Social ODE: Multi-Agent Trajectory Forecasting with Neural Ordinary Differential Equations

Song Wen, Hao Wang, and Dimitris Metaxas

Rutgers University
song.wen@rutgers.edu, hoguewang@gmail.com, dnm@cs.rutgers.edu

Abstract. Multi-agent trajectory forecasting has recently attracted a lot of attention due to its widespread applications including autonomous driving. Most previous methods use RNNs or Transformers to model agent dynamics in the temporal dimension and social pooling or GNNs to model interactions with other agents; these approaches usually fail to learn the underlying continuous temporal dynamics and agent interactions explicitly. To address these problems, we propose Social ODE which explicitly models temporal agent dynamics and agent interactions. Our approach leverages Neural ODEs to model continuous temporal dynamics, and incorporates distance, interaction intensity, and aggressiveness estimation into agent interaction modeling in latent space. We show in extensive experiments that our Social ODE approach compares favorably with state-of-the-art, and more importantly, can successfully avoid sudden obstacles and effectively control the motion of the agent, while previous methods often fail in such cases.

Keywords: Multi-Agent Modeling, Ordinary Differential Equations, Social ODEs

1 Introduction

The goal of multi-agent trajectory forecasting is to estimate future agent trajectories given historical trajectories of multiple agents. It has drawn much attention because of its widespread applications such as autonomous driving, urban data mining, path planning and traffic flow forecasting.

Multi-agent trajectory forecasting is a challenging problem because agent interactions (relational dimension) and underlying agent temporal dynamics (temporal dimension) jointly affect each agent in a nonlinear and complex way. By modeling the relational and temporal dimensions, previous deep learning approaches have shown to be promising. They often use graphs, social pooling, or spatial Transformers to model the relational dimension, while they apply RNNs or temporal Transformers to encode the temporal dimension. However, these methods usually fail to learn the underlying continuous temporal dynamics and the agent interactions with other agents explicitly. For example, spatial Transformers estimate the attention between any two agents, but the attention can not explain how one agent affects the other and does not incorporate agent information explicitly, such as distance with other agents. Moreover, RNNs recurrently update the hidden state discretely as shown in Fig. 1a, which is a limitation, because the agent trajectory is continuous as shown in Fig. 1b. These modeling limitations often lead to inaccurate and unsatisfactory results, such as reduced forecasting accuracy and collisions among agents.

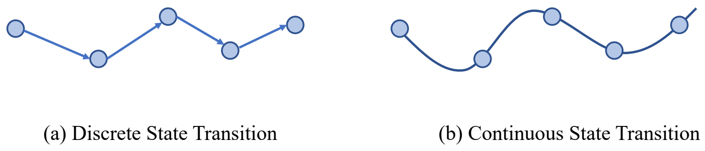

(a) Discrete State Transition
(b) Continuous State Transition

Fig. 1. Differences between discrete state transitions and continuous state transitions

To overcome the above limitations by previous methodologies, we propose Social ODE to explicitly model nonlinear agent interactions and agent temporal dynamics. In our Social ODE framework, the next position of each agent is determined based on the previous position and velocity. The agent's position and velocity are affected by other agents. For example, when we drive a vehicle (agent), if another vehicle approaches dangerously close, we tend to decrease or increase our vehicle's velocity and potentially change direction. Additionally, the distance between vehicles and the driver's driving habits determine how the vehicle's velocity changes. To incorporate into our Social ODE these real-world agent behaviors, we encode in latent space the real-world trajectories and we model them based on an Ordinary Differential Equation as follows:

$$\frac{dh(t)}{dt} = g(h(t), t),\ \ h(t) = h(0) + \int_0^t g(h, t)dt, \tag{1}$$

where h(t) is the state of the agent's latent trajectory at time t. Therefore, $g(h,t)$ models the current state and the nonlinear interactions with other agents.

The proposed Social ODE is an encoder-decoder architecture based on VAEs, where the encoder projects an agent's historical trajectories to latent space and the decoder recovers the historical trajectories and forecasts future agent trajectories using the latent space representation. To model and learn the agent's continuous latent trajectory from the historical trajectories, we use a Neural ODE that learns the underlying temporal dynamics of the agent's continuous trajectory. The agent's temporal dynamics are determined by the current state and the agent's interactions with other agents. To model the agent interactions explicitly (relational dimension), we decouple them into three components: distance, interaction intensity, and aggressiveness information. All three are multiplied to model their impact on the temporal dynamics of each agent. Because our Social ODE models the relational and temporal dimensions explicitly, an agent can also avoid collisions with other agents. In addition, using repellers and attractors, we can modify an agent's ODE to model more effectively an agent's trajectory, behavior (e.g., courageous) and goals.

The main contributions of our paper are the following:

Model an agent's trajectory relational and temporal dimensions explicitly: Our proposed Social ODE framework models the temporal dimension using a Neural ODE to learn an agent's trajectory continuous temporal dynamics in latent space, which are determined by the agent's current state and the interactions with other agents. We model agent interactions using the following three variables: distance with other agents, agent interaction intensity, and agent aggressiveness.

Effective agent trajectory control without retraining: We demonstrate how to modify the ODE to effectively control the trajectory of an agent using attractors and repellers. This allows the modification of an agent's trajectory without retraining. Using our approach we can model dynamic environments where new obstacles and attractors can appear dynamically.

Extensive experimental study: We conduct extensive experiments on several datasets by comparing our Social ODE methodology with other state-of-the-art approaches. We demonstrate that our Social ODE achieves improved accuracy on complex trajectory forecasting. We demonstrate its effectiveness in reducing agent collision rates in dynamic environments without retraining.

2 Related Work

Neural Ordinary Differential Equations. In [3], Neural ODE, a new class of deep learning model is proposed, which is a continuous-time neural network by solving ODEs. Following their work in modeling continuous-time sequences, Latent ODE [23] is proposed to model the irregularly-sampled time series. ODE2VAE [29] models high-dimensional sequences by a latent second order ODE. Dupont et al. propose Augmented neural ODE [5] to make the model more expressive by preserving the topology of the input space. To model non-continuous observations using Neural ODEs, Brouwer et al. propose GRU-ODE-Bayes [4]. Moreover, [6, 28, 14, 20, 22, 13] analyze and adapt Neural ODEs in other applications such as density estimation. [25, 19, 27] apply Neural ODEs in trajectory modeling or planning. Grunbacher et al. analyze the verification of Neural ODEs [9, 10]. Park et al. generate continuous-time video by Neural ODE [21].

Inspired by these approaches, we propose Social ODE based on Latent ODE to model the realistic trajectory and underlying temporal dynamics of the latent trajectory. Similar to Latent ODE, our model also implements trajectory interpolation and extrapolation. The difference is that our model encodes the agent interactions in the ordinary differential equation.

Multi-Agent Trajectory Forecasting. Social LSTM [1] is proposed by Alahi et al., which applies social pooling in the hidden state of LSTM. Following Social LSTM, Gupta et al. propose Social GAN [12], which uses global social pooling and GAN to generate a trajectory consistent with the input. Graph-VRNN [26] proposed by Sun et al. adopt graph network and RNN to model the relational dimension. Kipf et al. represent underlying agent interaction in latent space by graph [15]. Based on their work, Graber develop dynamic neural relational inferece [8], instead of static relation in [15]. Trajectron++ [24] is a graph-structured model with LSTM and accounts for environmental information. EvolveGraph [18] forecasts the trajectory by dynamic relational reasoning by latent interaction graph. AgentFormer [30] proposes a novel Transformer to joint model social and temporal dimensions. Gu et al propose DenseTNT [11] based on VectorNet [7] to encode all agent to vectors and graph.

Different from previous methods, our proposed Social ODE learns the underlying agent trajectory temporal dynamics in latent space using an ODE. The advantage of our approach is that it explicitly models the continuous-time agent trajectory which offers explainability. Besides, we model the relational dimension by incorporating agent distance, interaction intensity, and aggressiveness explicitly in the ODE.

3 Methodology

In this section, we first define the problem of trajectory forecasting. Then we present an overview of our proposed Social ODE and provide details of the formulation of the associated encoder, decoder, and loss function. Finally, we present agent trajectory control without retraining using Social ODEs.

3.1 Trajectory Forecasting

Multi-agent trajectory forecasting aims to estimate the future trajectories of multiple agents $\mathbf{X}_{T_h+1:T_h+T_f} = \{\mathbf{x}^i_{T_h+1}, \mathbf{x}^i_{T_h+2}, ..., \mathbf{x}^i_{T_h+T_f}, i = 1, ..., N\}$ simultaneously giving their historical trajectories $\mathbf{X}_{0:T_h} = \{\mathbf{x}^i_0, \mathbf{x}^i_1, ..., \mathbf{x}^i_{T_h}, i = 1, ..., N\}$, where N denotes the number of agents. T_h and T_f denote the historical and future trajectory temporal lengths, respectively. X_t denotes the state of all agents at time t and $\mathbf{x}^i_t$ denotes the state of agent i at time t, including its position and velocity.

3.2 Social ODE: Overview

Similar to Latent ODEs, which is a continuous-time, latent-variable method to model time series, our Social ODE is also an Encoder-Decoder architecture. As shown in Fig. 2, it concludes two components:

Encoder. It encodes the historical trajectory for each agent $\mathbf{X}^i_{0:T_h}$ into latent space. The encoder generates the initial state in latent space, which is set as the initial value for the ordinary differential equation. Different from a Latent ODE, we use a Spatio-Temporal Transformer as the encoder for improved learning.

Decoder. It generates latent trajectories and decodes the latent vector back to the real-world state, i.e., the agent's trajectory position and velocity. After sampling the latent vector from the encoder, we design an ODE to model the agent's interactions in the decoder. The agent's interactions are modeled by incorporating distance, interaction intensity, and aggressiveness explicitly in the ODE, while we model the temporal dimension based on the current latent state. Using the initial state in latent space, we solve the ODE to generate latent trajectories. At the end of our approach, the latent trajectory is converted to a real-world trajectory.

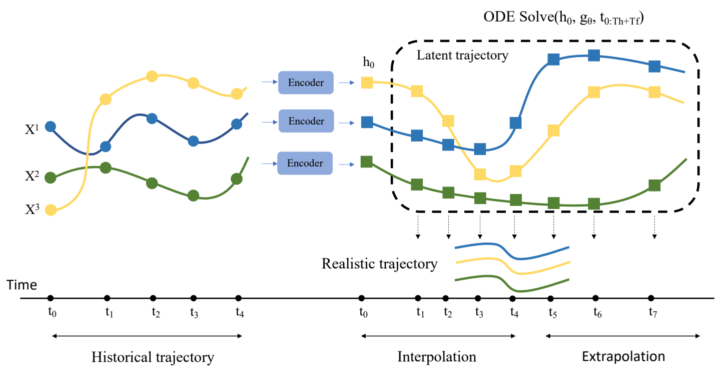


Fig. 2. Overview of the proposed Social ODE, which is composed of an encoder and a decoder. The encoder transfers the historical trajectory to latent space. The decoder first uses the ODE solver to generate a latent trajectory and then recover it back to a realistic trajectory. The output includes historical trajectory (interpolation) and future trajectory (extrapolation).

Using our approach, an agent's trajectory i, is modeled as:

$$\mu_{h^i}, \sigma_{h^i} = g_{enc}(x^i_{0:T_h}, x^j_{0:T_h}),\ j \neq i \tag{2}$$

$$\mathbf{h}^i_0 \sim N(\mu_{h^i}, \sigma_{h^i}), \tag{3}$$

$$\mathbf{h}^i_0, \mathbf{h}^i_1, ..., \mathbf{h}^i_{T_h+T_f} = \text{ODESolve}(\mathbf{h}^i_0, g_\theta, t_{0:T_h+T_f}) \tag{4}$$

$$\text{each } \hat{x}^i_t \sim p(\hat{x}^i_t|\mathbf{h}^i_t), \tag{5}$$

where eq.(2) is the encoder and eqs.(3) to (5) model the decoder. ODESolver is the numerical ODE solver given equation $\frac{dh}{dt} = g_\theta$ with initial value $\mathbf{h}^i_0$.

3.3 Encoder: Spatio-Temporal Transformer

To encode the historical trajectory for each agent to latent vectors, we use a Spatio-Temporal Transformer for each agent and the architecture is shown in Fig. 3.

Spatial Transformer. It is used to encode the relational dimension. Because the state of agent i at time t is only affected by states of other agents before time t, to reduce computation, we only take into account the states of other agents in time $t-1$ and t, which is shown in Fig. 3.

Temporal Transformer. After encoding the relational dimension in each time step, the new state sequence for each agent is generated. We use a Temporal Transformer to encode the generated state sequence for each agent and pool them to generate a latent vector.

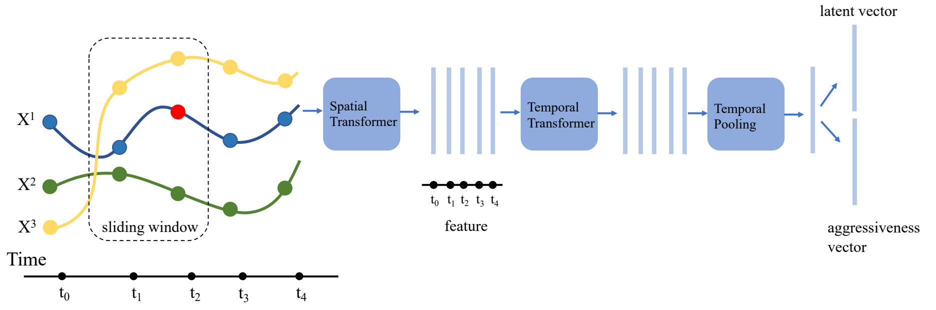


Fig. 3. The architecture of Spatio-Temporal Transformer. The figure shows that we apply Spatio-Temporal Transformer for trajectory X^1. When the red point is modeled, we improve the efficiency of the algorithm by using points only within the sliding window, which are states of other agents in the previous and current time steps.

3.4 Decoder

We use the decoder to recover the real-world trajectory from the initial value h_0. There are two steps in the decoder: solving an ODE and transferring the latent state to the real-world state.

In the Latent ODE model [3], after estimating the posterior $p(h_{t_0}|x_{0:t})$, the initial value h_{t_0} is sampled. These initial values are used to solve the corresponding ODE. However, the relational dimension is ignored in the standard Latent ODE formulation. To model agent interactions and improve agent prediction trajectory, we encode state sequences of each agent in the latent space and represent agent interaction in latent space using three variables: distance, interaction intensity and aggressiveness. We define the equation as:

$$\frac{dh_i(t)}{dt} = g_\theta(h_i(t), h_j(t)) \tag{6}$$

$$= \sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + f_\theta(h_i(t)), \tag{7}$$

where $h_i(t)$ denotes the latent vector of agent i in time t and g_θ is the derivative of the latent vector. Besides, $||h_i - h_j||$ denotes distance information between agent i and agent j, while $k(i,j)$ is the interaction intensity between two agents. a_i denotes the aggressiveness of agent i.

The agent interaction is modeled based on the following three components:

Interaction Intensity. It models how agent j affects the dynamics of agent i, which is denoted by $k(i,j)$. We concatenate the latent vectors of two agents (h_t^i and h_t^j) and the derivatives of two agents in the previous time step ($\frac{dh_t^i}{dt}$ and $\frac{dh_t^j}{dt}$), and apply a fully connected neural network to estimate $k(i,j)$.

Distance. It is obvious that the distance between two agents has a great influence on each other and the shorter distance between two agents means the greater influence. We represent this relationship explicitly in latent space, which is $\frac{1}{||h_i - h_j||}$, where i denotes the agent that is modeled and j denotes other agents.

In latent space, the L2 distant $||h_i - h_j||$ of two agent contains realistic distance information. When agent j come to i, the dynamics of agent i are affected, so $\frac{dh_i(t)}{dt}$ becomes larger.

Aggressiveness. In real-world situations, some agents tend to refuse to avoid other agents and others do not. Therefore, besides the distant information, the aggressiveness of an agent should also be incorporated. The aggressiveness can also be learned from the historical trajectory of the agent. As shown in Fig. 3 we use other fully connected networks before generating a latent vector in the encoder to estimate the aggressiveness vector.

In the equation, interaction intensity, distance and aggressiveness are element-wise multiplied together as an agent interaction term. Besides agent interaction, the previous state is also essential to learning temporal dynamics. Similar to standard latent ODE, we use fully connected networks in the temporal modeling, which is denoted by $f_\theta(h_i(t))$ in eq.(7). We add the agent interaction and feature of the current state together as the derivative of latent vector $h(t)$. By an ODE Solver, $h_{t_1}, h_{t_2}, .., h_{t_n}$ are estimated. Then we use fully connected neural networks to decode latent vectors to realistic states $x_{t_1}, x_{t_2}, .., x_{t_n}$ for each agent.

3.5 Loss Function

Because our method is based on the VAE model, we use the negative evidence lower bound (ELBO) in our loss function:

$$L_{elbo} = -E_{q_\phi(h_{t_0}|X_{0:T_h})}[\log p_\theta(X_{0:T_h}|h_{t_0})] + KL(q_\phi(h_{t_0}|X_{0:T_h})||p_\theta(h_{t_0})), \quad (8)$$

where $q_\phi(h_{t_0}|X)$ is the posterior distribution and $p_\theta(X_{0:T_h}|h_{t_0})$ denotes the interpolation period that recovers the historical trajectory.

Because ELBO only takes into account historical trajectories, the MSE loss is used to supervise prediction accuracy, which is the extrapolation period:

$$L_{mse} = \sum_i (\hat{y}_i - y_i)^2. \quad (9)$$

In training, the whole trajectory $X_{0:T_h+T_f}$ is input to the Social ODE. The output trajectory of input $X_{0:T_h+T_f}$ and input $X_{0:T_h}$ should be the same because they recover the same trajectory. Therefore, their latent vectors have the same distribution. Then we use another KL divergence as a loss function:

$$L_{kl} = KL(q_\phi(z_{t_0}|X_{0:T_h})||q_\phi(z_{t_0}|X_{0:T_h+T_f}))). \quad (10)$$

Consequently, the overall loss function is

$$L = L_{elbo} + L_{mse} + L_{kl} \quad (11)$$

3.6 Agent Controlling with Social ODE

Apart from forecasting future trajectories, our proposed Social ODE can also control a given agent's trajectory, by adding a term to the ODE based on repellers and attractors without the need for retraining. This control approach enables us to model real-world situations where obstacles (other agents) and agent goals can change in real-time and require the modification of the agent's trajectory.

Attractor and Repeller. Because our proposed Social ODE explicitly models the relational dimension, it is not hard to control an agent by modifying the relational dimension dynamically due to real-time changes in the other agents, obstacles and goals. We do this by adding terms modeling these changes as attractors and repellers. If we want to set one attractor (e.g., a location where the agent wants to reach), the ODE eq.(7) can be modified as follows:

$$\frac{dh_i(t)}{dt} = \sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + f_\theta(h_i(t), t) - \lambda(h_i(t) - h_g), \tag{12}$$

where h_g denotes the latent vector of the attractor and λ is a positive coefficient.

We show below that this modeling of an attractor dynamically reduces the distance between the agent and the goal (attractor). The distance information in latent space between the agent and the attractor is modeled as $(h_i(t) - h_g)^2$. We can prove that this distance keeps getting smaller over time by examining its time derivative as follows:

$$\begin{aligned} \frac{d(h_i(t) - h_g)^2}{dt} &= 2(h_i(t) - h_g) \times \frac{dh_i(t)}{dt} \\ &= 2(h_i(t) - h_g)[\sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + f_\theta(h_i(t), t)] - 2\lambda(h_i(t) - h_g)^2. \end{aligned} \tag{13}$$

Since λ is positive then the term is negative: $-2\lambda(h_i(t) - h_g)^2 < 0$. Therefore, if λ is large enough, $\frac{d(h_i(t)-h_g)^2}{dt} < 0$, which means the distance between the agent and the attractor decreases as time goes by.

Similarly, to add many attractors and repellers the ODE eq.(12) is further modified as follows:

$$\begin{aligned} \frac{dh_i(t)}{dt} =& \sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + f_\theta(h_i(t), t) + \sum_n (-\lambda_n (h_i(t) - h_g^n)) \\ &+ \sum_m (\lambda_m (h_i(t) - h_g^m)), \end{aligned} \tag{14}$$

where $(h^m)_g$ denotes the latent vector of a repeller and λ_m is a positive coefficient.

Adjusting Agent Interactions. To adjust the strength of agent interactions, we further modify the first two terms in eq.(14) by introducing two new parameters β_1 and β_2 to adjust the dynamics of agent interactions as shown in the following equation. For example, if β_1 is small and β_2 is large, the agent will be more aggressive and take less into account the other close-by agents.

$$\beta_1 \sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + \beta_2 f_\theta(h_i(t)). \tag{15}$$

Agent Return to Desired Trajectory. We modeled obstacle avoidance (different from agent interactions) as repellers. However, the influence of repellers can make the agent deviate from the desired trajectory. In order to ensure the agent returns to the desired trajectory after obstacle avoidance, we add one more term to eq.(14) as follows:

$$-\lambda \min_{\tilde{h}}(h_i(t) - \tilde{h}), \tag{16}$$

where $\tilde{h}_i(t)$ is the original trajectory prior to obstacle avoidance and the term $\min_{\tilde{h}}(h_i(t) - \tilde{h})$ is an attractor and ensures that the agent always returns to the closest point of the original trajectory.

After adding bother terms to eq.(14) the final equation is:

$$\begin{aligned}\frac{dh_i(t)}{dt} = &\sum_{j \neq i} \frac{1}{||h_i - h_j||} k(i,j) a_i + f_\theta(h_i(t), t) + \sum_n (-\lambda_n (h_i(t) - h_g^n)) \\ &+ \sum_m (\lambda_m (h_i(t) - h_g^m)) - \lambda \min_{\tilde{h}}(h_i(t) - \tilde{h}),\end{aligned} \tag{17}$$

4 Experiments

In this section, we present experimental results to evaluate the performance of our novel Social ODE framework. In 4.1 and 4.2, we present the training and test datasets and implementation details, respectively. Then we show comparison results with the state-of-the-art methods in 4.3. We present the agent control without the need for retraining in 4.4. Finally in 4.5, we conduct the ablation study.

4.1 Datasets

Our model is evaluated on the inD [2], rounD [17], and highD [16] traffic datasets. Those are datasets of naturalistic road user trajectories collected by a drone. For each dataset, 80 percent of the data are used for training and validation, and 20 percent are used for testing. We sample every 8 seconds as one instance and delete the case where some agents leave the area in the middle. In each trajectory, we sample one point every 0.4 seconds, so there are 20 points for agents which are present all the time. The trajectories in the first 4 seconds are used as input and those in the next 4 seconds are ground truth.

4.2 Implementation and Training Details

We normalize all the coordinates to range from 0 to 1. In the encoder module, the dimension of key, query and value is set to 128 and the number of heads is set to 8. Because there is no sequence information in the Spacial Transformer, positional encoding is only used in the Temporal Transformer. In the decoder module, the dimension of the latent vector is also 128. To model interaction intensity, a 512 × 128 fully connected network is used. We also use a 128 × 128 fully connected network to replace the last layer of the encoder to generate the aggressiveness vector. In the decoder process, there are two parts: interpolation and extrapolation. While inputting a historical trajectory $X_{0:T_h}$, our model will generate $\hat{X}_{0:T_h+T_f}$. The process of generating $\hat{X}_{0:T_h}$ is the interpolation, which is recovering the input like a VAE model, and that of $\hat{X}_{T_h+1:T_f}$ is the extrapolation, which estimates the prediction.

In the training phase, the Adam optimizer is used and the learning rate is set initially to 10^{-4}, which is then decayed by a factor of 0.5 when the loss comes to a plateau. We train the model on A100 GPUs using PyTorch.

4.3 Comparison Results

Evaluation Metric. We evaluated our model by ADE (Average Displacement Error), which is defined as

$$ADE = \frac{1}{T}\sum ||x_t^i - \hat{x}_t^i||, \tag{18}$$

where x_t^i is the ground truth and $\hat{x}_t^i$ is the prediction in extrapolation. ADE is used to evaluate the mean square error between the ground truth and the prediction.

Baseline. We compare our Social ODE with several state-of-the-art methods: (1) Social LSTM [1]: Social pooling of hidden states in LSTM is used to model agent interactions. (2) Social GAN [12]: GAN is combined with LSTM encoder-decoder to judge whether the generated trajectory is similar to the realistic generated trajectory. (3) DenseTNT [11]: The graph is used to extract the relationship among different agents and each node in the graph is an agent's trajectory. (4) AgentFormer [30]: It is a Socio-Temporal Transformer encoder-decoder model to jointly extract the time dimension and social dimension. The codes of all the above methods have been published, so we directly train and evaluate these models on inD, rounD and highD traffic datasets.

As shown in Table 1, Social ODE achieves better performance than Social LSTM, Social GAN, and DenseTNT. We classify the trajectory into the 'curve' and 'straight' classes. From Table 1, if forecasting is for a long period of time, Social ODE always performs best in curved trajectories, which means Social ODE can deal better with complicated trajectories.

New Agents and Obstacles Dynamically Appearing. Most previous methods assume that the number of agents/obstacles does not change over time. However, in reality, some agents/obstacles may enter or leave during the course of a trajectory, which has a social influence on other agents. For example, in the autonomous driving scenario, the appearance of a pedestrian(s) close to or in front of the vehicle/agent can happen suddenly and the agent needs to change its trajectory. We conduct a set of experiments to show how the agent's trajectory is modified when a sudden obstacle appears. To model the sudden obstacle, we place a static and a moving obstacle in the predicted agent trajectory from the

Method	length	inD		highD		rounD	
		straight	curve	straight	curve	straight	curve
Social LSTM	2s	0.2474	0.8537	0.2846	0.8347	0.2367	0.8986
Social GAN		0.2537	0.8236	0.2564	0.8977	0.2679	0.8876
DenseTNT		0.2367	**0.8046**	0.2465	0.8546	0.2268	0.8464
AgentFomer		**0.2346**	0.8124	**0.2368**	0.8263	**0.2140**	**0.8259**
Social ODE		0.2408	0.8147	0.2406	**0.8135**	0.2254	0.8357
Social LSTM	4s	0.7973	3.1463	0.9525	3.5364	0.7268	2.6473
Social GAN		0.7861	3.1583	0.8367	3.4637	0.7483	2.6940
DenseTNT		0.7794	3.1578	0.7431	3.1778	0.6543	2.4764
AgentFomer		**0.7604**	3.1483	**0.6814**	3.1527	**0.5924**	2.4748
Social ODE		0.7728	**3.1417**	0.6873	**3.1509**	0.6005	**2.4738**
Social LSTM	8s	2.7536	8.3456	2.4570	9.3365	2.5583	9.1346
Social GAN		2.6573	8.2478	2.3279	9.6437	2.9546	8.9446
DenseTNT		2.6644	8.1475	2.1345	9.3464	2.7854	8.4677
AgentFomer		**2.3474**	8.1457	**2.1167**	9.3258	**2.5337**	8.3464
Social ODE		2.6064	**8.1208**	2.1384	**9.3203**	2.6447	**8.3384**

Table 1. Evaluation on inD, rounD and highD traffic datasets. The bold means best performance.

test dataset. Using our Social ODE approach we observe that the agent modifies the original trajectory to avoid the obstacle collision. In this experiment we test how one agent adapts its trajectory to sudden appearing obstacles, while the trajectories of the other agents are not modified and are kept constant. In our experiments we consider that a collision occurs when the distance between the agent and the obstacle is less than 0.5 meters. Table 2 shows the collision rate of all methods. Social ODE achieves the lowest collision rate while avoiding the static or moving agent. Fig. 4 shows some examples of the agent avoiding an obstacle using our approach. It demonstrates that our model can correctly extract the social relationship among agents and make agents realize that they should avoid other agents or obstacles although there are no similar cases in the training dataset.

Method	Social LSTM	Social GAN	DenseTNT	AgentFormer	Social ODE
Static obstacle	28.6%	29.6%	22.8%	28.4%	**8.8%**
Moving obstacle	32.4%	35.2%	32.6%	33.0%	**12.8%**

Table 2. Collision rate of different methods when introducing a sudden obstacle in the trajectory. Numbers in bold show the best performance.

4.4 Agent Controlling with Social ODE

In section 3.6 we showed how our proposed Social ODE can control the agent's trajectory through eq.(17). We conduct experiments to show how the use of

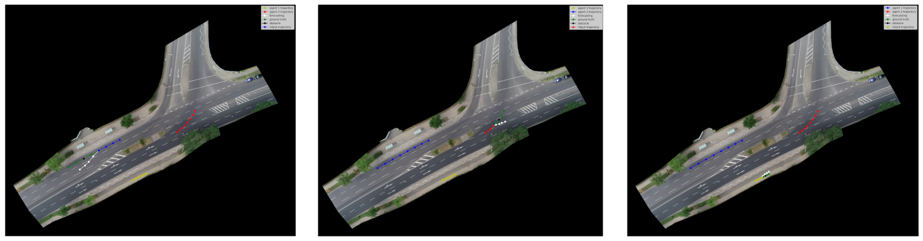


Fig. 4. Sudden obstacle visualization. In each image, the green trajectory is the ground truth and the white one is the predicted result. The black point is the sudden obstacle. The obstacle is placed in the ground truth trajectory. The white trajectory demonstrates that the agent successfully avoids the obstacle.

the attractor (target) and the repeller (obstacles) affect the agent. All the experiments in this section are conducted during testing, without the need for retraining.

Target. In the test dataset, one point near the last trajectory point is set as the target. We represent the target in latent space using the encoder and use eq.(17) to model the agent. λ is set from 0 to 10 and the reaching rate is computed within 8 seconds from the beginning of the trajectory. When $\lambda = 0$ then there is no target modeling within our Social ODE model. In this case reaching the target is defined when the distance between the agent and the target is less than 0.5 meters. All other methods except denseTNT cannot use the target to control the agent. DenseTNT directly plans the path between the start point and the target and therefore can't be used for dynamic target or obstacle introduction like our approach. We therefore present results of target reaching using our method in Table 3, when λ changes. The results show that the larger the value of λ results in stronger attraction by the target.

λ	0	2	4	6	8	10
Reaching rate	4.8%	8.6%	25.4%	43.6%	71.4%	**87.8%**

Table 3. Reaching rate with different values of λ, if we dynamically set the target to be the last trajectory point, during agent movement. Bold numbers mean the best performance.

Obstacle Avoidance and Return to the Agent Trajectory. In 4.3, showed how in our approach an agent can avoid an obstacle. However, after the target avoids a sudden obstacle, it should come back to the original trajectory assuming the target is not changed. We do some experiments to verify whether eq.(17) can control an agent to avoid a sudden obstacle and return to the original trajectory. Similar to 4.4, we place a static obstacle in the predicted trajectory. Fig. 5 shows that the agent can bypass the obstacle, similarly to how drivers react when encountering while driving a sudden obstacle. This is done without retraining and there is no similar case in the training dataset.

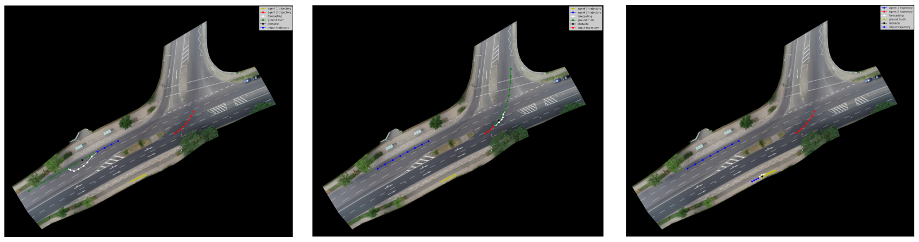
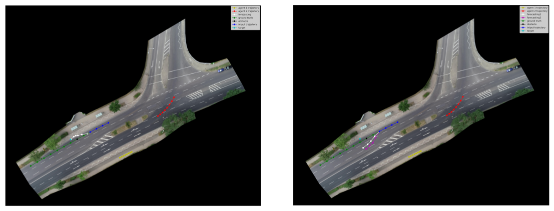


Fig. 5. Avoid the obstacle and return to the agent Trajectory. In each image, the green trajectory is the ground truth and the white is the predicted result. The black point is the sudden obstacle. The agent avoids the obstacle and returns to the original trajectory.

Obstacles and Targets. We also conducted experiments where we introduced obstacles and targets dynamically during an agent's trajectory and we showed modifications to the agent trajectory and the successful reaching of the target. Fig.6 demonstrates that the agent can avoid the obstacle and reach the target.

Adjusting the Relational Dimension. In eq.(15), the β_1 and β_2 are parameters that can modify the effect of the relational dimension. Larger β_1 means the agent's trajectory tends to be affected more by other agents and larger β_2 means the agent's trajectory tends to keep its previous moving pattern. The results in Fig. 6b show the effect of those parameters on agent trajectories.



(a) Avoid obstacle and reach target (b) Adjust the relational dimension

Fig. 6. (a) Avoid the obstacle and reach the target. The white trajectory is the output result of social ODE. The green one is the ground truth. The cyan point is the target and the black point is the obstacle. (b) Adjusting the relational dimension. White trajectory: $\beta_1 = 1$. Pink trajectory: $\beta_1 = 2$.

4.5 Ablation Study

In this section, we verify the design of the proposed Social ODE model. We do this by replacing components with similar components as follows.

Latent ODE Encoder + Our Decoder. We replace the decoder of the Latent ODE with our decoder, which enables the Latent ODE to model the relational dimension.

Social LSTM + Our Decoder. We use social LSTM to encode the relational dimension and temporal dimension for each agent to the latent vector. Then our decoder recovers the latent trajectory back to a realistic trajectory.
Our Encoder + Neural ODE. Our Spatio-temporal Transformer is used as the encoder, which generates the latent vector. The neural ODE decodes the latent vector to position and velocity.
Our Encoder + Social Pooling ODE. Our Spatio-temporal Transformer is used as the encoder to generate a latent vector. Instead of modeling the relational dimension by distance, interaction dynamics and aggressiveness, we use the social pooling from the Social LSTM model for latent vector in each time step.

The results are shown in Table 4. From the table, we can see that all the changes in our components result in a performance decrease, which means the design of our Social ODE is effective.

Method	inD		highD		rounD	
	straight	curve	straight	curve	straight	curve
Latent ODE + our decoder	0.7732	3.2147	0.7886	3.3394	0.6373	2.5367
Social LSTM + our decoder	0.7864	3.1584	0.7630	3.3256	0.6357	2.5774
Our encoder + Neural ODE	0.7925	3.1647	0.7974	3.2754	0.6438	2.6227
Our encoder + Social pooling ODE	0.7857	3.1594	0.7533	3.2740	0.6363	2.5830
Social ODE	**0.7728**	**3.1417**	**0.6873**	**3.1509**	**0.6005**	**2.4738**

Table 4. Ablation Study: Evaluation of changing some components on inD, rounD and highD traffic datasets. The forecasting length is 4 seconds. Bold depicts the best performance.

5 Conclusion

In this paper, we present a Social ODE, which models and learns agent interaction and underlying temporal dynamics explicitly. To model the agent interaction, our Social ODE decouples it into three components: distance, interaction intensity and aggressiveness, all of which are multiplied to estimate the relational dimension. Meanwhile, the underlying temporal dynamics are learned by a Neural ODE in latent space, which includes agent interaction and the current state. We have validated the performance of Social ODE through extensive experiments using traffic datasets. Compared with previous schemes, our Social ODE is shown to achieve favorable performance in terms of forecasting accuracy. Social ODE can also allow the dynamic insertion of obstacles, targets and agents during the course of an agent's trajectory without *retraining*. As a result, our model achieves a lower collision rate when sudden obstacles occur in the trajectory and can control the agent motion by dynamically inserting attractors or repellers.
Acknowledgements: Research partially funded by research grants to Metaxas from NSF: 1951890, 2003874, 1703883, 1763523 and ARO MURI SCAN.

References

1. Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., Savarese, S.: Social lstm: Human trajectory prediction in crowded spaces. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 961–971 (2016)
2. Bock, J., Krajewski, R., Moers, T., Runde, S., Vater, L., Eckstein, L.: The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. In: 2020 IEEE Intelligent Vehicles Symposium (IV). pp. 1929–1934 (2020). https://doi.org/10.1109/IV47402.2020.9304839
3. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in neural information processing systems **31** (2018)
4. De Brouwer, E., Simm, J., Arany, A., Moreau, Y.: Gru-ode-bayes: Continuous modeling of sporadically-observed time series. Advances in neural information processing systems **32** (2019)
5. Dupont, E., Doucet, A., Teh, Y.W.: Augmented neural odes. Advances in Neural Information Processing Systems **32** (2019)
6. Durkan, C., Bekasov, A., Murray, I., Papamakarios, G.: Neural spline flows. Advances in neural information processing systems **32** (2019)
7. Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., Schmid, C.: Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11525–11533 (2020)
8. Graber, C., Schwing, A.: Dynamic neural relational inference for forecasting trajectories. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. pp. 1018–1019 (2020)
9. Gruenbacher, S., Lechner, M., Hasani, R., Rus, D., Henzinger, T.A., Smolka, S., Grosu, R.: Gotube: Scalable stochastic verification of continuous-depth models. arXiv preprint arXiv:2107.08467 (2021)
10. Grunbacher, S., Hasani, R., Lechner, M., Cyranka, J., Smolka, S.A., Grosu, R.: On the verification of neural odes with stochastic guarantees. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 11525–11535 (2021)
11. Gu, J., Sun, C., Zhao, H.: Densetnt: End-to-end trajectory prediction from dense goal sets. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 15303–15312 (2021)
12. Gupta, A., Johnson, J., Fei-Fei, L., Savarese, S., Alahi, A.: Social gan: Socially acceptable trajectories with generative adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2255–2264 (2018)
13. Hasani, R., Lechner, M., Amini, A., Rus, D., Grosu, R.: Liquid time-constant networks. arXiv preprint arXiv:2006.04439 (2020)
14. Jia, J., Benson, A.R.: Neural jump stochastic differential equations. Advances in Neural Information Processing Systems **32** (2019)
15. Kipf, T., Fetaya, E., Wang, K.C., Welling, M., Zemel, R.: Neural relational inference for interacting systems. In: International Conference on Machine Learning. pp. 2688–2697. PMLR (2018)
16. Krajewski, R., Bock, J., Kloeker, L., Eckstein, L.: The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). pp. 2118–2125 (2018). https://doi.org/10.1109/ITSC.2018.8569552

17. Krajewski, R., Moers, T., Bock, J., Vater, L., Eckstein, L.: The round dataset: A drone dataset of road user trajectories at roundabouts in germany. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). pp. 1–6 (2020). https://doi.org/10.1109/ITSC45102.2020.9294728
18. Li, J., Yang, F., Tomizuka, M., Choi, C.: Evolvegraph: Multi-agent trajectory prediction with dynamic relational reasoning. Advances in neural information processing systems **33**, 19783–19794 (2020)
19. Liang, Y., Ouyang, K., Yan, H., Wang, Y., Tong, Z., Zimmermann, R.: Modeling trajectories with neural ordinary differential equations. In: IJCAI. pp. 1498–1504 (2021)
20. Liebenwein, L., Hasani, R., Amini, A., Rus, D.: Sparse flows: Pruning continuous-depth models. Advances in Neural Information Processing Systems **34** (2021)
21. Park, S., Kim, K., Lee, J., Choo, J., Lee, J., Kim, S., Choi, E.: Vid-ode: Continuous-time video generation with neural ordinary differential equation. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 2412–2422 (2021)
22. Quaglino, A., Gallieri, M., Masci, J., Koutník, J.: Snode: Spectral discretization of neural odes for system identification. arXiv preprint arXiv:1906.07038 (2019)
23. Rubanova, Y., Chen, R.T., Duvenaud, D.K.: Latent ordinary differential equations for irregularly-sampled time series. Advances in neural information processing systems **32** (2019)
24. Salzmann, T., Ivanovic, B., Chakravarty, P., Pavone, M.: Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In: European Conference on Computer Vision. pp. 683–700. Springer (2020)
25. Shi, R., Morris, Q.: Segmenting hybrid trajectories using latent odes. In: International Conference on Machine Learning. pp. 9569–9579. PMLR (2021)
26. Sun, C., Karlsson, P., Wu, J., Tenenbaum, J.B., Murphy, K.: Stochastic prediction of multi-agent interactions from partial observations. arXiv preprint arXiv:1902.09641 (2019)
27. Vorbach, C., Hasani, R., Amini, A., Lechner, M., Rus, D.: Causal navigation by continuous-time neural networks. Advances in Neural Information Processing Systems **34** (2021)
28. Yan, H., Du, J., Tan, V.Y., Feng, J.: On robustness of neural ordinary differential equations. arXiv preprint arXiv:1910.05513 (2019)
29. Yildiz, C., Heinonen, M., Lahdesmaki, H.: Ode2vae: Deep generative second order odes with bayesian neural networks. Advances in Neural Information Processing Systems **32** (2019)
30. Yuan, Y., Weng, X., Ou, Y., Kitani, K.M.: Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9813–9823 (2021)