictions and learn trajectories in Section 3. We introduce the discriminator network to differentiate movements in 3.4. We conclude in 4 and discuss future work.

2 Data

Let us impose a Cartesian (x, y)-coordinate system on the entire football field with the origin corresponding to the starting position of the ball in the center. As mentioned in the introduction, SciSports installed sensors to record the (x, y)-coordinate of all 22 players including the ball, and they are collected every 100 milliseconds for the entire duration of the match (which typically ranges from 90 to 95 minutes on extra time).

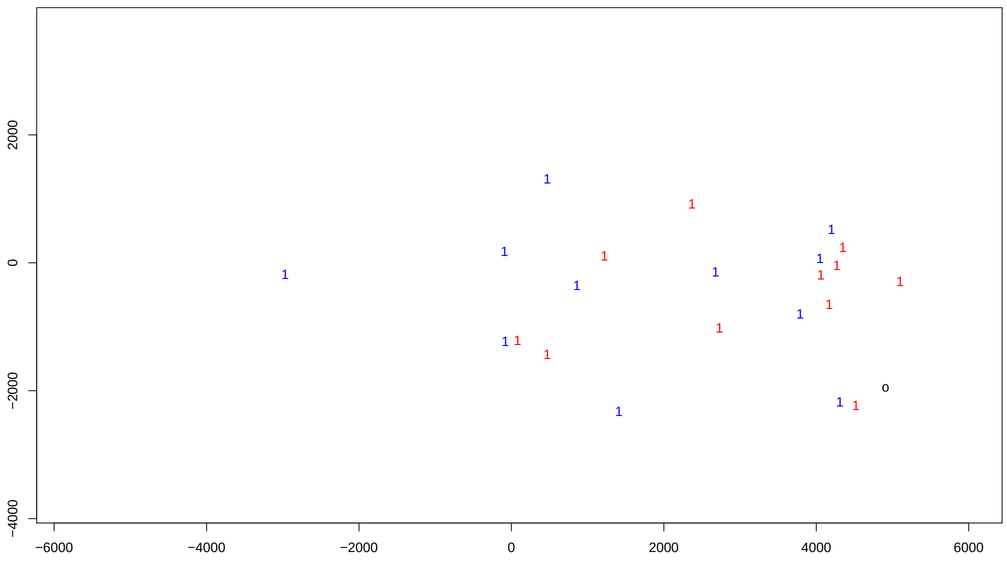


Figure 1: A snapshot in time (≈ 2 minutes into the game) of the positional data for all players (blue and red teams) and the ball (circle). Note the goalkeepers can be identified as the players standing at the leftmost and rightmost positions on the field.

We have these positional data for 14 matches and we have also basic information on the players, e.g., jersey number, roles and team membership. For illustration, Figure 1 shows a snapshot in time (around 2 minutes into the game) of the positional data for all players colored according to their respective red and blue teams, with 0 denoting the ball.

3 Methods

We consider statistical and machine learning techniques to model spatio-temporal trajectories of players and ball throughout the game in order to acquire meaningful insights on football kinematics. In Section 3.1, we model movements using Newtonian mechanics embedded in a state space framework, where player's position and velocity are estimated using Kalman filters. In Section 3.2, we use Generative Adversarial Networks to learn movements, such that two networks are pitted against each other to generate trajectories. Next in Section 3.3, we consider another class of networks called autoencoders, where we do data compression and train the network to replicate trajectories by learning important features. Finally, in Section 3.4 we investigate a method to discriminate between walking patterns of two different football players. The `R` and `Python` codes used to reproduce all our analyses can be found in `https://bitbucket.org/AnatoliyBabic/swi-scisports-2018`.

3.1 Newtonian Mechanics and the Kalman Filter

A single football player

For simplicity in exposition, let us first consider the case of modeling the movement of one football player in the first match. The same methods introduced in this section can be applied easily to the other 13 matches. We assume that this player is not a goalkeeper because we would like to model movement ranges that span at least half the field. As mentioned in the previous data section, SciSports installed sensors across the pitch to record a player's (x, y)-position every fixed $\Delta t = 100$ milliseconds as long as he/she remains playing for the game. Let us denote a player's position in the (x, y) plane at timestep t as $\boldsymbol{x}_t$ and let his/her velocity and acceleration at timestep t be $\boldsymbol{v}_t$ and $\boldsymbol{a}_t$ respectively. Now from Newtonian mechanics, we have the following relationships: $\boldsymbol{a}_t = d\boldsymbol{v}_t/dt$ and $\boldsymbol{v}_t = d\boldsymbol{x}_t/dt$. By solving these differential equations and rewriting their solutions in a recursive manner (with initial velocity as velocity at previous timestep), we obtain

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{x}_{t-1} + \Delta t \boldsymbol{v}_{t-1} + \frac{1}{2}(\Delta t)^2 \boldsymbol{a}_t \\ \boldsymbol{v}_t &= \boldsymbol{v}_{t-1} + \Delta t \boldsymbol{a}_t. \end{aligned} \tag{3.1}$$

From timestep $t-1$ to t, we assume that player moves at constant acceleration $\boldsymbol{a}_t$ that is normally distributed with mean $\boldsymbol{0}$ and unknown covariance matrix $\boldsymbol{Q}$. Using position and velocity $(\boldsymbol{x}_t, \boldsymbol{v}_t)$ as our latent state vector, let us now consider the following state space model:

$$\boldsymbol{y}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{\boldsymbol{W}_t} \underbrace{\begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{v}_t \end{pmatrix}}_{\boldsymbol{z}_t} + \boldsymbol{\varepsilon}_t, \tag{3.2}$$

$$\begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{v}_t \end{pmatrix} = \underbrace{\begin{pmatrix} \boldsymbol{I}_2 & \Delta t \boldsymbol{I}_2 \\ \boldsymbol{0} & \boldsymbol{I}_2 \end{pmatrix}}_{\boldsymbol{T}_t} \begin{pmatrix} \boldsymbol{x}_{t-1} \\ \boldsymbol{v}_{t-1} \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{1}{2}(\Delta t)^2 \boldsymbol{I}_2 \\ \Delta t \boldsymbol{I}_2 \end{pmatrix}}_{\boldsymbol{R}_t} \boldsymbol{a}_t \tag{3.3}$$

where $\boldsymbol{\varepsilon}_t \sim \mathrm{N}_2(\boldsymbol{0}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma} = \mathrm{Diag}(\sigma_x^2, \sigma_y^2)$ and $\boldsymbol{a}_t \sim \mathrm{N}_2(\boldsymbol{0}, \boldsymbol{Q})$. Here we write $\boldsymbol{I}_2$ to be the 2×2 identity matrix, $\boldsymbol{g} \sim \mathrm{N}_m(\boldsymbol{\mu}, \boldsymbol{\Omega})$ for a random variable $\boldsymbol{g}$ having a m-dimensional multivariate normal (Gaussian) distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Omega}$, and we denote $\mathrm{Diag}(\boldsymbol{b})$ to be the diagonal matrix with diagonal elements given by vector $\boldsymbol{b}$. We initialize $\boldsymbol{z}_1 \sim \mathrm{N}_2(\boldsymbol{0}, \boldsymbol{P}_1)$ and we assume that $\boldsymbol{\varepsilon}_t, \boldsymbol{a}_t, \boldsymbol{z}_1$ are mutually independent. In the state equation (3.3), the latent state vector $\boldsymbol{z}_t := (\boldsymbol{x}_t, \boldsymbol{v}_t)$ propagates forward in time according to Newtonian dynamics of (3.1). However in the observation equation (3.2), we only get to observe the player's position and not his/her velocity, and we assume that these position data are recorded using sensors with (Gaussian) measurement errors that is independent across time and (x, y)-directions.

To predict a player's movement and extract his/her velocity profile, we need to estimate the latent states $\boldsymbol{z}_t$ given noisy position data $\boldsymbol{y}_t$. To do this, we will use a Kalman filter to compute recursively the one-step state prediction $\boldsymbol{Z}_{t+1} = \mathrm{E}(\boldsymbol{z}_{t+1}|\boldsymbol{y}_t, \ldots, \boldsymbol{y}_1)$ and its error $\boldsymbol{\delta}_t = \boldsymbol{y}_t - \boldsymbol{W}_t\boldsymbol{Z}_t$, in conjunction with their estimated covariance matrices $\boldsymbol{P}_{t+1} = \mathrm{Var}(\boldsymbol{z}_{t+1}|\boldsymbol{y}_t, \ldots, \boldsymbol{y}_1)$ and $\boldsymbol{F}_t = \mathrm{Var}(\boldsymbol{\delta}_t) = \boldsymbol{W}_t\boldsymbol{P}_t\boldsymbol{W}_t + \boldsymbol{\Sigma}$. The exact Kalman recursion formulas for these calculations are given by (see Appendix A of Helske, 2017)

$$\begin{aligned}\boldsymbol{Z}_{t+1} &= \boldsymbol{T}_t(\boldsymbol{Z}_t + \boldsymbol{K}_t\boldsymbol{F}_t^{-1}\boldsymbol{\delta}_t)\\ \boldsymbol{P}_{t+1} &= \boldsymbol{T}_t(\boldsymbol{P}_t - \boldsymbol{K}_t\boldsymbol{F}_t^{-1}\boldsymbol{K}_t)\boldsymbol{T}_t^T + \boldsymbol{R}_t\boldsymbol{Q}\boldsymbol{R}_t,\end{aligned}$$

where $\boldsymbol{K}_t = \boldsymbol{P}_t\boldsymbol{W}_t^T$.

We have a total of 6 unknown parameters in our state space model, i.e., the two diagonal entries of $\boldsymbol{\Sigma}$ and all the 2×2 entries of $\boldsymbol{Q}$. Note here that these parameters do not depend on the timestep t and this is assumed to keep computations manageable, as the total recorded timestep n for a typical player in our positional data runs up to 57013 or 5701300 milliseconds (≈ 95 minutes with extra time). The Kalman filter equations above is then used to calculate the log-likelihood function, and it is given by

$$l_n = -\frac{np}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\left(\log\det\boldsymbol{F}_t + \boldsymbol{\delta}_t^T\boldsymbol{F}_t\boldsymbol{\delta}_t\right),$$

where p is the dimension of $\boldsymbol{y}_t$ at a fixed t, which in our present case is 2. We then compute the maximum likelihood estimator for the 6 covariance parameters using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm.

Remark 1. In actual implementations, some technical modifications are needed to speed up computations, particularly when $\boldsymbol{y}_t$ consists of high-dimensional observations at each time point. To solve for this dimensionality issue and to avoid direct inversion of $\boldsymbol{F}_t$, the state space model of (3.2) and (3.3) is recast into an equivalent univariate form and the latent states are estimated using a univariate Kalman filter (cf. Koopman and Durbin, 2000).

Remark 2. Concerning the choice of $\boldsymbol{P}_1$ in the initialization of $\boldsymbol{z}_1$, a commonly used default is to set $\boldsymbol{P}_1 = 10^7\boldsymbol{I}$ as a diffuse prior information. However, this is numerically unstable and prone to cumulative roundoff errors. Instead, we use the exact diffuse initialization method by decomposing $\boldsymbol{P}_1$ into its diffusive and non-diffusive parts, for more details see Koopman and Durbin (2003).

The Kalman filter algorithm and parameter estimation (including the univariate formulation and diffuse initialization) were performed using the `KFAS` package (see Helske, 2017) in the `R` software package.

Results

We modeled the movement of player with Jersey number 3 holding the position of left central midfielder. This midfielder was in the pitch for the entire game which lasted up to 5701300 milliseconds or around 95.022 minutes. Our aim here is to predict his/her future location and speed (including direction of movement), and also to quantify prediction uncertainties. Moreover, we would like to achieve these goals in an online manner such that we can update our estimates rapidly as new data is made available. We show below that the state space model we developed through Newtonian mechanics can simultaneously achieve these aims. In all the results below, we use a sliding window of 10 training samples for predictions, such that we first use 10 time points to predict the 11th point (one-step-ahead), then we move a timestep ahead and use the next 10 time points to predict the 12th point and so on.

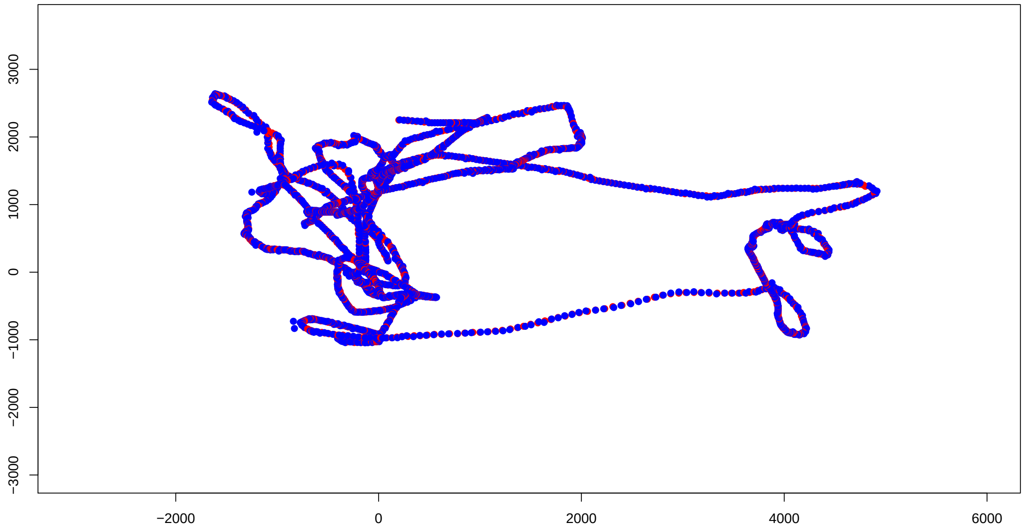


Figure 2: Blue: One-step-ahead predicted position, Red: True recorded position.

Figure 2 shows one-step-ahead predicted position of our midfielder (blue dots) for the first 2500 time points. We see that the state space model is able to make accurate predictions (when compared to the red true positions), even if we have used only the past 10 locations in our algorithm. Moreover, the model is able to trace out complicated movements and sharp corners as evident from the figure. In addition, one main advantage of our modeling approach is the ability to extract and predict the midfielder's velocity, a quantity which is not recorded but had to be deduced from position data. In strictly physical sense, velocity $\boldsymbol{v}_t = (v_{x,t}, v_{y,t})$ is a vector in $\mathbb{R}^2$ for our present situation, with its arrow pointing towards the direction of motion and its length encoding speed (a scalar quantity). The Kalman filter gives us estimates of the latent states and hence velocities. We attached these estimated velocity vectors to each predicted position to yield the predicted velocity vector field of Figure 3. By following the flow of these vectors, we can trace out the line of motion for this midfielder and determine his/her speed at a particular position in the field, with

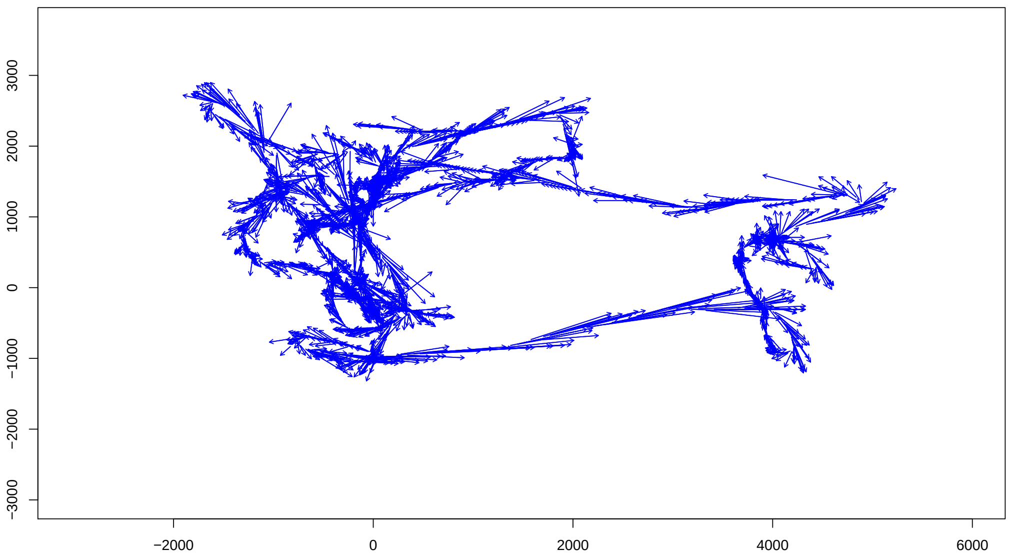


Figure 3: One-step-ahead predicted velocity vector field $\boldsymbol{v}_t$, arrow points to direction of motion and vector length is speed.

longer arrows indicating faster speed. We plotted these speeds $\|\boldsymbol{v}_t\|$ against the 2500 time points in Figure 4.

To see the correspondence between these three figures, let us focus on a distinguishing stretch of movement made by our midfielder, namely he/she starts at $(0, -1000)$, then sprints towards the goal post in the East, make two loops towards the North and again moved back across the field to the West, thus making a somewhat elongated rectangle on the field. We know that he/she is sprinting to the goal from Figure 3 due to the long arrows pointing to the East, with exact magnitudes given by the peak slightly after time 1000 in Figure 4. The midfielder has relatively lower speeds when making the double loop (from time 1200 to 1500 in Figure 4) and then he/she picks up the momentum when moving towards the West, as evident in the marked increase in speeds after time 1500.

Figure 5 shows the predictive performance of our model for longer time horizons, in this case, we are using 10 time points to predict 5-steps ahead. When compared with the one-step-ahead case of Figure 2, we see that there is some deterioration in our model's predictive capability, particularly for places when the midfielder's trajectory is curved. However the 5-step ahead predictions generally track the overall true movement sufficiently well, and their uncertainties are encoded inside the blue 95%-prediction rectangles (note: straight lines are an artefact of the algorithm). From this plot, we can deduce that positional uncertainties are the greatest when the midfielder is moving in loops or in circles.

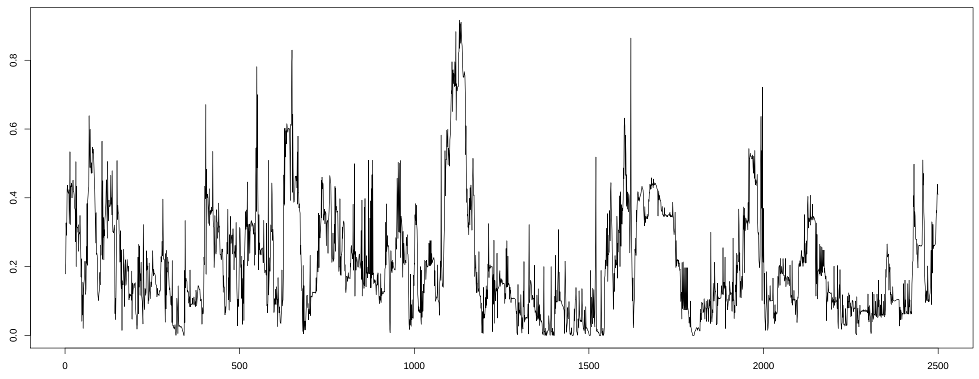


Figure 4: One-step-ahead predicted speed $\|\boldsymbol{v}_t\|$ (y-axis) against timesteps (x-axis).

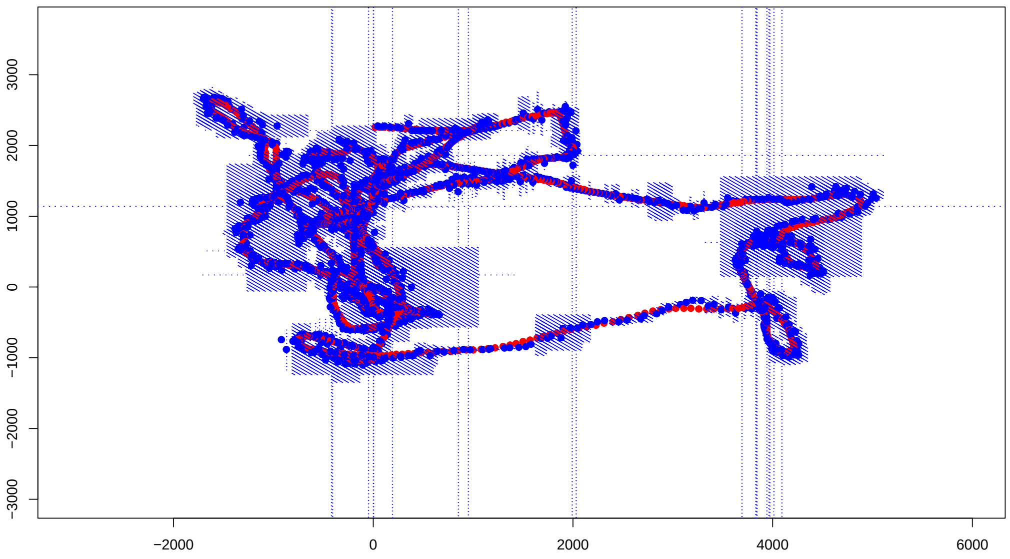


Figure 5: Blue dots: 5-step-ahead predicted position, Blue squares: 95%-prediction rectangle, Red dots: True recorded position.

The ball and all 22 football players

Let us now consider the general case of modeling all 22 football players including goalkeepers and the ball itself. A snapshot of the positional data at around 2 minutes into the game is shown in Figure 1. We can impose the same Newtonian mechanics to all players giving for all $k = 1, \dots, 23$,

$$\begin{aligned} \boldsymbol{x}_t^{(k)} &= \boldsymbol{x}_{t-1}^{(k)} + \Delta t \boldsymbol{v}_{t-1}^{(k)} + \frac{1}{2}(\Delta t)^2 \boldsymbol{a}_t^{(k)} \\ \boldsymbol{v}_t^{(k)} &= \boldsymbol{v}_{t-1}^{(k)} + \Delta t \boldsymbol{a}_t^{(k)}. \end{aligned} \tag{3.4}$$

By stacking up 23 copies of the single player case (3.2) and (3.3), we can convert the equations of motion above to the following state space model:

$$\boldsymbol{y}_t = \begin{pmatrix} \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_2 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{I}_2 & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_t^{(1)} \\ \boldsymbol{v}_t^{(1)} \\ \boldsymbol{x}_t^{(2)} \\ \boldsymbol{v}_t^{(2)} \\ \vdots \\ \boldsymbol{x}_t^{(23)} \\ \boldsymbol{v}_t^{(23)} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\varepsilon}_t^{(1)} \\ \boldsymbol{\varepsilon}_t^{(2)} \\ \vdots \\ \boldsymbol{\varepsilon}_t^{(23)} \end{pmatrix},$$

and the underlying latent state vectors are

$$\begin{pmatrix} \boldsymbol{x}_t^{(1)} \\ \boldsymbol{v}_t^{(1)} \\ \boldsymbol{x}_t^{(2)} \\ \boldsymbol{v}_t^{(2)} \\ \vdots \\ \boldsymbol{x}_t^{(23)} \\ \boldsymbol{v}_t^{(23)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_2 & \Delta t \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I}_2 & \Delta t \boldsymbol{I}_2 & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_2 & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{I}_2 & \Delta t \boldsymbol{I}_2 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{I}_2 \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_{t-1}^{(1)} \\ \boldsymbol{v}_{t-1}^{(1)} \\ \boldsymbol{x}_{t-1}^{(2)} \\ \boldsymbol{v}_{t-1}^{(2)} \\ \vdots \\ \boldsymbol{x}_{t-1}^{(23)} \\ \boldsymbol{v}_{t-1}^{(23)} \end{pmatrix}$$
$$+ \begin{pmatrix} \frac{1}{2}(\Delta t)^2 \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \Delta t \boldsymbol{I}_2 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}(\Delta t)^2 \boldsymbol{I}_2 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Delta t \boldsymbol{I}_2 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \frac{1}{2}(\Delta t)^2 \boldsymbol{I}_2 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \Delta t \boldsymbol{I}_2 \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_t^{(1)} \\ \boldsymbol{a}_t^{(2)} \\ \vdots \\ \boldsymbol{a}_t^{(23)} \end{pmatrix},$$

where the measurement error vector is $(\boldsymbol{\varepsilon}_t^{(1)} \quad \boldsymbol{\varepsilon}_t^{(2)} \quad \cdots \quad \boldsymbol{\varepsilon}_t^{(23)}) \sim \mathrm{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma} = \mathrm{Diag}(\sigma_{x,1}^2, \sigma_{y,1}^2, \sigma_{x,2}^2, \sigma_{y,2}^2, \dots, \sigma_{x,23}^2, \sigma_{y,23}^2)$ and the acceleration vector $(\boldsymbol{a}_t^{(1)} \cdots \boldsymbol{a}_t^{(23)}) \sim \mathrm{N}(\mathbf{0}, \boldsymbol{Q})$. Here the covariance matrix $\boldsymbol{Q}$ is crucial to model complicated movement interactions between football players and the ball. An unstructured $\boldsymbol{Q}$ consists of $46^2 = 2116$ parameters and adding the diagonal elements of $\boldsymbol{\Sigma}$ yields a total of 2162 parameters. We found that this general case takes prohibitively long to optimize, and we have to simplify the problem by imposing additional structure on $\boldsymbol{Q}$. To keep computations manageable, we start by assuming that $\boldsymbol{Q}$ is a block diagonal matrix given by $\boldsymbol{Q} = \text{BlockDiag}(\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_{23})$ where $\boldsymbol{Q}_k = \text{Var}(\boldsymbol{a}_t^{(k)})$ for $k = 1, \ldots, 23$. In other words, each player's movement is modeled using his/her own state space equations that are independent of the other players.

If the prediction horizon is short, e.g., one-step-ahead, we found that this choice of $\boldsymbol{Q}$ gives reasonable predictive performance as shown in Figure 6. Here we have used 5 past time points to predict one timestep ahead and we see that the one-step-ahead predicted player's position (blue) closely follows the truth (red) over the span of 206 time points. Moreover, the path of the ball is instantly recognizable as the zig-zag dotted line (due to it being the fastest object) embedded among the network of trajectories. If longer prediction horizons are sought, then this simplifying assumption might not give good performance and cross-covariance terms between players and ball are needed. To that end, one can consider low rank approximations or imposing sparsity constraints on $\boldsymbol{Q}$. Alternatively, we can turn to machine learning methods by training a (deep) multi-level neural network to learn these complex interactions, and we shall devote subsequent sections to explore this approach.

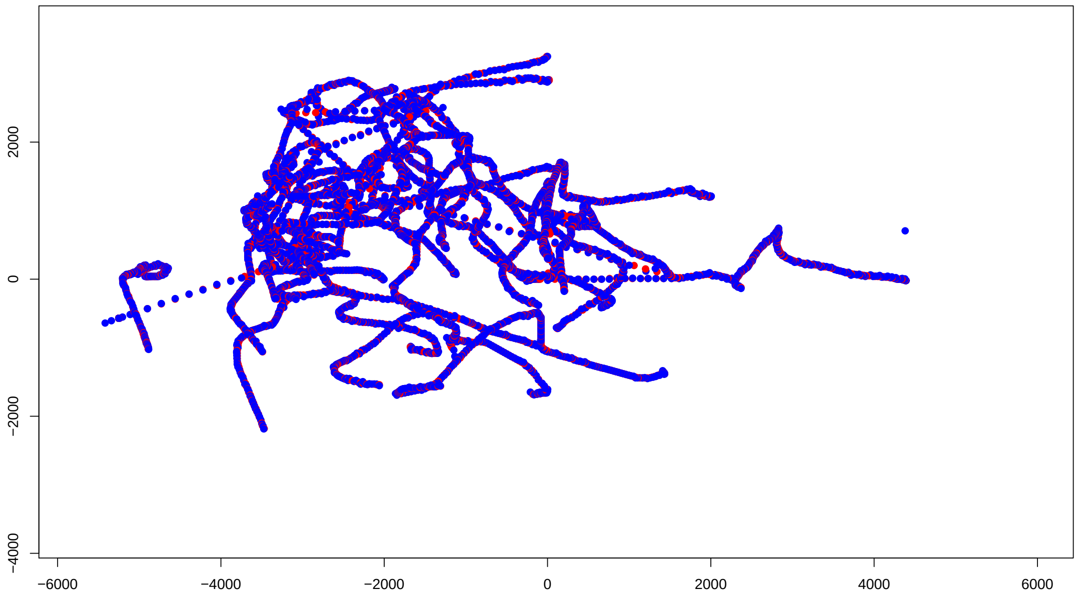


Figure 6: One-step-ahead predicted position for all 22 players (blue) with their true paths (red). The path of the ball is the zig-zag dotted line.

3.2 Generative Adversarial Network

Generative Adversarial Networks (GANs) are deep neural net architectures introduced by Goodfellow et al. (2014) which exploit the competition between two (adversarial) networks: a generative network called the Generator and a discriminative network called the Discriminator.

Both the Generator and Discriminator are trained with a training set of real observations, and against each other. The Discriminator is a classifier, it has to learn to differentiate between real and generated observations, labeling them as "realistic" and "fake" respectively. The Generator, on the other hand, has to learn to reproduce features of the real data and generate new observations which are good enough to fool the Discriminator into labeling them as "realistic".

GANs have been used with great success in image recognition, 3D-models reconstruction and photorealistic imaging, e.g. Karazeev (2017).

2D positional data into images

The data consists of 14 complete 90-minutes matches of (x, y)-coordinates for players and the ball (23 entities total) with a resolution of 10 cm and 10 frames per second, i.e., the trajectory of a player on a 10 seconds timespan corresponds to a (2×100)-vector of (x, y)-coordinates. Furthermore, there is an intrinsic difference in scale between the x and y coordinates due to the rectangular shape of the field, in our specific case a 68×104 m^2 field resulting in a 6800×10400 rectangle of possible (x, y)-coordinates. Under these circumstances, the image given by the trajectory of a single player on the pitch over a 10 seconds timespan, is a 6800×10400 image which is everywhere white (background color) with the exception of at most 100 pixels (the trace of the player).

Considering that state-of-art neural networks require a huge amount of resources to deal with 128×128 images, there is no hope in tackling the problem of trajectory generation without reducing the resolution of the image and sacrificing complexity for the sake of tractability. A work in a similar direction is Acuna (2017), in which images of basketball players and ball trajectories are used to train a Variational Autoencoders (VAEs), see Section 3.3.

We approach the problem differently, we treat the sequence of (x, y)-coordinates given by a trajectory as an image itself. By rescaling the data accordingly we can map the football field in the square $[-1, 1]^2$ and interpret a 10 seconds trajectory as a 2×100 gray-scale image, which we can input to the neural network machinery. Of course there is no need to think of this vector of normalized coordinates as an image, but this comparison will be useful in what follows.

Network setup

The algorithm we use is a repurposed version of the basic convolutional neural network found at Bruner and Deshpande (2017), which is meant to recognize and reproduce handwritten digits. There is a structural difference between the two:

- the original algorithm works with 28×28 black-and-white images of 10 possible states, i.e. the digits 0-9;
- our algorithm works with 2×52 gray-scale images, containing an aggregation of 20 seconds of play, with no upper bound on the number of states given that we are not classifying the trajectories by their topological properties.

Working with black-and-white images means that the information we are interested in is binary, either ON or OFF. These images are rough and have spiky edges, and every gray shade is pushed either towards black or white. With full gray-scale images however, we cannot afford to loose details by filtering out the gray areas, player movements produce very smooth transitions (between black and white) and this regularity is the major feature the network has to learn.

Another important difference is in the intrinsic asymmetry of the data:

- in the original version, both the Discriminator and the Generator look at 3×3 or 5×5 spatial features of the images: useful information about the topology of the shape can be obtained by looking at spatial neighborhoods of any given pixel;
- in our case we want to look a the x and y coordinates independently, therefore our Discriminator and Generator work with 1×5 and 1×10 temporal features: the information regarding the trajectory is contained in a temporal neighborhood of each position, i.e., its recent past and future.

By making this tweak to the original algorithm we exploit the natural directionality of the data and we avoid overlapping the spatial properties (i.e., the shade of gray) and the temporal properties (i.e., the variation in shade). To have a sense of what this means we visualize the correspondence between the (x, y)-coordinates and the real trajectory of a player, see Figure 7.

The algorithm

We limit our training set to all random samplings of 20 seconds trajectories of any single player (not goalkeepers nor the ball) during a single fixed match. This should give some extra structure for the network to work with while maintaining a diverse enough data sample.

The initialization of the parameters is the same as in the original algorithm, the Generator takes a standard Gaussian noise vector as input and then produces a new image based of the updates made by the network. To have a glance of what an untrained Generator is capable of, see Figure 8.

The Discriminator is then pre-trained with real trajectories and generated ones. After this first training epoch, the Discriminator is able to correctly discriminate between the real trajectories and the untrained noisy ones produced by the Generator. Here an epoch consists of one full learning cycle on the training set. Then the main training session begins. From the second epoch and above, the Discriminator is

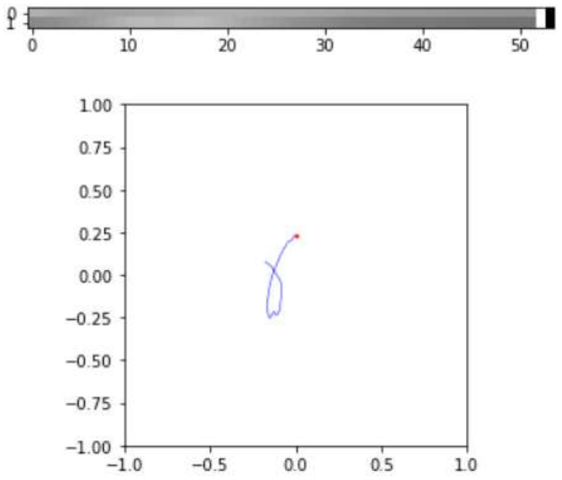


Figure 7: A non-trivial real trajectory and its twofold representation. The (x, y)-coordinates as gray-scale image (top) and the real trajectory on the rescaled football field (bottom).

trained with real and generated data and the Generator itself is trained against the Discriminator. This produces a Generator-Discriminator feedback loop that forces both networks to improve themselves with the objective to outperform the other. This is achieved by implementing a loss function to measure three quantities:

- Discriminator loss vs real: it measures how far the Discriminator is from labeling a real trajectory as "realistic";
- Discriminator loss vs Generator: it measures how far the Discriminator is from labeling a generated image as "fake";
- Generator loss vs Discriminator: it measures how far the Discriminator is from labeling "a generated image as "realistic".

The first loss function deals with the interaction between the Discriminator and the real world, it makes sure that the network is adapting to recognize new real observations. The second and third loss functions on the other hand, work against each other: one is trying to force the Discriminator to always label "fake" when presented with a generated image, while the other is forcing the Generator to produce data that mimics the Discriminator's perception of the real world. The loss function used throughout the algorithm is the cross-entropy loss, for a discussion see Seita (2017).

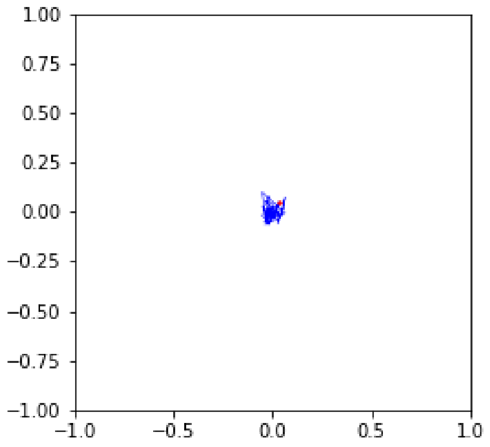


Figure 8: A trajectory from the untrained Generator.

Performance and limitations

Properly training a GAN requires a long time and a lot can go wrong in the process. The Generator and Discriminator need to maintain a perfect balance, otherwise one will outperform the other causing either the Discriminator to blindly reject any generated image, or the Generator to exploit blind spots the Discriminator may have. After a training session of 15 hours our GAN managed to go from random noise trajectories to smooth and structured ones, although not fully learning the underlying structure of the data. While the generated movements look impressive when compared to the untrained ones, they are still underperforming when confronted with the real world. First and foremost, the acceleration of the players make no physical sense, i.e., the algorithm is not able to filter out local small noise, and the trajectories are not smooth enough. The evolution of the network during training is shown in Figure 9. In the end the GAN is not consistent enough when asked to generate large samples of data, and after long training sessions, the average trajectory looked like the bottom-left one in Figure 9.

3.3 Autoencoder (and its variants)

The limitations of Generative Adversarial Networks were discussed in Section 3.2. This then led us to consider another class of networks popularly called Autoencoders (and its variants) to cope with the challenges of long training times and training instabilities. Similar to GAN, Autoencoder (AE) is an unsupervised machine learning technique that generates its own labels from the training data and use them to learn

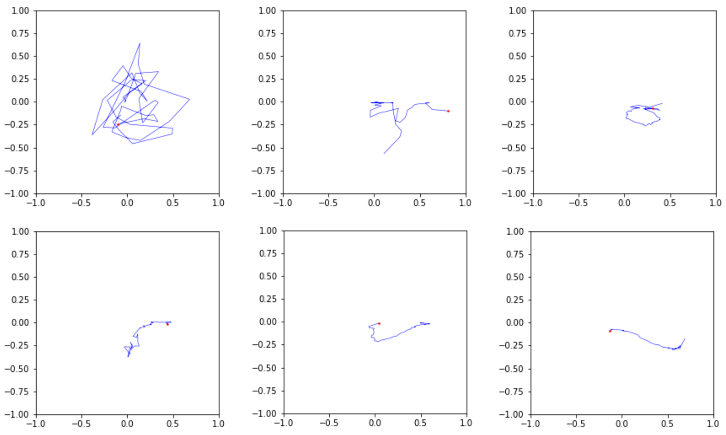


Figure 9: Different stages of GAN training. The network goes from random noise to shape recovery, but it is not able to filter out local noise consistently.

underlying hidden feature representations. AE is an automatic feature engineering, where compression and decompression is learned directly from the data. It ensures that the hidden representation maintains all the information of the input. This approach closely resembles Principal Component Analysis (PCA). Unlike PCA, which is a linear transformation, the nonlinear activation functions used in Autoencoders can encode non-linear mapping. Encoding this non-linear mapping is essential in the context of the problem at hand. Encoder and decoder serve as two main ingredients of Autoencoder-based learning. The decoder network is the mirror image of the encoder network. The reconstructed output is degraded compared to the input data. Loss functions are then required to quantify the difference between the actual input and the reconstructed output, with the mean squared error being the most widely used.

Our interest is to generate more data with similar features to the input dataset. A typical or conventional Autoencoder does not work well for such scenario and hence we explored another version of Autoencoder, called the Variational Autoencoder (VAE) introduced by Kingma and Welling (2013). VAE retains the Autoencoder architecture but uses Variational Bayes for parameters/hyperparameters estimation. This approach is a combination of Bayesian inference and a generative network. Specifically, the architecture of VAE is composed of an encoder and a decoder. Here, the encoder is a variational inference network which maps observed inputs to posterior distributions over the latent space and the decoder is a generative model which is capable of mapping arbitrary latent coordinates to distributions over the original data space. The encoder and decoder networks are probabilistic. The network bottleneck comes from sampling stochasticity rather than dimensionality reduction. The parameters of the encoder and decoder are the weights and biases of neural networks. AE and VAE are different only due to the constraint applied on the latent space in their formalisms. Another important thing to take into consideration is that the mean squared error is not a good error metric when dealing with distributions as in VAE. The Kullback-Leibler (KL) Divergence is used to measure the similarities/ differences between two distributions, and hence is employed as the loss function in VAE. This is what differentiates a VAE from a conventional Autoencoder, where the latter relies only on the reconstruction cost.

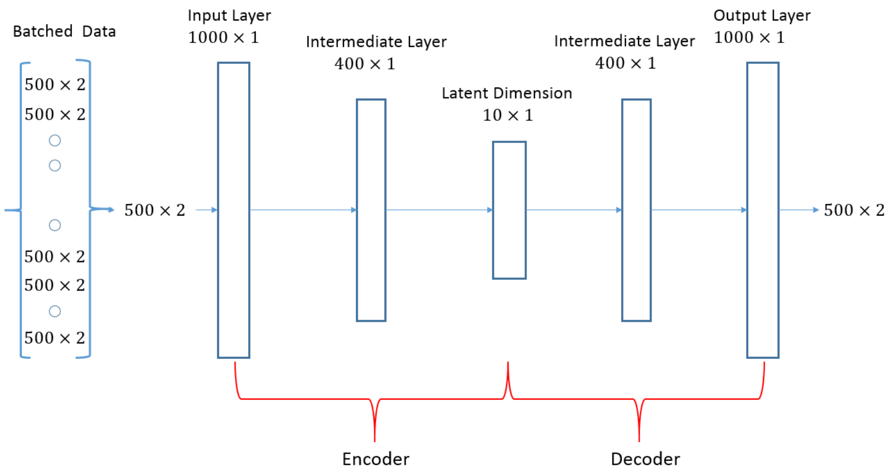


Figure 10: Complete system architecture.

Estimating the trajectories of the players and the ball is critical from the point of view of the game. Hence, we generate dynamic sequences to reflect the tendency of player(s) and the trajectory of the ball. We use Keras (Chollet et al., 2015) to conduct several numerical experiments. Keras is a neural network library written in Python and has a number of pre-built layers. Keras has support for Tensorflow and Theano implementations. We use Keras on top of Tensorflow, where the actual computations are performed. Tensorflow is a lower level library which facilitates the back-propagation calculation. Instead of writing a Python module from scratch, we use transfer learning by adapting an existing VAE implementation on the MNIST black-and-white handwritten digit dataset to our present football trajectories. This VAE implementation has two terms that contribute to the objective function. As we have continuous data which is unlike the binary data in the MNIST dataset, this prompted us to reformulate our loss function. We used the same objective function as for the MNIST example and then studied the contributing terms in the context of football trajectory generation. We found that the KL term has an insiginificant contribution and hence we compute our loss based on the dominating term, which is essentially equivalent to computing the loss based on the "mean squared" metric. Therefore, this VAE implementation is reduced to a quasi-AE by this simplification, in order to avoid using the binary cross entropy as loss (used in the aforementioned VAE implementation), as it is not an appropriate loss for a continuous dataset.

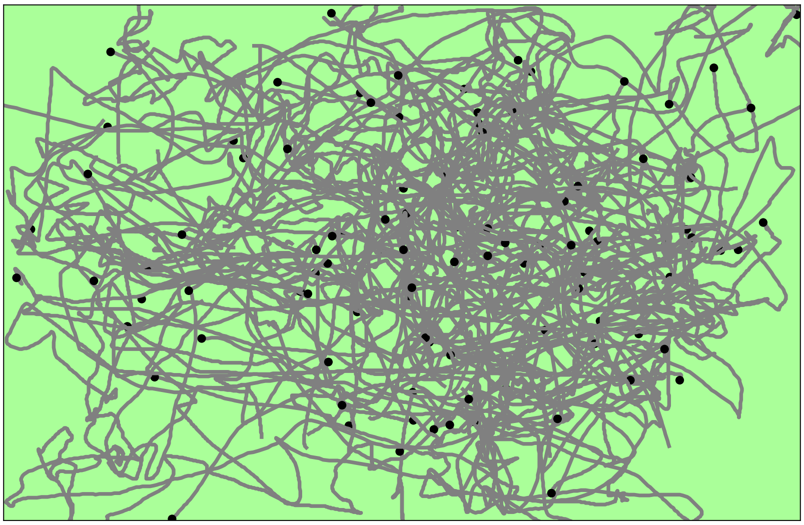


Figure 11: Original (a single player) trajectories of 100 test samples.

The resulting network architecture used to tackle our problem of trajectory generation is depicted in Figure 10. We use ReLu activation in the hidden layers and sigmoid activation in the input and the output layers. The weights of the neural network are updated after every epoch. We use 100 epochs for our numerical simulations, and utilize the "rmsprop" optimizer in its default settings. We reshape our data at several steps. The dimension of the input and the output layers of the network is 1000. The dimension of the hidden (intermediate) layers is chosen as 400. The dimension of the encoder output corresponds to the dimension of the latent space (chosen as 10 for numerical experiments). It is clear that we do not have any hidden layer, which is of dimension smaller than the encoder output. Notice that the encoder and decoder architectures are symmetric.

We use the data from 14 matches and divide it appropriately into batches. We then feed these batches into our neural network for training. The network is trained on a 10-dimensional (10D) latent space. Latent variables can be thought of as a compact, high level representation of the players' attributes, such as dribbling and running ability with the ball, spatial awareness, agility, balance, coordination etc. These variables can also be thought of as characteristics that completely capture the structure and evolution of the game. We generate new trajectories by moving in the 10D latent space and thereby explore evolution under perturbation of defining

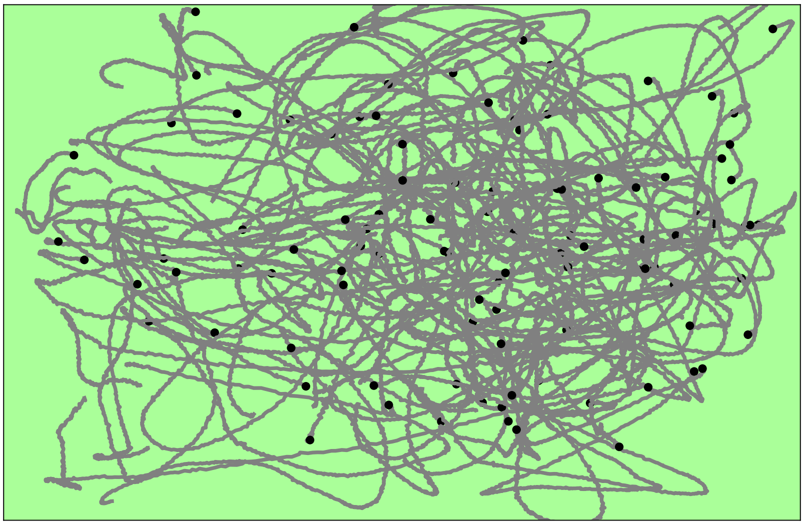


Figure 12: Generated (a single player) trajectories with latent samples encoded from original.

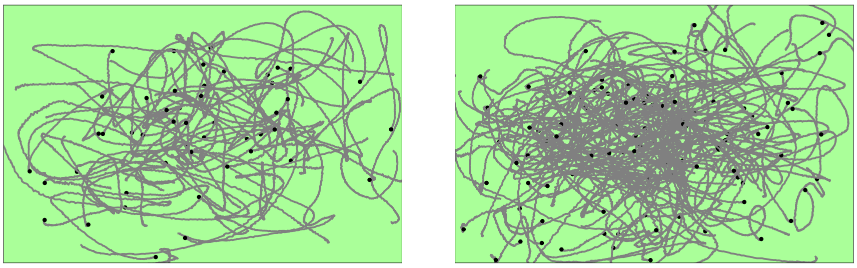


Figure 13: (left) Decoder is used on 50 latent samples drawn from $N(\mu, \sigma^2)$ and (right) Decoder is used on 100 latent samples drawn from $N(\mu, \sigma^2)$.

attributes. We sample points in the 10D representation space and move in various directions to see if we get some noticeable change in the decoded trajectories or just noise. Before we show the trajectories generated (from the decoder) with the random samples from the latent space, we draw the original trajectories (of a single player) in Figure 11. Figure 12 represents the decoded trajectories of the player from

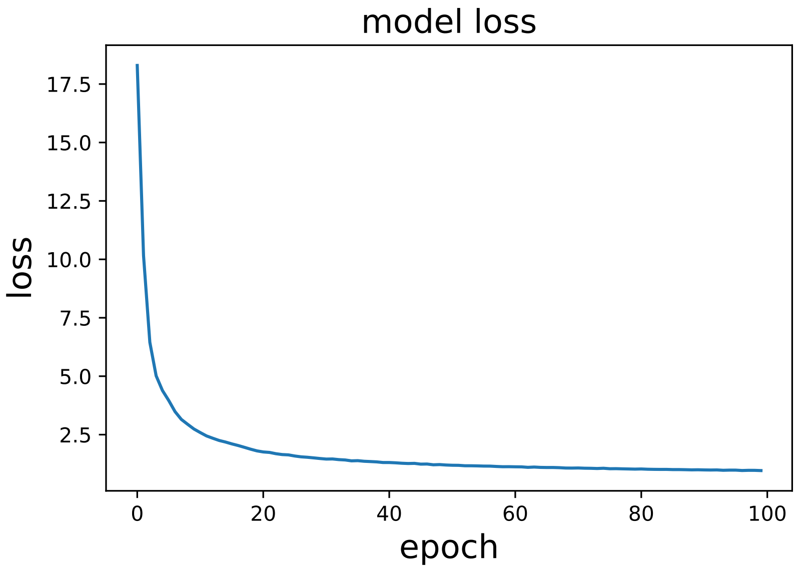
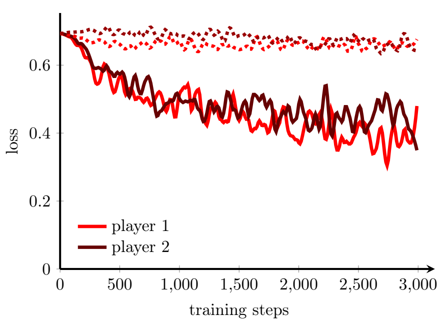


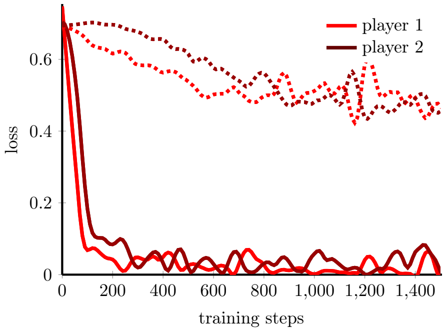
Figure 14: Evolution of training loss with each epoch.

the latent variables encoded from the originals. We now estimate μ and σ of the latent distribution from the original dataset and use this knowledge in drawing the samples, which are then passed through the decoder. Figure 13 shows the single player trajectories generated by the decoder with random samples drawn from the latent space, i.e., from the distribution $N(\mu, \sigma^2)$. These results demonstrate that this 10D representation of the data is a (heuristically) good coordinate system for the subspace of the data that is actually meaningful. It should be mentioned that the latent space dimension (here 10D) was chosen heuristically and is in no way the most optimal choice. Additionally, there are ranges of hyperparameters in neural networks where one can tune to achieve optimal results. Figure 14 demonstrates that the training error is quite low after certain number of epochs. It is worth mentioning that the results reported in this section use non-centered data, where the starting position of a players movement does not have to be $(0, 0)$. This non-centered data was also pre-processed (and aggregated) to avoid generating rapid zig-zag movements. This was an important step to obtain smooth trajectories. However, we did not investigate the behavior of the VAE for centered datasets.

Unlike GAN, training an Autoencoder is not a time consuming process. The neural network architecture of an Autoencoder seems to perform well as evident from the trajectories obtained from the trained Autoencoder and visualized in the figures above. These generated trajectories make physical sense and look indistinguishable from original or real trajectories, and we conclude that the VAE (or rather its quasi-AE version) does a decent job and is well suited to robustly learn the trajectories of



(a) example 1



(b) example 2

Figure 15: Two examples of the Discriminator loss function for both players as a function of the number of training steps. The solid lines are the results for uncentered data and the dashed lines contain the results for the centered data. The two examples contain four different players.

the players and the ball.

3.4 Discriminator

In the previous section, we studied several methods to predict and generate movement trajectories of soccer players. In this section, we study to what extent movement trajectories of different soccer players can be distinguished. To this end, we test the Discriminator network of the GAN introduced in Section 3.2 on data of different soccer players. We train the Discriminator on the data of two soccer players, and then test if the Discriminator is able to distinguish their motion patterns. The success rate of the Discriminator to distinguish one player from the other then gives some insight in how different the movement behaviors of two different players are.

The loss function for the Discriminator is the same as in Section 3.2. The data we use as input for the Discriminator are (x, y)-coordinates of 10-second player trajectories. We test the Discriminator on these unedited (x, y)-trajectories, and on centered (x, y)-trajectories, where the coordinates of each trajectory are centered such that the first coordinate always equals $(0, 0)$. Thus, by using the uncentered data, the Discriminator may distinguish two players by using their positions on the field, whereas the Discriminator can only use movement patterns of particular players when the centered data are used.

Figure 15 shows the Discriminator loss function for both players as a function of the number of training steps for two different sets of two players. We see that the loss function declines more for the uncentered data than for the centered data. Thus, the Discriminator distinguishes uncentered trajectories based on the location on the field where movement pattern happens. The two different examples also show that it is easier to distinguish some players than others. Table 1 shows the success rate of correctly identifying the player corresponding to a given trajectory after the training period for the two sets of players of Figure 15. The success rate of the Discriminator using the uncentered data is higher than for the centered data in both examples. Using the centered data, the Discriminator has difficulties distinguishing between players 1 and 2 in the first example. In the second example, the success rate is much higher. Thus, some players display more similarities in their movement patterns than other players.

		Player 1	**Player 2**
example 1	non-centered	0.74	0.9
	centered	0.2	0.96
example 2	non-centered	0.98	0.82
	centered	0.54	0.95

Table 1: The success rate of the Discriminator after training on the two examples of Figure 15.

4 Conclusion and future work

For short term online predictions, the state space modeling approach is able to give accurate position and velocity predictions. However for longer time horizons, more sophisticated data-driven methods might be needed, and our research shows that autoencoders seem to perform particularly well. All of our computations were performed using standard CPUs, and this imposed limitations in computational power when training our models (particularly GANs). Moreover as predictions are needed rapidly in actual football matches, we need a significant boost in computation speed and this calls for the use of GPUs for training.

As state space models give interpretable outcomes such as velocity vector fields and speeds, it would be interesting to explore the idea of embedding these models in (deep) neural networks, e.g., using recurrent neural nets to model latent state evolution.

The autoencoders were trained using spatio-temporal data from all 14 matches and it would be interesting to see how they can generalize to a new game. Moreover, we would like to investigate what restrictions on the game states would still allow for accurate reproduction of the trajectories. Additionally, we would like to add more structures to the architecture of the Variational Autoencoder, so that we are able to encode the differences between players. This would then give us insights into the different qualities of football players, and to determine which quality is crucial to winning games. By continuing this line of work, we could conceivably find an appropriate state space such that the football game can be fitted into a Reinforcement Learning framework that is practical to work with, and at the same time allows us to formulate optimal policies and strategies.

References

D. Acuna. Unsupervised modeling of the movement of basketball players using a deep generative model. 2017. Technical Report.

J. Bruner and A. Deshpande. Generative adversarial networks for beginners. Retrieved from `https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners`, 2017.

F. Chollet et al. Keras. `https://keras.io`, 2015.

J. E. Goff and M. J. Carré. Trajectory analysis of a soccer ball. *American Journal of Physics*, 77(11):1020–1027, 2009.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

J. Helske. KFAS: Exponential family state space models in R. *Journal of Statistical Software*, 78(10):1–39, 2017.

A. Karazeev. Generative adversarial networks (GANs): Engine and applications. Retrieved from `https://blog.statsbot.co/generative-adversarial-networks-gans-engine-and-applications-f96291965b47`, 2017.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. 2013. URL `http://arxiv.org/abs/1312.6114`. arXiv: 1312.6114.

S. J. Koopman and J. Durbin. Fast filtering and smoothing for multivariate state space models. *Journal of Time Series Analysis*, 21(3):281–296, 2000.

S. J. Koopman and J. Durbin. Filtering and smoothing of state vector for diffuse state-space models. *Journal of Time Series Analysis*, 24(1):85–98, 2003.

H. M. Le, P. Carr, Y. Yue, and P. Lucey. Data-driven ghosting using deep imitation learning. In *MIT Sloan Sports Analytics Conference, Boston, MA.*, 2017.

J. Ren, J. Orwell, G. A. Jones, and M. Xu. Real-time modeling of 3-d soccer ball trajectories from multiple fixed cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(3):350–362, 2008.

K. D. Routley. *A markov game model for valuing player actions in ice hockey.* PhD thesis, Applied Sciences:, 2015.

D. Seita. Understanding generative adversarial networks. Retrieved from `https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/`, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

X. Yu, Q. Tian, and K. W. Wan. A novel ball detection framework for real soccer video. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, volume 2, pages II–265–8 vol.2, July 2003a.

X. Yu, C. Xu, Q. Tian, and H. W. Leong. A ball tracking framework for broadcast soccer video, July 2003b.