Motion and Goal Conditioned Trajectory Density Estimation

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

Submitted by

Beauty Raj (20114022)

beauty_r@cs.iitr.ac.in

Sisir Kumar Padhi (20114092)

sisir_kp@cs.iitr.ac.in

Yash Meena Sunil Chanda (20114106)

yash_msc@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

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

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)

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.

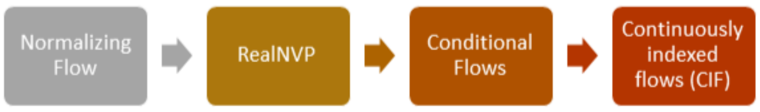


Figure 1: Evolution Of Normalizing Flows

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.

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

$$\implies x = f_\theta(z) = f_k \ldots f_2 \cdot f_1(z) \tag{2}$$

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:

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

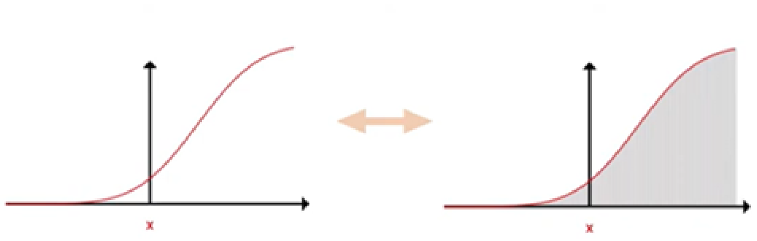


Figure 2: **Effect on Area.** $f : z \rightarrow x$ With distribution $p_\theta(z)$ defined over $z$in Z. When transforming a probability distribution using a bijective function the total probability remains the same.

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

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.

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

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

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

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

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 $f : x \rightarrow z$:

- 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.

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

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.

$$x_{1:d} = z_{1:d}$$

$$x_{d+1:D} = y\left(z_{d+1:D}; m\left(z_{1:d}\right)\right)$$

$$\frac{\partial x}{\partial y} = \left| \det \begin{bmatrix} I_d & 0 \\ \frac{\partial x_{d+1:D}}{\partial Z_{Z1:D}} & \frac{\partial x_{d+1:D}}{\partial Z_{Z1:D}} \end{bmatrix} \right| \tag{6}$$

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.

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.

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.

$$y_{t+1} = y_t \cdot e^{s(x_t, c)} + t(x_t, c) \tag{7}$$

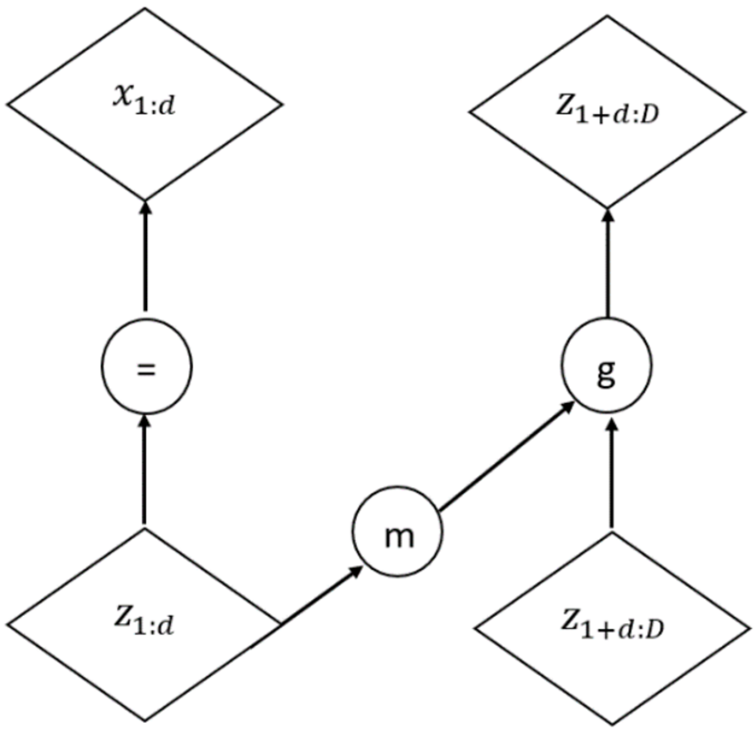


Figure 3: **Coupling Layer.** When following the approach on the above image we get a linear function on determinant of the Jacobian. This is because of the auto-regressive property.

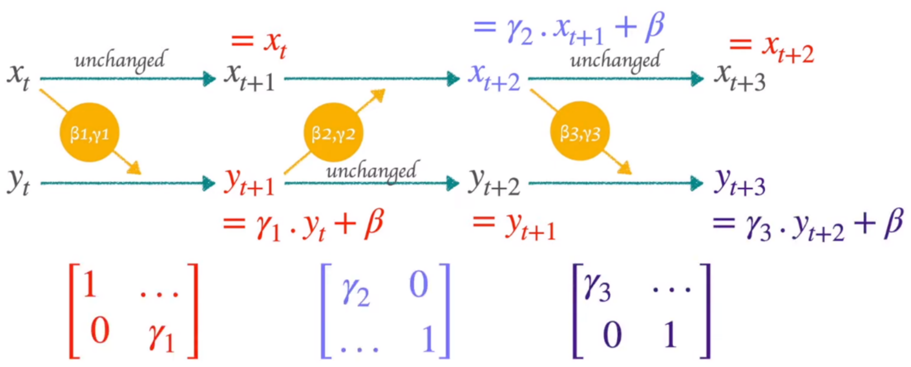
2.2 Social Encoding of Input Feature Vector

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., $O_t = \{(x^i_{t-T_o-1}, \ldots, x^i_t) \mid i \in (0, N)\}$, 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

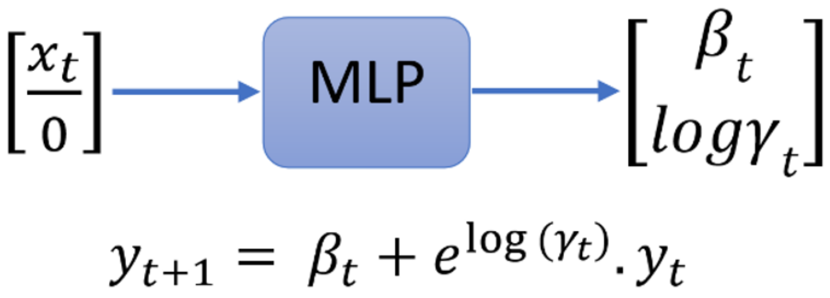
Now we need to encode the agent's information and its interactions with neighboring nodes. To encode the observed history of the modelled agent, their current and previous states are fed into a Long Short-Term Memory (LSTM) network [3] with 32 hidden dimensions. Since we are modelling trajectories, we 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.



x_t
unchanged
x_{t+1}
$= x_t$
$= \gamma_2 . x_{t+1} + \beta$
x_{t+2}
unchanged
x_{t+3}
$= x_{t+2}$
β1,γ1
β2,γ2
β3,γ3
y_t
y_{t+1}
unchanged
y_{t+2}
y_{t+3}
$= \gamma_1 . y_t + \beta$
$= y_{t+1}$
$= \gamma_3 . y_{t+2} + \beta$
$\begin{bmatrix} 1 & \dots \\ 0 & \gamma_1 \end{bmatrix}$
$\begin{bmatrix} \gamma_2 & 0 \\ \dots & 1 \end{bmatrix}$
$\begin{bmatrix} \gamma_3 & \dots \\ 0 & 1 \end{bmatrix}$

Figure 4: **Layers in RealNVP.**



$\left[\frac{x_t}{0}\right]$
MLP
$\begin{bmatrix} \beta_t \\ log\gamma_t \end{bmatrix}$
$y_{t+1} = \beta_t + e^{\log(\gamma_t)} . y_t$

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

Now that all the equations have been covered let us see how the training and testing is done in Flow chain.
Implementation Details:-
The encoding is done as mentioned in the [Section tag] and is named as Temporal social encoder in the figure this gives us c-encoded information on which the flows can be conditioned.

$$y = f(z) \tag{8}$$

$$p(y) = p(z) \left|\det \nabla_y f^{-1}(y, c)\right| \tag{9}$$

The parameter of the invertible function f can be learned by maximizing the likelihood of sample -ground truth from the datasets.

$$NLL = -\log p(\hat{y} \mid c) \tag{10}$$

Density estimation involves transforming previous time step estimation using normalizing flows. At a given time t, we need to find future density $p\left(\widetilde{x}_{t+1}|o_t\right), \ldots, p(\widetilde{x}_{t+T_f}|o_t)$ by using

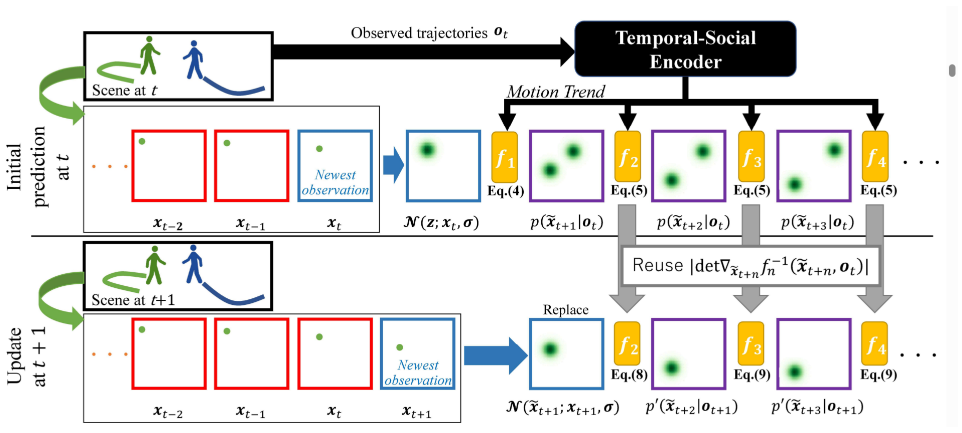


Figure 5: **Architecture of the FlowChain Model.**

chain of conditional CIFs, each transforms as below :-

$$p\left(\widetilde{x}_{t+n-1}|o_t\right) \rightarrow p\left(\widetilde{x}_{t+n}|o_t\right))$$

These are the equation how we update the PDFs. Initially, we take Gaussian density around the latest observation and apply Conditional CIF to estimate density at latter steps:-

$$p\left(\widetilde{x}_{t+1} \mid o_t\right) = \mathcal{N}\left(z; x_t, \sigma\right) \left|\det \nabla_{\widetilde{x}_{t+1}} f_1^{-1}\left(\widetilde{x}_{t+1}, o_t\right)\right| \tag{11}$$

$$p\left(\widetilde{x}_{t+n} \mid o_t\right) = p\left(\widetilde{x}_{t+n-1} \mid o_t\right) \left|\det \nabla_{\widetilde{x}_{t+n}} f_n^{-1}\left(\widetilde{x}_{t+n}, o_t\right)\right| \tag{12}$$

All f_1 to f_n are trained independently using log loss function. From f_n we can sample multiple trajectories for best-of-N metric by sampling multiple z as below:

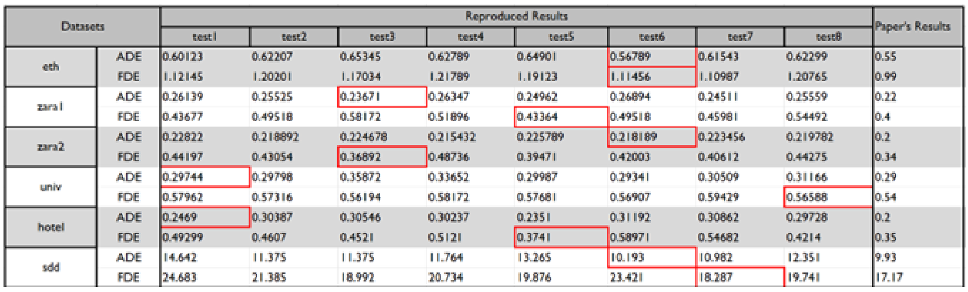
$$\{\widetilde{x}_{t+1}, \ldots, \widetilde{x}_{t+n}\} = \{f_1\left(z, o_t\right), \ldots, f_n \circ \ldots \circ f_2 \circ f_1\left(z, o_t\right)\} \tag{13}$$

3 Our Work

3.1 Reproduction of Results

- **Dataset:** For this we are using ETH/UCY and SDD real image data set which captures pedestrian moments at differ environments. On both the datasets we are given T_o=8 and T_f=12 number of past observed and future predicted steps. These steps have a time interval of 0.4 seconds. In total it consists of 1536 pedestrians in crowded settings with challenging scenarios like group behaviour, people crossing each other, collision avoidance and groups forming and dispersing. This dataset has five scenes: ETH, ZARA1, ZARA2, UNIV, HOTEL. For training and testing we train on 4 scenes and test on the remaining one. All of this is obtained from the Social-GAN [2] .
- In this task First we have gathered all the information about how to do the data preprocessing, training, and testing. After debugging the version control errors, we have run the code for best of 10 and the results have been displayed in Figure 1. The evaluation is

popular metrics Average Displacement Error(ADE) and Final Displacement error (FDE). ADE is the mean l_2 distance between the estimated trajectories and the corresponding ground truth trajectories over time. On the other hand FDE measures the l_2 distance between the end points of the predicted trajectory and the ground truth trajectories.

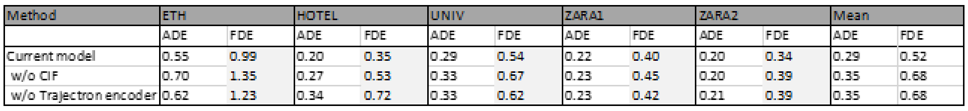


Datasets		Reproduced Results								Paper's Results
		test1	test2	test3	test4	test5	test6	test7	test8	
eth	ADE	0.60123	0.62207	0.65345	0.62789	0.64901	0.56789	0.61543	0.62299	0.55
	FDE	1.12145	1.20201	1.17034	1.21789	1.19123	1.11456	1.10987	1.20765	0.99
zara1	ADE	0.26139	0.25525	0.23671	0.26347	0.24962	0.26894	0.24511	0.25559	0.22
	FDE	0.43677	0.49518	0.58172	0.51896	0.43364	0.49518	0.45981	0.54492	0.4
zara2	ADE	0.22822	0.218892	0.224678	0.215432	0.225789	0.218189	0.223456	0.219782	0.2
	FDE	0.44197	0.43054	0.36892	0.48736	0.39471	0.42003	0.40612	0.44275	0.34
univ	ADE	0.29744	0.29798	0.35872	0.33652	0.29987	0.29341	0.30509	0.31166	0.29
	FDE	0.57962	0.57316	0.56194	0.58172	0.57681	0.56907	0.59429	0.56588	0.54
hotel	ADE	0.2469	0.30387	0.30546	0.30237	0.2351	0.31192	0.30862	0.29728	0.2
	FDE	0.49299	0.4607	0.4521	0.5121	0.3741	0.58971	0.54682	0.4214	0.35
sdd	ADE	14.642	11.375	11.375	11.764	13.265	10.193	10.982	12.351	9.93
	FDE	24.683	21.385	18.992	20.734	19.876	23.421	18.287	19.741	17.17

Figure 6: **Reproduced Results based on Best-of-8 metrics.** The red borders highlight the best reproduced values for each dataset. The paper's results column contain best of 20 metrics as cited in the paper.

3.2 Ablation Study

Three main parts of the architecture of the FlowChain, Trajectron++ encoder as the temporal-social encoder, CIF and three-layer RealNVP [1] inside CIF. To substitute continuously indexed flows (CIF) in the approach, we used normal conditional normalizing flow models, indicated as "w/o CIF". To test the importance of trajectron++ encoder where we have also considered the social interactions we used a transformer layer instead to encode the observed past trajectory, labeled as "w/o trajectron encoder." The main difference is transformer layer independently encodes each trajectory and because of that it does not account for social interactions.



Method	ETH		HOTEL		UNIV		ZARA1		ZARA2		Mean	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
Current model	0.55	0.99	0.20	0.35	0.29	0.54	0.22	0.40	0.20	0.34	0.29	0.52
w/o CIF	0.70	1.35	0.27	0.53	0.33	0.67	0.23	0.45	0.20	0.39	0.35	0.68
w/o Trajectron encoder	0.62	1.23	0.34	0.72	0.33	0.62	0.23	0.42	0.21	0.39	0.35	0.68

Figure 7: **Results of the Ablation Study.**

3.3 New Architecture: A Goal-based Approach

With the aim of improving the results of FlowChain, we decided to augment the model's architecture with a goal-based approach by including a goal position g in the conditional information input parameter c in the conditional CIFs.

3.3.1 Generating a viable goal position g via a Goal Module

The Goal Module combines the environment details and the velocity from the past trajectories to estimate a probability distribution over the possible goal positions, which is then used to sample a goal position g. The environment details are represented as an RGB image or a semantic map of $H \times W$, captured from above. This image is input to the GM. Inside the GM, an encoder-decoder CNN with skip connections receives the scene image, where the scene image features are concatenated with the motion features like velocity from the Motion Encoder.

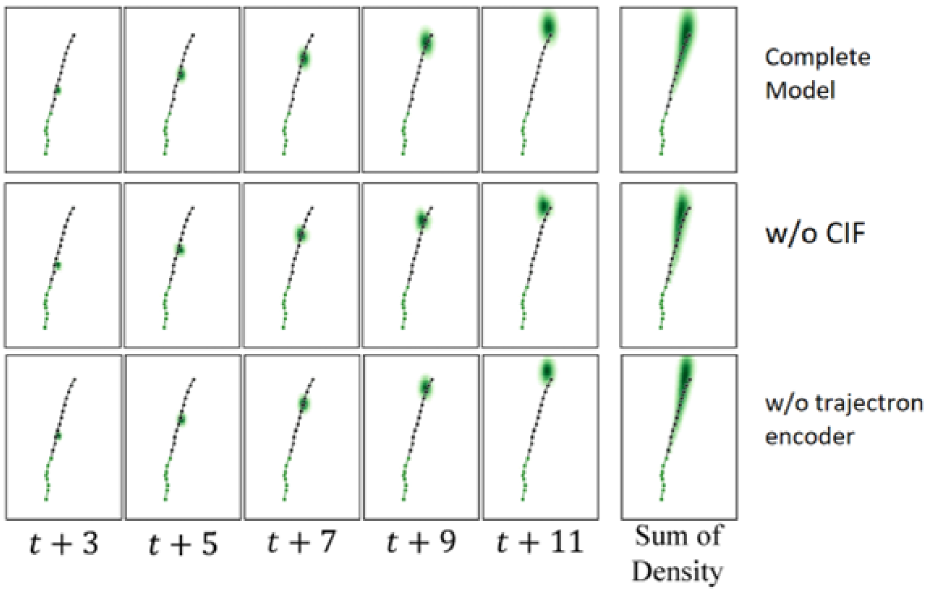


Figure 8: **Temporal estimated densities of ablated models on ETH/UCY dataset.**

The CNN analyzes the past trajectories and the image and outputs a probability map that reflects the different modes captured by associating the agent trajectories with the scene. The CNN decoder outputs a discrete probability distribution map $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ where each α_i is the probability of a particular cell being the end position of the agent. This discrete distribution is then used to sample a plausible goal position by using the Gumbel-Softmax-Trick [insert link]. This allows us to sample the distribution but also backpropagate the loss through the random process of sampling a two-dimensional goal position.

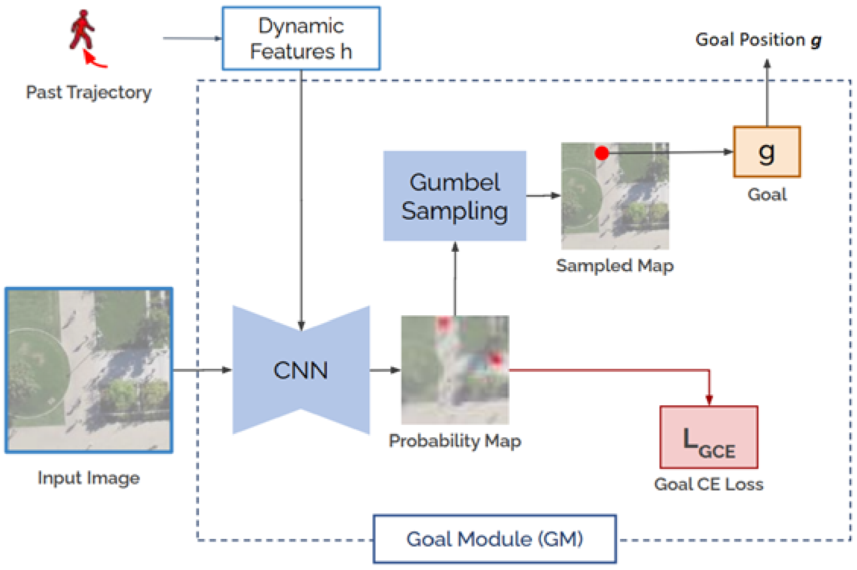


Figure 9: **Goal Module Architecture**

3.3.2 Incorporating the Goal Module in FlowChain

Before the CIFs were only conditioned on condition vector c which inherently represented the observed trajectories and their social interactions till now as of the previous To time steps, but now in the upgraded model, we also generate a goal position and condition the flows on both the past trajectories and the estimated goal position. The Goal Module's architecture is shown below, the output of the GM is a goal g which in conjunction with c gives c'.
And the conditioning vector c is also upgraded to c' via the following equation:

$$c' = c + g \tag{14}$$

c is the original condition vector based solely on the observed trajectories.

c' is the updated c on which the CIFs will be conditioned.

g is the sample goal position from the estimated goal density distribution.

3.3.3 Why augmenting a goal position to the model's input information may improve results?

In the previous model which simply conditioned the normalizing flows on the observed trajectories o_t, the predicted trajectories are based solely on past observations, simply deducing the future trajectory from this information would generate generic trajectories and as a result imprecise paths because turning left, right or going straight are all equally probable outcomes. This is also due to the fact that human trajectories depend primarily on their intention which is inherently unknown to an external observer.
Therefore, a more reliable method would be to first capture the multi-modality of the task, that is generate a potential future goal position distribution which would account for the diverse modes present in the scene and then sample a likely goal position from it. By modes we mean different types of factors which would impact a pedestrian's trajectory such as velocity, manner of walking, unique solutions for dealing with obstacles and certain external factors specific to any particular agent.
As we are now capturing the stochastic nature of trajectory prediction in more detail than before the results may improve, but it still greatly depends on how we go about implementing it.

3.4 Fast Update Using Flows

One of the key highlights of this model is the extremely fast update using the nature of the motion so far and the most recent observed position. These two factors are reliable for accurately predicting the next position because, firstly the nature of the motion cannot change suddenly because an agent moving straight in the current time step will keep moving straight in the immediate next timestep, as the individual time steps are momentary, and we don't consider the action of impulsive forces. Secondly if we make the next prediction without the information provided by the most recent observed position then it is noticed that the accuracy of density estimation degrades along with the number of updates because it has to extrapolate the missing trajectory of the previous timestep, which leads to stochastic unpredictable outcome. Hence, when we use the prolonged nature of the motion observed till now along with the most recent

observed position, we can reliably predict the next position as well as rapidly.
The next question is what effect will the augmentation of the Goal Module have on this update process? To answer this question, we first need to recognize the fact that agents or the humans in a pedestrian prediction context usually first set a goal which they want to reach and then they formulate a route to the goal and dealing with the obstacles that arise on the way as they come but the goal doesn't change. So, this implies that the trajectory of a person is primarily towards the goal and the goal mostly doesn't change so we can reuse the previously computed Jacobians based on the nature of the motion and the newest observed position. Below shows how this way of update differs from other models.

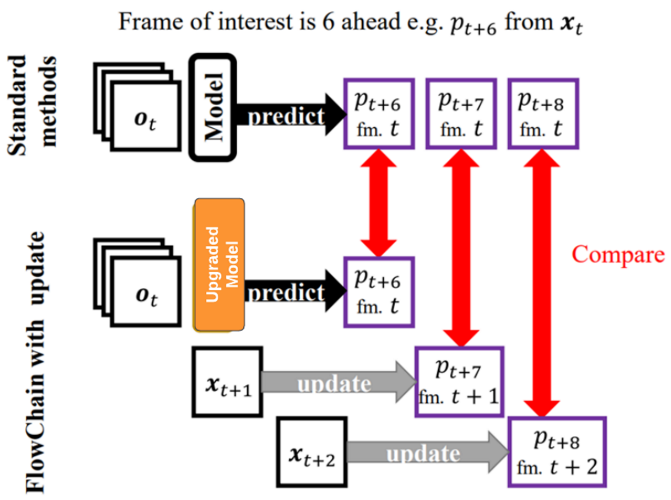


Figure 10: **Update Process of the upgraded model vs that of other models.**

The figure demonstrates that for predicting the distribution p at $t = t + 7$, we don't simply use $\mathbf{o}_t$ but rather we use $\mathbf{o}_t$ along with the newest observed position x_{t+1}.

3.4.1 How is the probability density updated?

We transform the most recent observed position using the motion trend and model it in each flow $f_n(\widetilde{x}_{t+n}, \mathbf{o}_t)$. For the next update, we first convert the newest observed position x_{t+n} at $t + n$ by converting the estimated density $p(\widetilde{x}_{t+n} \mid \mathbf{o}_t)$ with the Gaussian density $\mathcal{N}(x_{t+n}, \sigma)$. This replacement propagates the information of the newest observed position to the latter steps.

Also, as the motion trend is reused in each flow transformation $f_n(\widetilde{x}_{t+n}, \mathbf{o}_t)$, we can reuse the log-det-jacobians $\log \left|\det \nabla_{\widetilde{x}_{t+n}} f_n^{-1}(\widetilde{x}_{t+n+1}, o_t)\right|$, which is the variation of the estimated densities and is not needed to be evaluated again, hence omitting great computation cost.

The equation of FlowChain's update is expressed as:

$$p'\left(\widetilde{x}_{t+2} \mid o_{t+1}\right) = \mathcal{N}\left(\widetilde{x}_{t+1}; x_{t+1}, \sigma\right) \left|\det \nabla_{\widetilde{x}_{t+2}} f_2^{-1}(\widetilde{x}_{t+2}, o_t)\right| \tag{15}$$

$$\begin{aligned} p'\left(\widetilde{x}_{t+n} \mid o_{t+1}\right) &= p'\left(\widetilde{x}_{t+n-1} \mid o_{t+1}\right) \left|\det \nabla_{\widetilde{x}_{t+n}} f_n^{-1}(\widetilde{x}_{t+n})\right| \\ &= \exp\left(\log \mathcal{N}\left(\widetilde{x}_{t+1}; x_{t+1}, \sigma\right) + \sum_{n'=2}^{n} \log \left|\det \nabla_{\widetilde{x}_{t+n'}} f_{n'}^{-1}(\widetilde{x}_{t+n'+1}, o_t)\right|\right) \end{aligned} \tag{16}$$

4 Future Work

5 Conclusion

References

[1] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. 2016. cite arxiv:1605.08803Comment: 10 pages of main content, 3 pages of bibliography, 18 pages of appendix. Accepted at ICLR 2017.

[2] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. *CoRR*, abs/1803.10892, 2018.

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