Team LSTM: Player Trajectory Prediction in Basketball Games using Graph-based LSTM Networks

by

Setareh Cohan

BSc, Sharif University of Technology, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

January 2020

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Team LSTM: Player Trajectory Prediction in Basketball Games using Graph-based LSTM Networks

submitted by **Setareh Cohan** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

James J.Little, Computer Science
Co-supervisor

Leonid Sigal, Computer Science
Co-supervisor

Helge Rhodin, Computer Science
Additional Examiner

Abstract

Autonomous systems deployed in human environments must have the ability to understand and anticipate the motion and behavior of dynamic targets. More specifically, predicting the future positions of agents and planning future actions based on these predictions is a key component of such systems. This is a challenging task because the motion behavior of each agent not only depends on its own goal intent, but also the presence and actions of surrounding agents, social relations between agents, social rules and conventions, and the environment characteristics such as topology and geometry.

We are specially interested in the problem of human motion trajectory prediction in real-world, social environments where potential interactions affect the way people move. One such environment is a basketball game with dynamic and complex movements driven by various social interactions. In this work, we focus on player motion trajectory prediction in real basketball games. We view the problem of trajectory prediction as a sequence prediction task where our goal is to predict the future positions of players using their past positions.

Following the success of recurrent neural network models for sequence prediction tasks, we investigate the ability of these models to predict motion trajectories of players. More specifically, we propose a graph-based pooling procedure that uses relation networks and incorporates it with long short-term memory networks. We study the effect of different graph structures on the accuracy of predictions.

We evaluate the different variations of our model on three datasets; two publicly available pedestrian datasets of ETH and UCY, as well as a real-world basketball dataset. Our model outperforms vanilla LSTM and Social-LSTM baselines on both of these datasets.

Lay Summary

Motion trajectory prediction is an important task which aims to predict the future positions of agents given an observed sequence of their positions. This task has many application domains such as service robots, self-driving vehicles, and advanced surveillance systems. One of the major challenges for this task is the complexity of human behavior and diversity of factors affecting movements of agents such as own goal intent, interactions with surrounding agents, social rules and environment characteristics. In this thesis, we look at the problem of player trajectory prediction in basketball games, a complex and dynamic environment. We propose a model that jointly predicts player trajectories while considering interactions that take place in the game. These interactions include player-player and player-ball interactions. We evaluate the effectiveness of our model on two sets of data and outperform baseline models.

Preface

The entire work presented here is original work done by the author, Setareh Cohan, under the supervision of James J. Little and Leonid Sigal.

Table of Contents

List of Tables

List of Figures

Glossary

GNN	graph neural network
LSTM	long short-term memory
MLP	multi-layer perceptron
ReLU	rectified linear unit
RNN	recurrent neural network
VRNN	variational recurrent neural network

Acknowledgments

I would like to express my gratitude for those that have helped me throughout this journey.

First, my supervisors, James J.Little and Leonid Sigal who have guided me at every step of the way. Dr. Little, your cheerful attitude and earnest support have been always motivating. Dr. Sigal, your mentorship and invaluable feedback have made this work possible. I would like to thank my examiner committee member, Helge Rhodin, who took the time to read this thesis.

Next, I would like to thank my lovely husband, Saeid, for his love and enduring support. Next in line are my colleagues, Gursimran and Rayat, who have helped and inspired me.

Moreover, I want to thank my parents whose unconditional love and support have encouraged me throughout my life. I am forever grateful for your tireless efforts and all the opportunities you have provided for me. Next, I would like to thank my brother, Arman, who has always been my role model. I would not be here if it was not for you.

I would like to thank my friends who make my life more enjoyable. Special thanks must go to my beloved Pouneh and Arash, who, although no longer with us, continue to inspire me to live better and happier.

Chapter 1

Introduction

Perceiving, understanding and predicting human motion are essential for coexisting and interacting with humans. These predictions are the foundations based on which we must plan our future actions and movements. For example, drivers driving on a road must be able to foresee the future positions of fellow road users and adjust their paths accordingly to avoid collisions while moving towards their destination. Another example includes pedestrians walking on a sidewalk, moving towards their destinations while avoiding obstacles and accommodating fellow pedestrians.

Humans have the natural ability to understand the motion of one another and make accurate predictions of these motions. While making these predictions, we consider many complicated common-sense rules such as respecting personal space and yielding right of way. Despite being natural to humans, predicting the motion of human targets while considering such rules and social behaviors is an extremely challenging problem. Motion not only depends on one's goal, but also the presence and actions of surrounding agents, social interactions between agents, social rules and common-sense rules, and the environment characteristics such as topology and geometry. Most of these factors are not observable and need to be inferred from the available contextual information. Moreover, to be effective, motion prediction needs to be accurate and real-time operatable.

Similar to humans, intelligent systems deployed in human environments, must be able to make accurate predictions of the motion of all dynamic targets including humans around it. More specifically, predicting the future positions of these targets

and planning future movements based on these predictions is a key component of such systems. Examples of such systems include socially-aware robots, self-driving vehicles, and advanced surveillance systems.

We are interested in the problem of human motion trajectory prediction in real-world, social environments. One such environment is a basketball game with dynamic and complex motions where various interesting factors affect players motions. In a basketball game, player trajectories are affected by their personal goal intents, the goals of their teams, and movements of surrounding players. In this work, we focus on predicting players' future motion trajectories in real basketball games.

Similar to Alahi et al. (2016), we view the problem of trajectory prediction as a sequence prediction task where the goal is to predict the future positions of players using their past positions. Following the success of recurrent neural network (RNN) models for sequence prediction tasks, we first investigate the ability of a long short-term memory (LSTM) models to predict the trajectories of basketball players. Next, we incorporate game structure information and interactions of players into our LSTM-based model by proposing a graph-based pooling procedure that uses relation networks. This pooling allows the LSTMs of connected players to share their hidden state information with each other. This architecture which we call "Team-LSTM" is capable of predicting future positions of players while modeling and learning the interactions between connected players.

We evaluate our Team-LSTM model on two types of data: publicly available pedestrian trajectories (Pellegrini et al., 2009), (Lerner et al., 2007); and real basketball trajectories, rendered as a series of bird's eye views of the court (Zheng et al., 2016). For the basketball dataset, we consider poolings based on three different graph structure and for the pedestrian datasets, we consider one graph structure and test all these variations of our Team-LSTM model. Our experiments show that our Team-LSTM model outperforms the vanilla LSTM and Social-LSTM models on the pedestrian datasets. On the basketball dataset, all the variations of our model achieve comparable results with both the vanilla LSTM and Social-LSTM models and only one of the three variations outperforms the baselines.

The remainder of this thesis is organized as follows: In Chapter 2, we review related work on the topic of human behavior prediction, and human motion trajec-

tory prediction in particular. Chapter 3, contains the background information on sequence prediction with Neural Networks. In Chapter 4, we start by defining the problem we will be working on. Then, we review the Social-LSTM model in more depth. Finally, we describe our graph-based pooling and our Team-LSTM model in detail. Chapter 5 contains an overview of the datasets used for evaluation. We describe the metrics we used while evaluating our models and then we provide the results of our experiments. Lastly, in Chapter 6, we discuss the limitations of our model and possible future directions for the work.

Chapter 2

Related Work

Human behavior prediction is an important and challenging problem. Different aspects of this problem have been studied by numerous scientific communities, such as computer vision, robotics, and self-driving vehicles. Generally, research in forecasting human behavior can be grouped into predicting human-space interactions or human-human interactions (Gupta et al., 2018). The goal of the former is to learn scene-specific motion patterns (Ballan et al., 2016; Coscia et al., 2016; Hu et al., 2011; Kim et al., 2011; Kitani et al., 2012; Morris and Trivedi, 2011; Zhou et al., 2011). The latter which models the dynamic content of the scenes, including how people interact, is the focus of our work. In this work, we focus on the most closely related literature and review some of these works.

Human-Human Interaction

Pioneering work of Helbing and Molnar (1995) presented a human motion model called the Social Forces model. Social Forces models two types of forces. First, attractive forces that guide people towards their goal and second, repulsive forces that encourage people to avoid collisions. This model has been shown to achieve competitive results even on modern pedestrian datasets (Lerner et al., 2007; Pellegrini et al., 2009). Social Forces was later extended to robotics (Luber et al., 2010), and activity understanding (Leal-Taixé et al., 2011, 2014; Mehran et al., 2009; Pellegrini et al., 2010; Yamaguchi et al., 2011). Tools popular in economics have also been used to model human behavior by Antonini et al. (2006). Moreover, Gaussian

processes were used for modeling human motion Tay and Laugier (2008); Wang et al. (2007). All these methods use handcrafted energy potentials based on relative distances and scene-specific rules. In contrast, generic data-driven approaches have been proposed more recently which outperform the above traditional ones.

RNNs for Human Motion Trajectory Prediction

RNNs are a powerful class of dynamic models that have been shown to be successful in sequence prediction in different domains such as machine translation (Chung et al., 2015), speech recognition (Chorowski et al., 2014; Chung et al., 2015; Graves and Jaitly, 2014), and image captioning (Karpathy et al., 2014; Vinyals et al., 2015; Xu et al., 2015). Following this success, Alahi et al. (2016) were the first to view at human trajectory prediction problem as a sequence prediction task and use RNNs. They show that using a single RNN per person fails to capture interactions between people and propose a social pooling layer that enables modeling such interactions. Lee et al. (2017) propose an RNN encoder-decore framework that exploits a variational autoencoder (VAE) to predict trajectories. Although successful, this model does not model human-human interactions and fails in crowded, dynamic scenes. Inspired by this limitation, Gupta et al. (2018) propose Social-GAN model. This model consists of an RNN encoder-decored generator and an RNN-based discriminator. Their model has a variety of losses which encourages the model to spread its predictions and cover the space of possible paths while being consistent with the observed sequences. Social-GAN is also able to capture the social interactions because of a new pooling mechanism they call global pooling that uses a multi-layer perceptron (MLP) followed by a max pooling layer. They show that this pooling is more computationally efficient and works as well or better than the social pooling of Alahi et al. (2016).

Graph-structured RNNs

There are several papers that combine standard RNNs with graph-structured representations. Chang et al. (2016) learn a physics simulator by proposing a procedure for factorizing object dynamics into pairwise interactions. Battaglia et al. (2016) propose a model that can reason how objects in complex systems inter-

act. In addition, Sanchez-Gonzalez et al. (2018) introduce a new class of learnable models based on graph networks (GN). Following these works and the introduction of stochastic RNNs, Sun et al. (2019) proposed a model called Graph-VRNN which incorporates variational recurrent neural networks (VRNNs) with graphs. This model is able to perform state estimation and future forecasting at the level of objects and relations directly from image pixels. On a similar line, Yeh et al. (2019) proposed a model called GVRNNT for multi-agent trajectory prediction. This model incorporates graph neural networks (GNNs) and VRNNs for joint trajectory prediction suitable for sports. The main goal of using GNNs in GVRNNT is modeling permutation-equivariant interactions among players while in the work of Sun et al. (2019), the main goal of using GNNs was learning the underlying graph structure of the game. We propose to incorporate RNNs with graphs for player trajectory prediction in basketball games. Our model is computationally efficient as it only requires the position coordinate of players. Moreover, due to the use of relation networks (Santoro et al., 2017) as our graph support, our model can learn the graph structure.

Chapter 3

Background

Our task is structured as a standard sequence prediction problem where we are interested in predicting the next values for a given input sequence. Our model is a RNN and its building blocks are LSTM units. In this chapter, we review some background material relevant to the rest of our discussion.

3.1 Neural Networks

Neural networks (NNs) are computational models that map an input to an output. Inspired by the biological neural connectivities in human and animal brains, neural networks "learn" to perform tasks by considering examples, without being specifically programmed with task-specific rules (Hinton and Salakhutdinov, 2006; McCulloch and Pitts, 1990). For instance, neural networks are capable of learning to classify images into different groups using manually labeled data or without any prior knowledge (Krizhevsky et al., 2012). Analogous to the neurons and synapses in a biological brain, neural networks consist of a number of connected computational units called artificial neurons. An artificial neuron receives one or more inputs where each input has a weight associated to it. This weight determines its relative importance to other inputs and is adjusted as the learning proceeds. Each neuron also has a non-linearity function called an activation function through which it passes the weighted sum of its inputs. The purpose of this function is to introduce some non-linearity to the otherwise linear computation of

the neurons. Therefore, the output of each neuron is computed by some non-linear function of the weighted sum of its inputs. Figure 3.1 shows an example of a single neuron with m inputs of x_1 to x_m, activation function φ, and output y. The output of this neuron is

$$y = \varphi\left(\sum_{i=1}^{m} w_i x_i\right) \tag{3.1}$$

where w_i is the weight associated to input x_i.

Neurons of a neural network are typically organized into multiple layers; an input layer, multiple hidden layers, and an output layer. The input layer of a neural network is composed of input neurons that bring the initial data into the network for further processing by succeeding layers. The hidden layer is a layer in between input layers and output layers, where neurons take in a set of weighted inputs and produce an output through their activation function. The output layer produces the output and its neurons perform the same operations like the ones in the hidden layer. A neural network with multiple layers is also called a deep neural network.

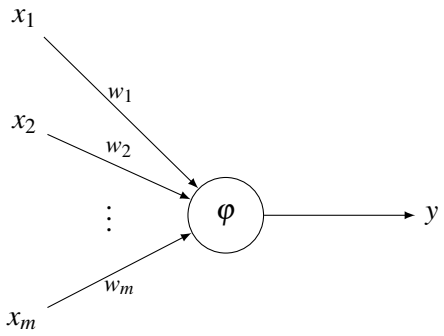

Figure 3.1: An artificial neuron with inputs x_1 to x_m, activation function φ and output y. Each input x_i has weight w_i which is learned over time from training data.

In typical neural networks, neurons of each layer can only connect to neurons in the preceding or following layers. When the direction of these connections is from inputs to hidden layers and to outputs, the network is called a feed-forward neural network. Between two layers, multiple connection patterns are possible. They can be fully connected, with every neuron in one layer connected to every neuron in the next layer. These networks are called fully connected neural networks and

Figure. 3.2 is an example of such networks.

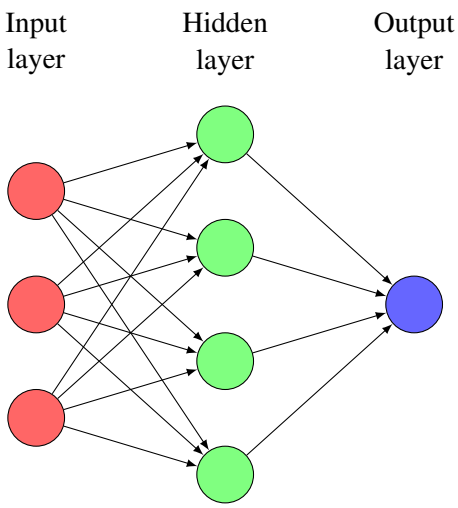

Figure 3.2: A 2-layer fully connected feed-forward neural network consisting of an input layer, a hidden layer and an output layer.

Learning in neural networks is equivalent to adjusting the network parameters so that network performance increases in a task considering sample observations. This process which is called training involves finding the connection weights of the network such that the cost function, error between observed outputs and ground-truth values, decreases over time. Backpropagation is the method used for training by calculating the gradient of the cost function associated with a given output with respect to its weights. The weight updates can then be done using any optimization algorithm such as gradient descent (GD) (LeCun et al., 2015).

3.2 Activation Function

As introduced in Section 3.1, an activation function of a neuron defines the output of that neuron given a set of input values. More specifically, to create the output of a neuron, the activation function is applied to the weighted sum of its inputs. The purpose of activation functions is incorporating non-linearity into computations. Without them, neural networks are essentially linear regression models unable to perform complex tasks known to be possible for neural networks (Hornik et al.,

1989).

There are many forms of activation functions used in practice. Sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) are three of the most applied activation functions which are described below:

- Sigmoid: A bounded and differentiable function defined for all real input values. It maps its real-valued input to a real value between 0 and 1 and has a non-negative derivative at all points.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

- Tanh: A bounded and differentiable function defined for all real input values. It maps its real-valued input to a real value between -1 and +1.

$$\tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{3.3}$$

- ReLU: It takes a real-valued input and thresholds it at zero by replacing negative values with zero.

$$f(x) = \max\{0, x\} \tag{3.4}$$

3.3 Multi-Layer Perceptron

A multi-layer perceptron (MLP) is a special sub-class of neural networks. An MLP is a feed-forward neural network with at least one hidden layer. The forward direction of connections means that connections start from the input layer and go towards the output layers. The hidden layers and output layer of an MLP contain a non-linear activation function. Figure. 3.2 is an example of a two-layered MLP (Murtagh, 1991).

3.4 Recurrent Neural Networks

A recurrent neural network (RNN) is a deep neural network where connections between neurons can form a loop. These looping connections allow the use of

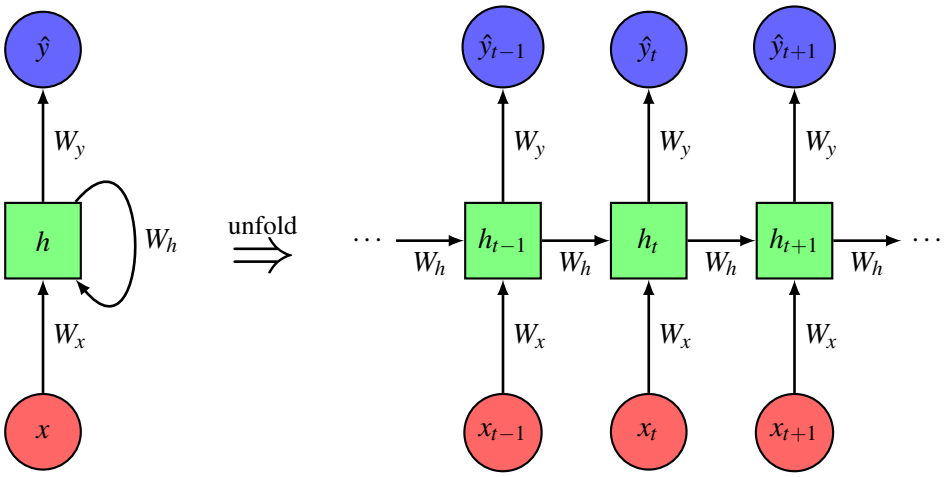

Figure 3.3: A RNN unfolded into a full chain-structured network.

previous outputs for present computations. RNNs essentially consist of copies of a single network layer connected with a directed connection to its following layer. Each layer consists of an input neuron, a hidden neuron, and an output neuron, and is responsible for computations at a particular time step (Cleeremans et al., 1989; Jordan, 1986; Pearlmutter, 1989). Figure. 3.3 shows an RNN unfolded into a chain-structured network where x_t, h_t and $\hat{y}_t$ denote the input, the hidden state, and the output of the RNN at time step t respectively. W_x, W_h and W_y are connections weights of each layer which are shared across all the layers of the RNNs, and are learned during training. At time step t, hidden state h_t is first computed using input x_t, and then output $\hat{y}_t$ is computed using the freshly computed h_t as below:

$$\begin{aligned} h_t &= \varphi_h(w_h h_{t-1} + w_x x_t + b_h) \\ \hat{y}_t &= \varphi_y(W_y h_t + b) \end{aligned} \tag{3.5}$$

where φ_h is the non-linear activation function of the hidden neuron and φ_y is the non-linear activation function of the output neuron.

3.5 Long Short-Term Memory

One of the appealing features of an RNN is its capability to exploit previous information in present computations. In theory, RNNs are capable of handling arbitrary long-term dependencies in the input sequence. In practice, when the distance between the current time step and past relevant information is small, RNNs are successful. However, as this distance grows, RNNs become unable to learn to connect the past information. This problem, called the vanishing gradient problem, is caused by backpropagation through time during training RNNs. During backpropagation, gradients of later outputs have to propagate back to affect computations of early outputs. As gradients pass back through many time steps, they tend to exponentially decrease. As the gradients shrink, the contributions of these gradients to computations of early steps become negligible. The vanishing gradient problem in RNNs is explored in depth in Bengio et al. (1994).

Long short-term memory (LSTM) is a special type of RNN designed to avoid the vanishing gradient problem when long-term dependencies exist. The original LSTM architecture was proposed by Hochreiter and Schmidhuber (1997) and is composed of a memory cell, an input gate, an output gate. The cell remembers values over different time intervals and the three gates regulate the flow of information into and out of the cell. The input gate regulates the extent to which a new input value flows into the cell and the output gate regulates the extent to which the value in the cell is used to compute the output activation of the LSTM unit. Forget gates were added by Gers et al. (1999) to the original LSTM structure. The forget gate controls the extent to which a value remains in a cell and without these gates, cell state values of LSTMs may grow indefinitely and eventually cause the network to break down. More modification was proposed by Gers and Schmidhuber (2000) who incorporated peephole connections from internal cells to the gates of the same cell and omitted output activation function. These connections help with learning precise timings of events. The most commonly used LSTM architecture in the current literature can contain all these modifications and was proposed by Graves and Schmidhuber (2005). They introduced full backpropagation through time for LSTM networks in this version which simplifies the training procedure and the implementation of LSTMs. Figure 3.4 shows two variants of this LSTM model;

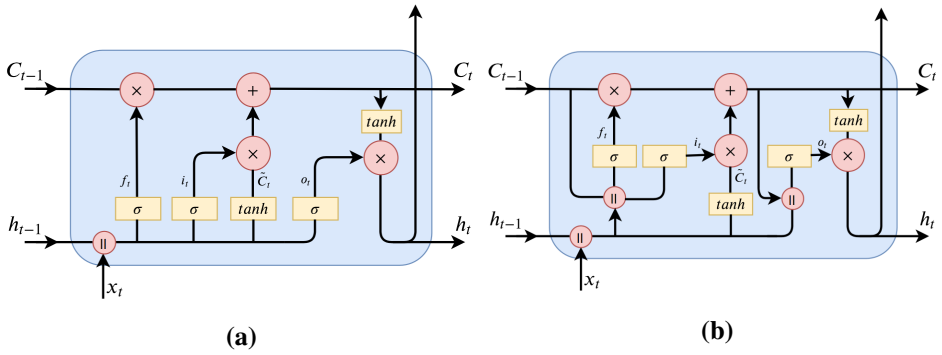

Figure 3.4: Two variations of LSTM cells. 3.4a shows an LSTM cell without peephole connections which we call vanilla LSTM. 3.4b shows an LSTM cell with peephole connections. Cancatination operation is shown with symbol || in these figures.

Figure 3.4a without peephole connections which we refer to as vanilla LSTM, and figure 3.4b with peephole connections. The computations used during a forward pass of a vanilla LSTM unit as shown in figure 3.4a are as below:

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) && \text{forget gate} \\
\tilde{C}_t &= \tanh(W_c[h_{t_1}, x_t] + b_c) && \text{candidate memory cell} \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) && \text{input gate} \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t && \text{memory cell} \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) && \text{output gate} \\
h_t &= o_t * tanh(C_t) && \text{hidden state}
\end{aligned}
$$

where x_t is the input, C_t is the cell state value, and h_t is the hidden state of the LSTM unit at time step t. σ denotes the sigmoid activation function, and $*$ indicates element-wise multiplications between two vectors. For the LSTM structure with

peephole connections of 3.4b, computations of a forward pass are the following:

$$
\begin{aligned}
f_t &= \sigma(W_f[C_{t-1}, h_{t-1}, x_t] + b_f) && \text{forget gate} \\
\tilde{C}_t &= \tanh(W_c[h_{t_1}, x_t] + b_c) && \text{candidate memory cell} \\
i_t &= \sigma(W_i[C_{t-1}, h_{t-1}, x_t] + b_i) && \text{input gate} \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t && \text{memory cell} \\
o_t &= \sigma(W_o[C_t, h_{t-1}, x_t] + b_o) && \text{output gate} \\
h_t &= o_t * tanh(C_t) && \text{hidden state}
\end{aligned}
$$

Another variation of the vanilla LSTM is the Gated Recurrent Unit (GRU), introduced by Cho et al. (2014). This architecture combines the forget gate and input gate into a single update gate. It also merges the cell state and hidden state. The output gate is called a reset gate in this architecture which specifies whether the current hidden state would ignore the previous hidden state or not. The resulting model is a simpler structure than the standard vanilla LSTM model and is more computationally efficient.

Chapter 4

Method

The problem of human trajectory prediction is a challenging and important problem. Basketball games are complex and dynamic environments where player trajectories are inspired by many interesting factors including goal of teams, team structures, roles of players, and interactions between players. The characteristics of basketball games motivate us to build a model that can predict a player's motion trajectory while incorporating game-related structures and taking the behavior of other players into account. In this section, we first describe the basis of our model called the Social-LSTM model. Following that, we describe our modification which is a new graph-based pooling procedure that uses relation networks. An overview of our model called TEAM-LSTM can be viewed in figure 4.2.

4.1 Problem Formulation

Following the convention of previous works, we assume that each scene is initially preprocessed to obtain the spatial x-y coordinates of all players at different time steps (Alahi et al., 2016; Gupta et al., 2018; Sun et al., 2019). At any time step t, i-th player in the scene is presented by their x-y coordinates (x_t^i, y_t^i). Our task is to predict the positions of all players from time steps $T_{\text{obs}+1}$ to T_{pred} while having their ground-truth positions from time step 1 to T_{obs} observed. This is essentially equivalent to a sequence prediction problem where the input sequence is the observed positions of a player and the output is a sequence identifying the future positions

of this player at different time steps.

4.2 Social LSTM

As discussed in Section 4.1, the problem of human motion trajectory prediction can be formulated as a sequence prediction task. Following the success of LSTM networks for learning and generalizing the properties of sequences like handwriting (Graves, 2013) and speech (Graves and Jaitly, 2014), Alahi et al. (2016) were the first to use LSTMs for the task of human trajectory prediction. In particular, each person has their own LSTM where the observed sequence of his/her positions is the input of this LSTM. The output of each LSTM is the predicted positions of its corresponding person in future time steps. While this model works well for scenes with few people having little to no interactions, Alahi et al. (2016) show that it fails to capture interactions of people in crowded and interactive scenes.

Motivated by this limitation, Alahi et al. (2016) proposed the Social-LSTM model. In this model, the interactions of nearby people are incorporated by introducing a new pooling mechanism called social pooling. With social pooling, at any time step, each LSTM cell receives pooled hidden state information from the LSTM cells of its neighbors, allowing them to share motion trajectory information. Neighboring LSTMs are modeled using a grid-based pooling as below. The pooled social hidden state tensor of i-th person at time step t is shown by H_t^i and computed as below:

$$H_t^i(m,n,:) = \sum_{j \in \mathcal{N}_i} 1_{mn}[x_t^j - x_t^i, y_t^j - y_t^i] h_{t-1}^j \tag{4.1}$$

where h_t^i, x_t^i, and y_t^i are the hidden state, and x and y positions of i-th person at time step t. h_{t-1}^j coresponds to the hidden state of j-th person at time step $t-1$, $1_{mn}[x,y]$ is and indicator function determining whether (x,y) is located in the (m,n)-th cell of the grid. $\mathcal{N}_i$ is the set of neighbors of person i. $H_t^i(m,n,:)$ is essentially the sum of hidden states of all neighbors in the n-th grid of person m. Figure 4.1 shows a visualization of this pooling.

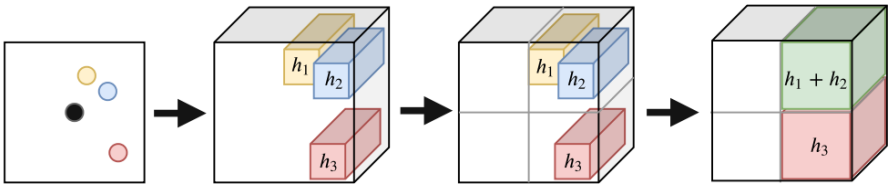

Figure 4.1: Social pooling for the person represented by a black-dot. The hidden states of the neighbors (shown in yellow, blue and red) are pooled within a certain spatial distance.

4.2.1 Position Prediction

The pooled social hidden tensor of person i at time step t is incorporated into the calculations of its LSTM cell as below:

$$\begin{aligned} e_t^i &= \phi(x_t^i, y_t^i, W_e) \\ a_t^i &= \phi(H_t^i, W_a) \\ h_t^i &= \text{LSTM}(h_{t-1}^i, e_t^i, a_t^i, W_l) \end{aligned} \tag{4.2}$$

where ϕ is an embedding function with ReLU non-linearity, and W_e, W_a are the embedding weights, and W_l is the weights of the LSTM cell.

To predict the x-y position of person i at the next time step $t+1$, the hidden state of time step t is passed through a linear output embedding layer which returns the parameters of a bivariate Gaussian distribution; the mean $\mu_{t+1}^i = (\mu_x, \mu_y)_{t+1}^i$, standard deviation $\sigma_{t+1}^i = (\sigma_x, \sigma_y)_{t+1}^i$, and correlation coefficient ρ_{t+1}^i. The predicted coordinates $(\hat{x}_{t+1}, \hat{y}_{t+1})^i$ are then sampled from this bivariate Gaussian distribution:

$$\begin{aligned} [\mu_{t+1}^i, \sigma_{t+1}^i, \rho_{t+1}^i] &= W_p h_t^i \\ (\hat{x}_{t+1}, \hat{y}_{t+1})^i &\sim \mathcal{N}(\mu_{t+1}^i, \sigma_{t+1}^i, \rho_{t+1}^i) \end{aligned} \tag{4.3}$$

The network is trained by minimizing the negative log-likelihood loss (NLL) of the ground-truth future positions given the predicted distribution parameters, for all trajectories in the training dataset. NLL of the trajectory of i-th person or i-th

trajectory, L^i is defined as below:

$$L^i(W_e, W_a, W_l, W_p) = -\sum_{t=T_{\text{obs}+1}}^{T_{\text{pred}}} \log P(x_t^i, y_t^i | \mu_t^i, \sigma_t^i, \rho_t^i) \tag{4.4}$$

4.3 Approach

Looking back at the social pooling mechanism explained in detail in section 4.2, we can see that it has the following properties:

- Hidden states are shared among people who are close enough to fall in one anothers neighborhood distances.
- We do not differentiate between hidden states of people that fall within a certain grid.

These properties show that Social-LSTM model which was desined for modeling interactions between pedestrians is not a good match for modeling the interactions of basketball games. This is because in a basketball game:

- Interactions are not limited to spatially close players.
- There is a difference between the type of interactions that are in a certain grid; for instance player-player interactions are different from player-ball interactions. Moreover, team-members interact differently from opponents.
- There are certain interactions that are expected to be present in a basketball game such as interactions between team members, between opponents, and between players and the ball.

Inspired by these differences, we propose an LSTM-based model called "Team-LSTM" for player motion trajectory prediction in basketball games. We introduce a new graph-based pooling method which uses relation networks and is capable of incorporating the structure of the game while modeling the interactions between players and players and the ball. The overall architecture of our model is shown in figure 4.2.

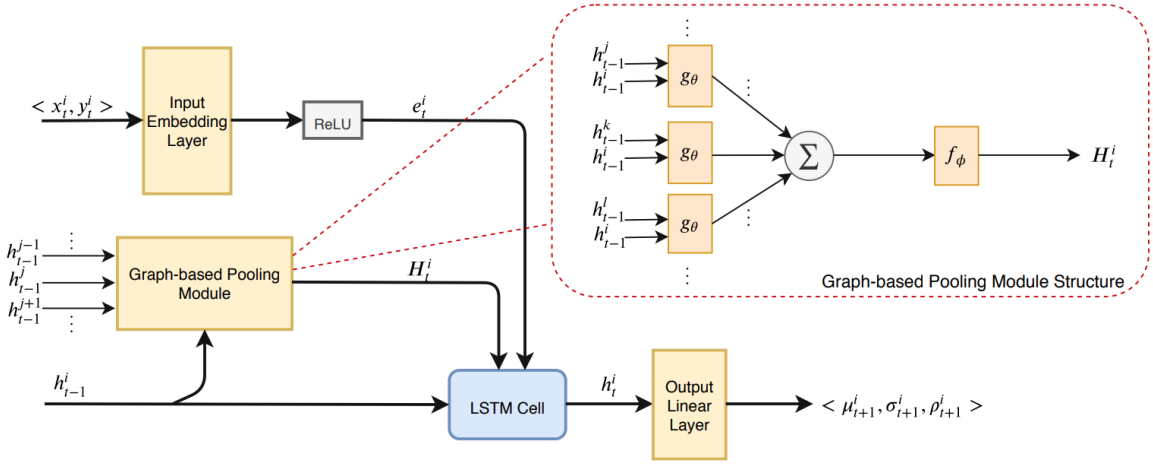

Figure 4.2: The figure shows the block diagram of our Team-LSTM model. Input embedding layer, which is a linear layer followed by ReLU non-linearity, computes the input embedded vector e_t^i. The graph-based pooling module computes the pooled hidden tensor H_t^i. It consists of a g_θ and f_ϕ modules (shown in orange) which are both MLPs. g_θ inputs all hidden state pairs of connected agents and the summation of these vectors is then fed into f_ϕ to compute H_t^i. The output linear layer is responsible to derive the outputs given the current hidden state h_t^i.

4.3.1 Graph-based Pooling

In a basketball game, every player has a different motion pattern; they move with their own velocities and accelerations while being affected by different game structure factors such as goal intent of their team, their interactions with their teammates, opponents, and the ball. Incorporating these interactions into an LSTM-based model is expected to boost the accuracy of the predictions. Therefore, we propose a new graph-based pooling structure using relation networks proposed by Santoro et al. (2017). Our pooling allows for each LSTM cell to receive pooled hidden state information from the LSTM cells of the players connected to it. Connected LSTMs are initially determined by a set of hidden state pairs, however, the relations between these pairs is later learned by the relation network modules during training. Given a hidden state dimension D, and neighborhood size N_o, and embedding dimension R, we construct an $N_o \times R$ dimensional pooled hidden tensor

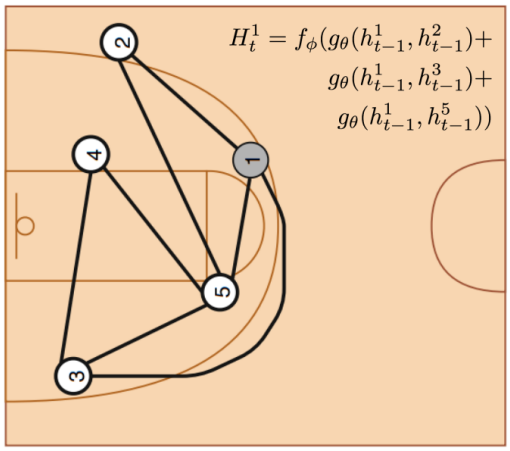

Figure 4.3: Pooled hidden state of player 1 at time step t, H_t^i, is computed as shown.

for the i-th trajectory at time step t shown by H_t^i:

$$H_t^i = f_\phi \left(\sum_{j \in \mathcal{N}_i} g_\theta(h_{t-1}^i, h_{t-1}^j) \right) \tag{4.5}$$

where h_{t-1}^i is the hidden state of i-th player at time step $t-1$, and $\mathcal{N}_i$ are the set of players connected to player i at time step $t-1$. Relation network modules g_θ and f_ϕ are functions with learnable parameters θ and ϕ respectively. For our purpose, $g_\theta : 2 \times D \to D$ and $f_\phi : D \to R$ are MLPs where the parameters are learnable weights, making them end-to-end differentiable. The output of g_θ is called a "relation", and its role is to infer the ways its two hidden state inputs are related, or if they are related at all. Therefore, our pooling model can learn the relations between pairs of hidden states and the initial graph structure only provides the pairing. Figure 4.3 shows a visualization of our graph-based pooling. x

4.3.2 Position Prediction

Incorporation of the newly desinged pooled hidden tensor into the LSTM calculations is similar to the one of Alahi et al. (2016) and is done by the following

calculations:

$$
\begin{aligned}
e_t^i &= \phi(x_t^i, y_t^i, W_e) \\
h_t^i &= \text{LSTM}(h_{t-1}^i, e_t^i, H_t^i, W_l)
\end{aligned}
\tag{4.6}
$$

where $\phi : 2 \rightarrow R$ is an embedding function with ReLU non-linearity, and W_e is the embedding weight, W_l is the weights of the LSTM cell. H_t^i, our newly proposed pooled hidden tensor defined in equation 4.5, is concatinated with e_t^i and is inputed to the LSTM cell. Comparing equations of 4.2 and 4.6, we can see that instead of passing H_t^i through an embedding layer, we now directly use it as an input to the LSTM cell. Note that the LSTM cells used in both the Social-LSTM model and our model is the vanilla LSTM explained in section 3.5.

Similar to Alahi et al. (2016), positions are assumed to be from a bivariate Gaussian distribution. To predict the position of player i at time step $t+1$, the hidden state of time step t is passed through a linear output embedding layer $\Phi : R \rightarrow 5$ which returns five parameters of the bivariate Gaussian distribution; the mean $\mu_{t+1}^i = (\mu_x, \mu_y)_{t+1}^i$, standard deviation $\sigma_{t+1}^i = (\sigma_x, \sigma_y)_{t+1}^i$, and corelation coefficient ρ_{t+1}^i. The predicted coordinates $(\hat{x}_{t+1}, \hat{y}_{t+1})^i$ are then sampled from this bivariate Gaussian distribution:

$$
\begin{aligned}
[\mu_{t+1}^i, \sigma_{t+1}^i, \rho_{t+1}^i] &= W_p h_t^i \\
(\hat{x}_{t+1}, \hat{y}_{t+1})^i &\sim \mathcal{N}(\mu_{t+1}^i, \sigma_{t+1}^i, \rho_{t+1}^i)
\end{aligned}
\tag{4.7}
$$

Along the same lines, the training is performed by minimizing the NLL of the ground-truth positions given the prediced distribution parameters as below:

$$
L^i(W_e, W_l, W_p, \phi, \theta) = -\sum_{t=T_{\text{obs}+1}}^{T_{\text{pred}}} \log P(x_t^i, y_t^i | \mu_t^i, \sigma_t^i, \rho_t^i) \tag{4.8}
$$

4.3.3 Inference for Trajectory Prediction

During test time, we use the trained Team-LSTM model to predict the future position $(\hat{x}_t^i, \hat{y}_t^i)$ of the i-th player. From time step T_1 to T_{obs}, we input the ground-truth positions to our Team-LSTM cells. However, from time $T_{\text{obs}+1}$ to T_{pred}, we use the

predicted position from the previous Team-LSTM cell in place of the ground-truth coordinates as they are unavailable.

Chapter 5

Experiments

Our implementation is done with PyTorch. Following Alahi et al. (2016), we set the values of hidden state dimension to 128 for all the Team-LSTM cells, and embedding dimension is set to 64. We build pooling modules of equation 4.5 following Santoro et al. (2017). g_θ is a four-layer MLP with 256 units per layer with ReLU non-linearities between every pair of layers. f_ϕ is a three-layer MLP with 256 units per layer, 50% dropout at second layer and ReLU non-linearities between all pairs of layers. The pairings provided to g_θ are determined by an input graph structure to the Team-LSTM model. We use the fully-connected graph structure for the pedestrian dataset and for the baskeball dataset, we use three different graph structures: fully-connected graph, a disconnected graph where each person is connected to its team-mates, and a connected graph where each person is connected to its team-mates and the ball. The input embedding layer of equation 4.6 is a linear layer followed by a ReLU non-linearity. The output embedding layer Φ of equation 4.7 is also a linear layer mapping the D-dimensional hidden state to a 5-dimensional output. Following Alahi et al. (2016), we used a learning rate of 0.003 and Adagrad optimizer Duchi et al. (2011) with a weight decay of 0.0005 for training the model. We trained all the variations of our model and the baseline models for 30 epochs on the pedestrain dataset and for 12 epochs on the basketball dataset. We use the same hyperparameters described above while implementing and training the baseline models as well.

5.1 Datasets

We validate our Team-LSTM model on two types of data. First is a dataset of real basketball trajectories, rendered as a series of birds eye views of the court. Next two dataset are publicly available pedestrian trajectories datasets: ETH and UCY. All these datasets are explained in detail below.

5.1.1 Basketball Data

We use the basketball dataset from Zheng et al. (2016) which contains basketball tracking data from a professional basketball league. The dataset includes a training set and test set, each containing a number of possession files. Each of these possession files include a sequence of game events during which one team has possession of the ball. Possessions end when the ball is captured by the opposing team, or if a shot is made. Training set contains $32,377$ and the test set contains $3,953$ possession files.

Each possession file is stored as a sequence of tuples, one for each time step. Each tuple contains 16 numbers with formatted as:

$$(\text{ball}, \text{i}_1, ..., \text{i}_{10}, \text{label}_1, \text{label}_2, \text{label}_3, \text{label}_4, \text{goal})$$

where the first 11 numbers are the positions of the ball and players, the next 4 numbers are the velocity labels of of player 1 for 4 intermediate time-steps, and the last number is the macro-goal label of player 1.

Players 1 - 5 are the offense team, defined as the team that has the ball in the possession. Players 6 - 10 are the defense team. For each time step, player and ball positions are stored as serial indices in a 400×360 grid representation of the half-court, indexed row-major, with $(0,0)$ in the top-left. Figure 5.1 shows a visualization of this positioning. The player and ball positions are sampled at $25/4 = 6.25$Hz and each possession contains at most 70 sub-sampled frames.

For each time step, the velocity labels consist of the instantaneous velocity of player 1 for 4 intermediate time steps. A single velocity label is a row-major serial index in a 17×17 velocity grid centered at that player, where $(0,0)$ is in the top-left. The macro-goal label is a stationary point of player 1 along its trajectory. Macro-

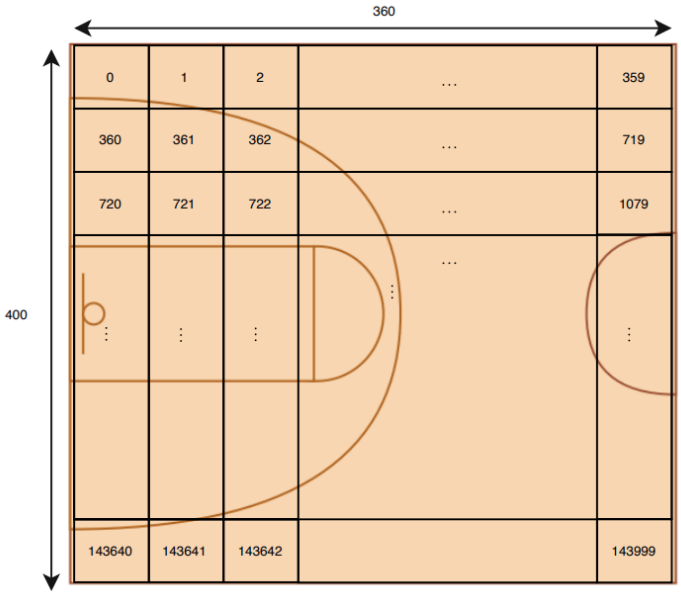

Figure 5.1: At each time step, player and ball positions are stored as serial indices in a 400×360 grid representation of the half-court.

goals are weak labels for the long-term intent of the player, i.e. which region of the court the player wants to reach.

In order to build the trajectories, we only consider the positions and ignore the velocity and macro-goal labels. In Subsection 5.1.1, we explain how the trajectories are extracted from the raw possession files.

Preprocessing

As explained earlier, each possession file includes the successive positions of the ball and all 10 players. The successive positions of each agent is called a trajectory and we are intersted in extracting the 11 trajectories of each possession file. We want each trajectory to have a number of examples with format of:

$$(\text{frame\_number}, \text{agent\_id}, y, x) \tag{5.1}$$

and we preprocess all possession files to extract all such trajectories.

For each possession file with 11 agents, we have 11 trajectories to extract. To extract trajectory of a-th agent of p-th possession file with postion value pos_value in the raw files, assuming the possession file has n tuples, we follow the steps below:

- Build n examples of format 5.1.
- Set frame_number of example i to $p \times n + i$ for $i \in [0, 10]$.
- Set agent_id of all examples to $p \times 11 + a$.
- Set y to $\lfloor \frac{\text{pos\_value}}{360} \rfloor$.
- Set x to (pos_value mod 360).

Following the above steps, we will have 11 trajectories per possession file of the dataset. Frame numbers and agent ids of trajectories of different possession files will be distinct as well. Overall, there are $356,147$ training and $43,483$ test examples in the final preprocessed dataset. For this dataset, following the experimental setup of Sun et al. (2019), we set every 5 frames as 1 step. Therefore, we consider 10 steps in total, 6 observed and 4 unobserved. We consider all 11 agents in our model.

5.1.2 Pedestrian Data

The two publicly available human motion trajectory datasets that we used for evaluating our models are the ETH and UCY datasets. The former contains 2 sets of data and the later contains 3 sets of data. In order to make full use of the dataset while training our models, we use the leave-one-out approach. We train and validate our model on 4 sets and test on the remaining set. We repeat this procedure for all of the 5 sets.

For both the ETH and UCY datasets, frames are sampled at a frame rate of 0.4 and trajectories are 8secs long. Following experiments of Alahi et al. (2016), we observe the first 3.2secs or 8 frames of each trajectory and predict the next 4.8secs or 12 frames.

ETH

The ETH dataset proposed by Pellegrini et al. (2009), contains human motion trajectory data from public scenes. This dataset includes two scenes each with 750 different pedestrians and is split into two sets of data: ETH and Hotel. The format of enteries of trajectories are similar to the one of basketball 5.1.

UCY

The UCY dataset was proposed by Lerner et al. (2007) and similar to the ETH dataset, consists of real-world human trajectories of public scences. This dataset contains two scenes with 786 people and has 3 components: ZARA01, ZARA02 and UCY. Agian, the format of enteries of trajectories are similar to the one of basketball 5.1.

5.2 Metrics

Following the conventions of previous work (Alahi et al., 2016; Gupta et al., 2018; Pellegrini et al., 2009), we use the following metrics to evaluate our model:

- Average Displacement Error (ADE): Average ℓ_2 distance between ground-truth positions and our predictions. The ADE at T consecutive frames containing N agents is calculated as

$$ADE = \frac{1}{T}\frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N}\sqrt{(\hat{x}_t^i - x_t^i)^2 + (\hat{y}_t^i - y_t^i)^2}. \tag{5.2}$$

- Final Displacement Error (FDE): The distance between the final ground-truth position and our final predicted position at the end of the prediction period. Therefore, the FDE of player i at last frame $t = T_{\text{pred}}$ containing N agents is calculated as

$$FDE = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(\hat{x}_{T_{\text{pred}}}^i - x_{T_{\text{pred}}}^i)^2 + (\hat{y}_{T_{\text{pred}}}^i - y_{T_{\text{pred}}}^i)^2}. \tag{5.3}$$

5.3 Results

In this part, we evaluate the accuracy of different variations of our model on the basketball and pedestrain datasets. The variations of our Team-LSTM model are:

- Graph-FC: Our Team-LSTM model where the initial graph structure is a fully-connected graph.
- Graph-TMS: Our Team-LSTM model where the initial graph structure is disconnected graph with each player connected to all of their team members. This model is only applicable to the basketball dataset.
- Graph-TMSB: Our Team-LSTM model where the initial graph structure is graph with each player connected to all of their team members and the ball. This model is also only applicable to the basketball dataset.

We compare against the following baselines:

- Vanilla: A simple vanilla LSTM with no pooling mechanism.
- Social: The Social-LSTM method proposed by Alahi et al. (2016).

The accuracy results of our model and the baselines, computed on ADE and FDE metrics tested on pedestrain datasets are available in Table 5.1. As expected, our Team-LSTM model with an initial fully-connected graph structure outperforms both the vanilla LSTM and Social-LSTM models. Social-LSTM model performs slightly better than the vanilla LSTM model. However, the Social-LSTM results are not as good as the reported results of the original paper (Alahi et al., 2016). The authors of this paper trained the model on synthetic dataset and then fine-tuned on real datasets. We do not use synthetic data to train the models which could potentially lead to worse performance.

Table 5.2 shows accuracy results of our model and the baselines, computed on ADE and FDE metrics tested on the basketball dataset. Again, the vanilla LSTM model has the highest prediction error among all models. Social-LSTM performs slightly better than the vanilla model but the improvements are very small. All variations of our model achieve errors lower than or equal to the errors achieved by Social-LSTM and vanilla LSTM models. Graph-TMS performs worse than

Metric	Dataset	Methods		
		Vanilla	Social	Graph-FC
ADE	ETH	0.181	0.182	**0.159**
	HOTEL	0.158	0.153	**0.139**
	ZARA01	0.373	0.287	**0.265**
	ZARA02	0.233	0.232	**0.193**
	UCY	0.221	0.215	**0.192**
	Avg.	0.233	0.214	**0.188**
FDE	ETH	0.287	0.277	**0.262**
	HOTEL	0.251	0.254	**0.232**
	ZARA01	0.469	**0.352**	0.363
	ZARA02	0.331	0.294	**0.264**
	UCY	0.302	0.276	**0.273**
	Avg.	0.328	0.291	**0.277**

Table 5.1: Quantitative results of methods across pedestrian datasets. We report two error metrics Average Displacement Error (ADE) and Final Displacement Error (FDE) for the unobserved part of the trajectories in meters. Our method consistently outperforms S-LSTM method and is especially good for long term predictions (lower is better).

Graph-FC and Graph-TMSB. This seems intuitive as the location of the ball and its relative position to player positions is a valuable information while predicting the positions of all agents (players and the ball). Moreover, Graph-TMS performs worse than Graph-FC. This is also intuitive as the fully-connected structure is able to capture more of the interactions. Graph-TMSB slightly outperforms Graph-FC in our setting. However, we expect Graph-FC to be the strongest variation as it has the potential to learn all the pairwise relations between all the agents while Graph-TMSB has access to a subset of these relations. In general, the unimpressive performance of Team-LSTM variations against the Social-LSTM model may be due to the restricted training time. As our pooling is more complex than the social pooling, it requires longer training time to learn the model parameters than the Social-LSTM model. However, We have trained all of the variations of our Team-LSTM model for 12 epochs while training the Social-LSTM model for 30 epochs. Although a weak argument and needs further investigation, longer training could result in better performance for all the variations of our model.

Metric	Methods				
	Vanilla	Social	Graph-FC	Graph-TMS	Graph-TMSB
ADE	0.196	0.193	0.193	0.194	**0.191**
FDE	0.264	0.261	0.259	0.259	**0.257**

Table 5.2: Quantitative results of methods on basketball dataset. We report the two error metrics Average Displacement Error (ADE) and Final Displacement Error (FDE) for the unobserved part of the trajectories. Our method consistently outperforms the Social-LSTM method and is especially good for long term predictions (lower is better).

Figure 5.2 shows the average ℓ_2 distance between the ground-truth and predicted positions of all agents from time step $t = 1$ to $t = T_{obs}$ for the basketball dataset. As can be seen, variations accross different models is quite small. We can see that the Graph-TMSB and Graph-FC errors generally go down over time, as the system integrates evidence from multiple frames. Similar to results of table 5.2, Graph-TMS and Graph-FC outperform other methods, and the Graph-TMS method slightly outperforms Graph-FC. As expected, the vanilla model generally has the highest error and the Social-LSTM method outperforms the vanilla model. Graph-TMS starts with a higher initial error than other methods, but manages to outperform both the vanilla and social methods. Errors of all these three models starts to increase after 3 time steps. This shows that, unexpectedly, they all fail to utilize information from earlier observations.

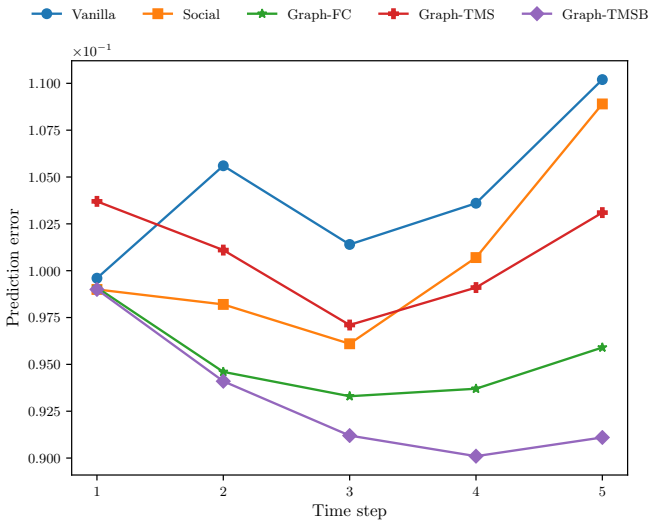

Figure 5.2: Average ℓ_2 distances to ground-truth positions at different time steps on basketball data.

Chapter 6

Conclusions

In this thesis, we explored a graph-based pooling method for LSTMs called the Team-LSTM model for player motion trajectory prediction in basketball games. By looking at trajectories as a sequence of positions, we view trajectory prediction as a sequence prediction task. Following the success of LSTMs for sequence prediction, the first solution would be to use an LSTM cell per person and predict the trajectories of all players. However, Alahi et al. (2016) show that the naive use of one LSTM cell per person does not capture the interactions of people who are close to each other. They address this limitation by proposing a new pooling strategy called Social pooling. At every time step, the LSTM cell corresponding to each person receives pooled hidden states from the LSTM cells of its neighbors. This pooling allows the LSTMs of spatially close sequences to share their hidden states with each other, and can automatically learn typical interactions that take place among pedestrians.

In our work, we focus on predicting the motion trajectories of basketball players. In a basketball game, player trajectories are affected by a number of factors such as the interactions of each player with its team-members, opponents, as well as the ball. Although Social-LSTM model can capture interactions between players, it is designed for capturing pedestrian interactions. Pedestrian interactions are limited to people who are spatially close and do not contain any game-related information such as being a member of a team. We expect that incorporating such game-related information while pooling the hidden states, can improve the accu-

racy of predictions of basketball players. Therefore, we propose a new graph-based pooling structure using relation networks proposed by Santoro et al. (2017). Our pooling allows for each LSTM cell to receive pooled hidden state information from the LSTM cells of players connected to it. Although connected LSTMs are intially determined by a set of LSTM pairs, the relations betwenn these pairs are later learned by the relation network modules during training. Following this pooling procedure, we have a tensor per player which includes information from hidden states of players connected to it. These pooled hidden state tensors are then incorporated into LSTM calculations similar to Alahi et al. (2016) as shown in equations 4.6. We finally sample future locations from a bivariate Gaussian distribution whose parameters are the outputs of our LSTM cells.

We evaluate our model on three datasets. Two publicly available pedestrian datasets, ETH (Pellegrini et al., 2009) and UCY (Lerner et al., 2007), and a basketball dataset containing real basketball trajectories (Zheng et al., 2016). Our model outperforms the Social-LSTM and vanilla LSTM models on the pedestrian dataset by a good margin. On the basketball dataset, Graph-TMSB variation of our model also outperforms the Social-LSTM and vanilla LSTM models. We believe that we can achieve better results with longer training on the basketball dataset.

One line of future work is to train our model for longer and observing the performance of our Team-LSTM model. As our pooling adds a number of parameters to the base LSTM model, we expect it to require longer training time in order to accurately learn these parameters.

Another one would be to compare our model against other variations of RNNs such as VRNNs, or to implement our pooling on top of such cells instead of vanilla LSTM cells (Chung et al., 2015). One of the interesting recent works on trajectory prediction is a graph-structured variational RNNs (Graph-VRNN) which predicts positions by taking the actual frames as input. It would be interesting to compare our model agains this recent model as well.

Another line of future work is to include more data in our model. For instace, static-scene frame images could be embedded and used as an additional input to the LSTM cells. Another example would be to input players poses as an additional input the LSTM cells. Moreover, we could use the teams of players as additional input to the LSTMs. This is a very interesting direction as it preserves team mem-

bership information even when the graph structure does not enforce it.

Furthermore, we could incorporate more game-related stuctures into our pooling. For instance, we could consider three different relation functions g_θ for team-member interactions, opponent interactions, and player-ball interactions. Another example would be to extend our model by including multi-class settings where different player roles such as center, forward and guard (or agent types in general such as bicycles, skateboards, carts, and pedestrians) are considered in the pooling procedure.

Lastly, there is a need for a richer real-world basketball dataset that contains visual information along positions data. Although the currently used basketball dataset is a good step for evaluating our model, it does not contain useful information such as roles of players, positions of players in a game rather than a single possession, and the acutal frame images. Also, there is a need for a deeper exploration of what architecture works better for our task. Currently, we are using LSTMs for our base-RNN and the MLPs for our graph-based pooling are identical to the ones used by Santoro et al. (2017). A comparative study to benchmark different RNN architectures could also be very useful for future research in this direction.

Bibliography

A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016. → pages 2, 5, 15, 16, 20, 21, 23, 26, 27, 28, 32, 33

G. Antonini, M. Bierlaire, and M. Weber. Discrete choice models of pedestrian walking behavior. *Transportation Research Part B: Methodological*, 40(8): 667–687, 2006. → page 4

L. Ballan, F. Castaldo, A. Alahi, F. Palmieri, and S. Savarese. Knowledge transfer for scene-specific motion prediction. In *European Conference on Computer Vision*, pages 697–713. Springer, 2016. → page 4

P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016. → page 5

Y. Bengio, P. Simard, P. Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994. → page 12

M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016. → page 5

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. → page 14

J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint arXiv:1412.1602*, 2014. → page 5

J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015. → pages 5, 33

A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381, 1989. → page 11

P. Coscia, F. Castaldo, F. A. Palmieri, L. Ballan, A. Alahi, and S. Savarese. Point-based path prediction from polar histograms. In *2016 19th International Conference on Information Fusion (FUSION)*, pages 1961–1967. IEEE, 2016. → page 4

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. → page 23

F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE, 2000. → page 12

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999. → page 12

A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. → page 16

A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772, 2014. → pages 5, 16

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18 (5-6):602–610, 2005. → page 12

A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2255–2264, 2018. → pages 4, 5, 15, 27

D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995. → page 4

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. → page 7

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. → page 12

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. → page 9

W. Hu, N. Xie, L. Li, X. Zeng, and S. Maybank. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):797–819, 2011. → page 4

M. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of the Eighth Annual Conference of the Cognitive Science Society (Erlbaum, Hillsdale, NJ), 1986*, 1986. → page 11

A. Karpathy, A. Joulin, and L. F. Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, pages 1889–1897, 2014. → page 5

K. Kim, D. Lee, and I. Essa. Gaussian process regression flow for analysis of motion trajectories. In *2011 International Conference on Computer Vision*, pages 1164–1171. IEEE, 2011. → page 4

K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert. Activity forecasting. In *European Conference on Computer Vision*, pages 201–214. Springer, 2012. → page 4

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. → page 7

L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 120–127. IEEE, 2011. → page 4

L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese. Learning an image-based motion context for multiple people tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3542–3549, 2014. → page 4

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015. → page 9

N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017. → page 5

A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. In *Computer graphics forum*, volume 26, pages 655–664. Wiley Online Library, 2007. → pages 2, 4, 27, 33

M. Luber, J. A. Stork, G. D. Tipaldi, and K. O. Arras. People tracking with human motion predictions from social forces. In *2010 IEEE International Conference on Robotics and Automation*, pages 464–469. IEEE, 2010. → page 4

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 52(1-2):99–115, 1990. → page 7

R. Mehran, A. Oyama, and M. Shah. Abnormal crowd behavior detection using social force model. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 935–942. IEEE, 2009. → page 4

B. T. Morris and M. M. Trivedi. Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach. *IEEE transactions on pattern analysis and machine intelligence*, 33(11):2287–2301, 2011. → page 4

F. Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991. → page 10

B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989. → page 11

S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th International Conference on Computer Vision*, pages 261–268. IEEE, 2009. → pages 2, 4, 27, 33

S. Pellegrini, A. Ess, and L. Van Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *European conference on computer vision*, pages 452–465. Springer, 2010. → page 4

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018. → page 6

A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017. → pages 6, 19, 23, 33, 34

C. Sun, P. Karlsson, J. Wu, J. B. Tenenbaum, and K. Murphy. Stochastic prediction of multi-agent interactions from partial observations. *arXiv preprint arXiv:1902.09641*, 2019. → pages 6, 15, 26

M. K. C. Tay and C. Laugier. Modelling smooth paths using gaussian processes. In *Field and Service Robotics*, pages 381–390. Springer, 2008. → page 5

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015. → page 5

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007. → page 5

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015. → page 5

K. Yamaguchi, A. C. Berg, L. E. Ortiz, and T. L. Berg. Who are you with and where are you going? In *CVPR 2011*, pages 1345–1352. IEEE, 2011. → page 4

R. A. Yeh, A. G. Schwing, J. Huang, and K. Murphy. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2019. → page 6

S. Zheng, Y. Yue, and P. Lucey. Generating long-term trajectories using deep hierarchical networks. In *Advances In Neural Information Processing Systems*, pages 1543–1551, 2016. → pages 2, 24, 33

B. Zhou, X. Wang, and X. Tang. Random field topic model for semantic region analysis in crowded scenes from tracklets. In *CVPR 2011*, pages 3441–3448. IEEE, 2011. → page 4