Motion and Goal Conditioned Trajectory Density Estimation

A Project Report for CSN-400A (B.Tech Project) of Autumn Semester 2023-2024

Submitted by

Yash Meena Sunil Chanda (20114106)
yash_msc@cs.iitr.ac.in
Sisir Kumar Padhi (20114092)
sisir_kp@cs.iitr.ac.in
Beauty Raj (20114022)
beauty_r@cs.iitr.ac.in

Supervised by

Prof. Pravendra Singh & Prof. Neetesh Kumar

Department of Computer Science and Engineering
Indian Institute of Technology (IIT) Roorkee

14/11/2023

Candidate's Declaration

I declare that the work carried out in this report entitled "Motion and Goal Conditioned Trajectory Density Estimation" is presented on behalf of the fulfilment of the course CSN-499A submitted to the Department of Computer Science and Engineering, Indian Institute of Technology Roorkee under the supervision and guidance of Prof. Pravendra Singh and Prof. Neetesh Kumar, Dept. Computer Science and Engineering. I further certify that the work presented in this report has not submitted anywhere for any kind of certification or award of any other degree/diploma.

Date: 13/11/2023
Place: IIT Roorkee

Yash Meena Sunil Chanda – 20114106
Sisir Kumar Padhi – 20114092
Beauty Raj - 20114022

Certificate

This is to certify that the above statement made by the candidates is correct to the best of my knowledge and belief.

Date: 13/11/2023
Place: IIT Roorkee

(Signature of the Supervisor)

Candidate's Declaration

I declare that the work carried out in this report entitled "Depth First Search in the Semi-Streaming Model" is presented on behalf of the fulfilment of the course CSN-400A submitted to the Department of Computer Science and Engineering, Indian Institute of Technology Roorkee under the supervision and guidance of Prof. Shahbaz Khan, Dept. Computer Science and Engineering. I further certify that the work presented in this report has not submitted anywhere for any kind of certification or award of any other degree/diploma.

Date: 13/11/2023
Place: IIT Roorkee

Kancharla Nikhilesh Bhagavan - 20114043
Madamanchi Ashok Chowdhary - 20114052
Macharla Sri Vardhan - 20114051

Abstract

Trajectory prediction is not determined just by using past trajectories but also on stochastic factors and environment details. For predicting pedestrian trajectories one must take into account the inherently stochastic nature of human behavior, at any point in time the agent may turn right or keep going straight with equal likelihood. If there are such factors then for forecasting the future actions of an agent we must first estimate where the user wants to go by producing a goal distribution for possible goal positions of an agent taking into account the scene details and the agent's dynamic features which are not just confined to the past velocities but also on walking style, way of dealing with obstacles and any other information from LIDAR this is crucial because many of the existing models do not consider the dynamic constraints and visual information of the environment around them. In this report we propose an upgrade to an existing architecture of a model called FlowChain, a normalizing flow based trajectory prediction model. The upgrade is in the form of an augmentation to the conditioning of the flows by using additional information in the form of a plausible goal position. The new model will be a normalizing flow based trajectory prediction model with goal conditioned input which gives a multi-modal probability distribution for human trajectory prediction. It will allow a rapid analytical computation and reliable fast update of the estimated distribution by incorporating the new observed positions of the agent and reusing the grasped trajectory patter.

Contents

1 Introduction

1.1 Objective

1.2 Motivation

1.3 Problem Statement

2 Previous Work

The field of human trajectory prediction is an import field of study in research and is marked by a diverse range of research that aims to address the indeterministic nature of human motion. Various techniques have been explored to decrease prediction error, including some cutting-edge approaches such as Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), LSTM-based methods[1], transformer-based methods, and more recently, Graph Convolutional Networks (GCNs). Each of these methods brings a unique perspective to the challenge of predicting human trajectories, leveraging advancements in machine learning. In this paper we are going to focus on one such technique Normalizing Flows.

2.1 Maths Behind Normalizing Flows

Below is the advancement we are using, with each successive method surpassing the capabilities of its predecessor. This sequence of advancements is the foundation of our approach to modelling and predicting the future probability distribution of the positions and trajectory.

Figure 1: Evolution Of Normalizing Flows

2.1.1 Normalizing Flow

Within this section, our focus is towards understanding the fundamental concept underlying normalizing flows—a technique employed to construct complex probability distributions by transforming simpler ones using some function f. Let unfold this concept gradually, with step-by-step examples for a comprehensive understanding. Let's suppose we have a continuous random variable z with some simple distribution like a spherical Gaussian or a uniform distribution, etc for easy sampling and density evaluation.

$$z = p_\theta(z) = N(z; 0,\ I) \tag{1}$$

$$\implies \quad x \;=\; f_\theta(z) \;=\; f_k \;\ldots\; f_2 \,.\, f_1(z) \tag{2}$$

The basic intuition is that the distribution of variable x at a point x_o is equal to the distribution of z at the point $f^(-1)(x_0)$, but they are both not equal:

$$p_\theta(x) \;\neq\; p_\theta(f^{-1}(x)) \tag{3}$$

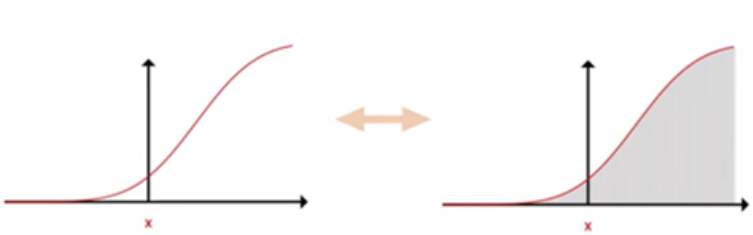

Figure 2: **Effect on Area.** $f : \; z \to x$ With distribution $p_\theta(z)$ defined over $z$in Z. When transforming a probability distribution using a bijective function the total probability remains the same.

$$|p(x) \cdot \Delta x| = |pz \cdot \Delta z| \tag{4}$$

In the limiting case when width approaches 0,

$$\begin{aligned} |p(x)\,dx| &= |p(z)\,dz| \\ p_\theta(x) &= p_\theta(z) * \left|\det\left(\frac{\partial z}{\partial x}\right)\right| \end{aligned} \tag{5}$$

The term inside determinant here is the Jacobian of f^{-1}, this scalar magnitude gives how much the transformation locally expands or contracts space z.
The sequence of bijective transformation is known as normalizing flow.

$_\theta(x) = p_\theta(z) \prod_{i=1}^{k} \left|\det \frac{\partial f_i^{-1}}{\partial z_i}\right| = p_\theta(z) \left|\det\left(\frac{\partial f^{-1}}{\partial z}\right)\right|$

For training/learning the parameters of the function we maximize the log-likelihood,

$\log p_\theta(x) = \log p_\theta(z) + \sum_{i=1}^{k} \log \left|\det \frac{\partial f_i^{-1}}{\partial z_i}\right|$

$\text{maximize}\,(\log\,(p(x))) \iff \text{maximize}(\log p(z) + \log\left|\frac{dz}{dx}\right|)$

The probability distribution of $p_\theta(z)$ is kept as a known distribution, such as uniform or Gaussian. This helps in learning the mapping x to z.

The basic requirements of $f : x \to z$:

- The function must be invertible, as we don't want two points x_1 and x_2 to map to the same z. We would also like to calculate x for a given z, thus requiring a function for which f^{-1} exists. In the 1D case, it's a continuous increasing or decreasing function, for example, a cumulative distribution function.
- We want the determinant of the Jacobian of the function f to be easily computable. If it is on $O(n^2)$, it will square with an increasing number of dimensions.

$|\det \begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} \end{bmatrix}| \neq 0$ and easily computable

2.1.2 RealNVP and conditional flows

We keep the first part of the D dimension vector same i.e., for the first subset we simply apply the identity, copying over the elements to the same positions in the layer output x.To calculate the remaining portion of x, we use two functions m and g. m takes in input the first subset $z_{1:d}$, and the output of m and the second subset $z_{d+1:D}$ serve as the inputs for g. We can look at the Jacobian of this transformation because the elements in the first subset are not affected by elements in the second subset, we end up with lower triangular Jacobian matrix. Because elements in the first subset are left unchanged by the coupling layer, we also need to alternate roles between the two subsets as we compose multiple coupling layers together.

$\rightarrow x_{1:d} = z_{1:d} \Longrightarrow \frac{\partial x}{\partial y} = \text{Id} \quad 0 \frac{\partial x_{d+1:D}}{\partial Z_{1:D}} \frac{\partial x_{d+1:D}}{\partial Z_{1:D}}$

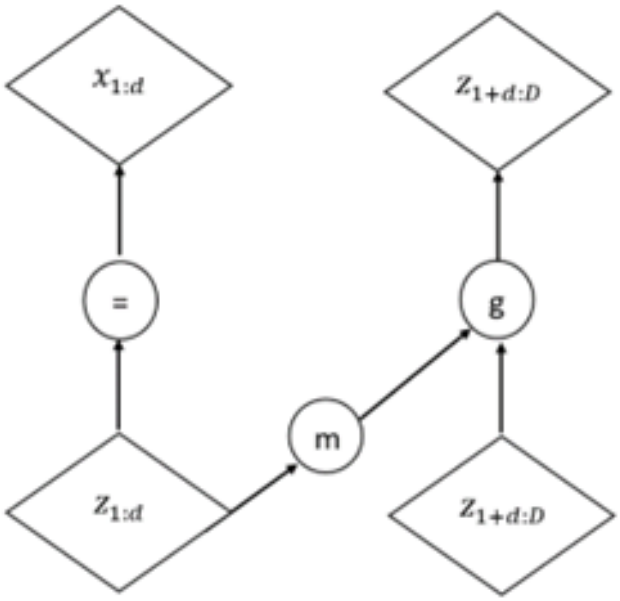

Figure 3: **Coupling Layer.** When following the approach on the above image we get a linear function on determinant of the Jacobian. This is because of the auto-regressive property.

As for inverting this transformation we can invert the identity portion. The second part is the function g, it is required to be invertible with respect to its first argument given the second, since we can trivially recover the second argument. Here m can be as complex as a neural network, the overall transformation is easily invertible, and the Jacobian determinant is simple to compute.

$$z_{1:d} = x_{1:d}$$
$$z_{d+1:D} = g^{-1}(x_{d+1:D}; m(x_{1:d}))$$

For affine coupling the function is shift and scale. We get two variables γ and β using a part of data x_t then scale and shift the remaining part.

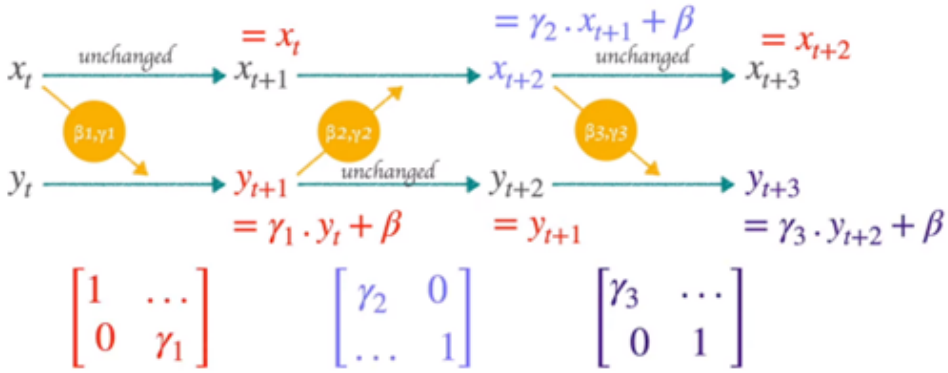

Figure 4: **Layers in RealNVP.**

The values β_1, γ_1, can be generated using x_a by any arbitrary function, in our case we will be using a neural network.

$$y_{t+1} = \beta_t + e^{\log(\gamma_t)} \cdot y_t$$

In case of conditional Affine coupling layer, the affine coupling layer is modified to incorporate additional conditional information, denoted as c. This can be simply done by providing the conditional information c as input to the neural network additional to the partial input x_t.

$$y_{t+1} = y_t \cdot e^{s(x_t,c)} + t(x_t, c) \tag{6}$$

2.2 Social Encoding of Input Feature Vector

We are given a few steps of past trajectory, and we need to find the future steps based on that. We have $N(t)$ number of agents $A_1, A_2, A_3, \ldots, A_n(t)$. Each agent's trajectory is represented as a sequence $(x_0, x_1, x_2, \ldots, x_t)$ of positions $x_t = (x_t, y_t)$ over discrete time steps. At a given time t, we have the history of $T_{o\text{-step}}$ for all agents, i.e., $O_t = \left\{(x^i_{t-T_o-1}, \ldots, x^i_t) \mid i \in (0, N)\right\}$, where i denotes the index of each agent. We have x as the observed position at time t, $\widetilde{x}$ is the predicted future position, and $\hat{x}$ is the ground truth of the future position.

2.2.1 Modeling Agent History

Now we need to encode the agent's information and its interactions with neighboring nodes. To encode the observed history of the modelled agent, their current and previous states are fed into a Long Short-Term Memory (LSTM) network [1] with 32 hidden dimensions. Since we are modelling trajectories, we input O_t iteratively. To keep the model fast and simple, they have used simple single integrators to model the pedestrians.

Encoding agent interactions: To encode the influence of the neighboring pedestrians on our modelled agent, we first create a graph and encode the graph edges. Let the graph be $G = (V, E)$ to represent the current scene at the given time. Nodes represent the agents/pedestrians, and edges represent the interactions between them. An edge (A_i, A_j) is present in E if A_i influences A_j. Here we are using l_2 distance as a base to find whether they have influence on one another or not. In formal terms, an edge is defined from agent A_i to A_j if the Euclidean distance between their 2D world positions, represented by x^i_t and x^j_t, satisfies the condition $||x^i_t - x^j_t||_2 \leq d_S$. d_S is a distance parameter encoding the perception range of the pedestrian.

First, the edge information is gathered from neighboring pedestrians. This gathering involves the element-wise sum. These aggregated states are then input to an LSTM with 8 hidden dimensions, and the weights are the same across all pedestrians, generating an influence vector that signifies the combined effect of all neighboring nodes. Finally, the node history and edge influence encodings are concatenated to get an encoded vector c from past trajectories.

2.3 Flow Chain Explanation and Architecture

3 Our Work

3.1 Reproduction of Results

3.2 New Architecture: A Goal-based Approach

3.3 Fast Update Using Flows

4 Future Work

5 Conclusion

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.