Trajectory Prediction

Yueyuan Li

2022/7/20

Outline

- Introduction:
  - What & Why
  - Literature review
- Representative methods
  - Social-LSTM
  - VectorNet

01 Introduction

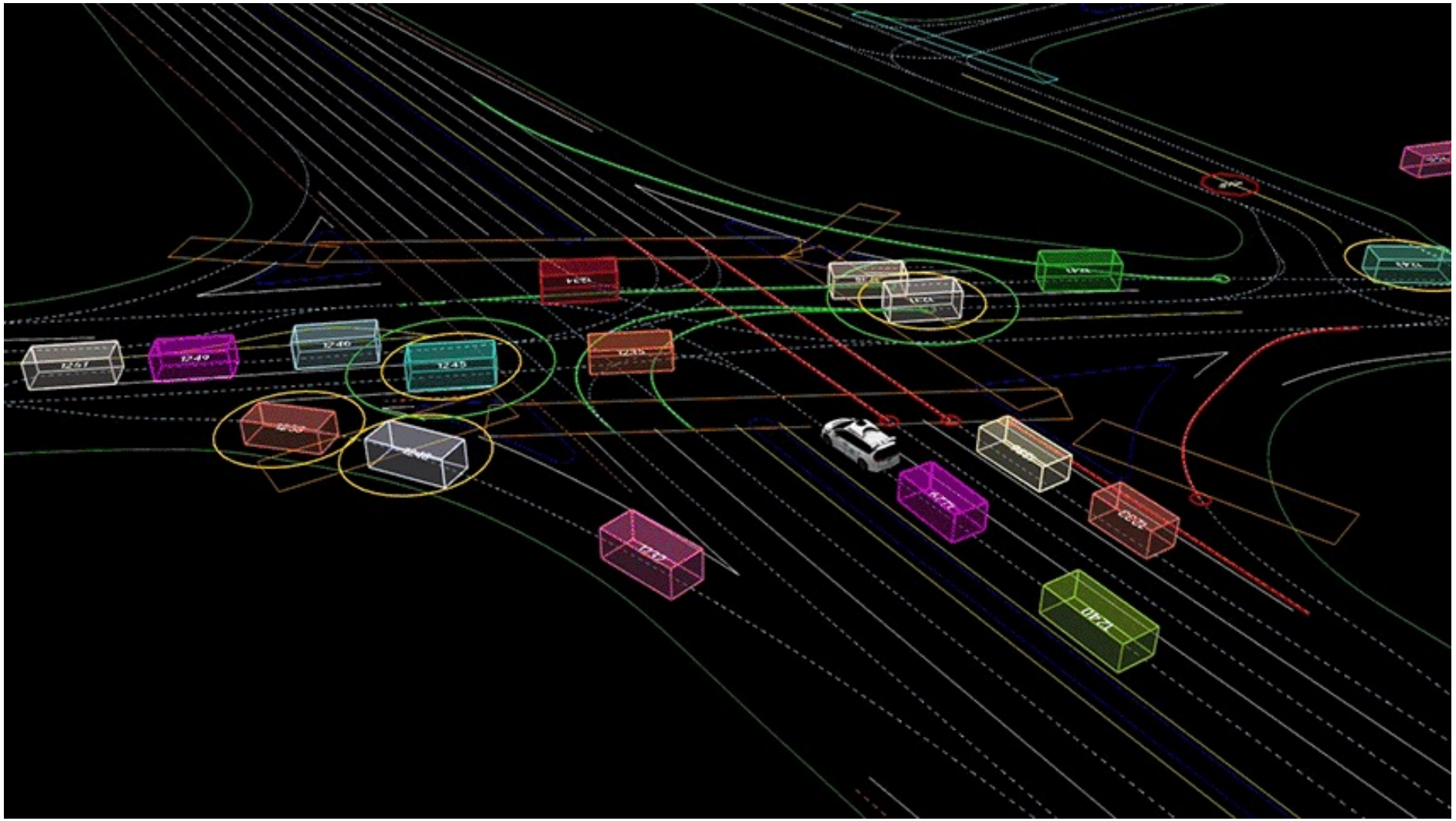
What is trajectory prediction?

Task: Predicting the future spatial coordinates of agents.

- Duration: short-term (< 3s) or long-term (3~5s)
- Agents: such as cars, buses, pedestrians, etc.

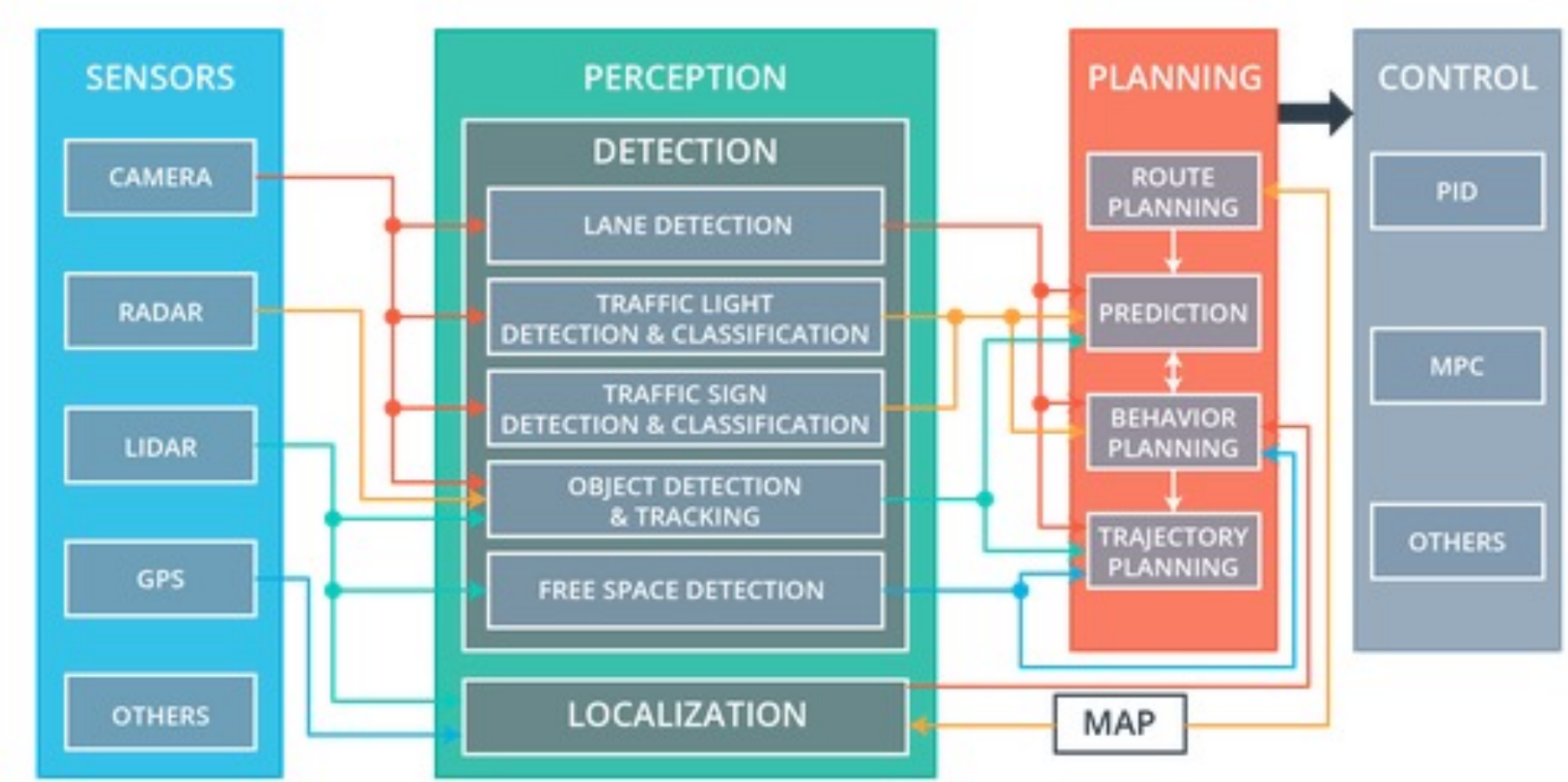
What is trajectory prediction?

What is trajectory prediction?



Why trajectory prediction?

Application: Mainly for self-driving vehicles



Main Challenges

- State representation: Extract effective features from input;
- Interaction modeling: The agents are interacting with each other;
- Multi-modal output: The correct output is not unique;
- Tradeoff between safety and efficiency;
- Corner cases

Previous works

Traditional approaches for motion prediction are based on hand-crafted rules and energy potentials

- Rule-based:
  - Social force (1995): attractive and repulsive force
  - Clustering technique (2004): based on distance measurement
- Data-driven:
  - VectorNet (2020): improve state representation
  - LaneGCN (2020): model the interaction patterns
  - GMM/MDNs: implicitly modeling the multi-modes as latent variables

02

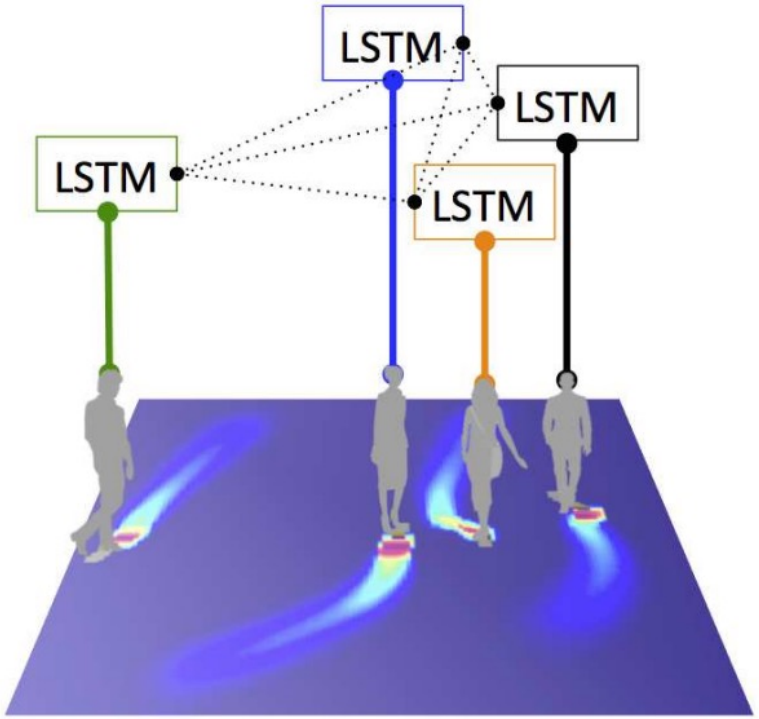
Social LSTM

Human Trajectory Prediction in Crowded Spaces

Social LSTM

Social LSTM: Human Trajectory Prediction in Crowded Spaces

Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, Silvio Savarese; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 961-971



Abstract

Humans navigate complex crowded environments based on social conventions: they respect personal space, yielding right-of-way and avoid collisions. In our work, we propose a data-driven approach to learn these human-human interactions for predicting their future trajectories. This is in contrast to traditional approaches which use hand-crafted functions such as Social forces. We present a new Long Short-Term Memory (LSTM) model which jointly reasons across multiple individuals in a scene. Different from the conventional LSTM, we share the information between multiple LSTMs through a new pooling layer. This layer pools the hidden representation from LSTMs corresponding to neighboring trajectories to capture interactions within this neighborhood. We demonstrate the performance of our method on several public datasets. Our model outperforms previous forecasting methods by more than 42% . We also analyze the trajectories predicted by our model to demonstrate social behaviours such as collision avoidance and group movement, learned by our model.

LSTM

- Long Short-Term Memory network (LSTM)

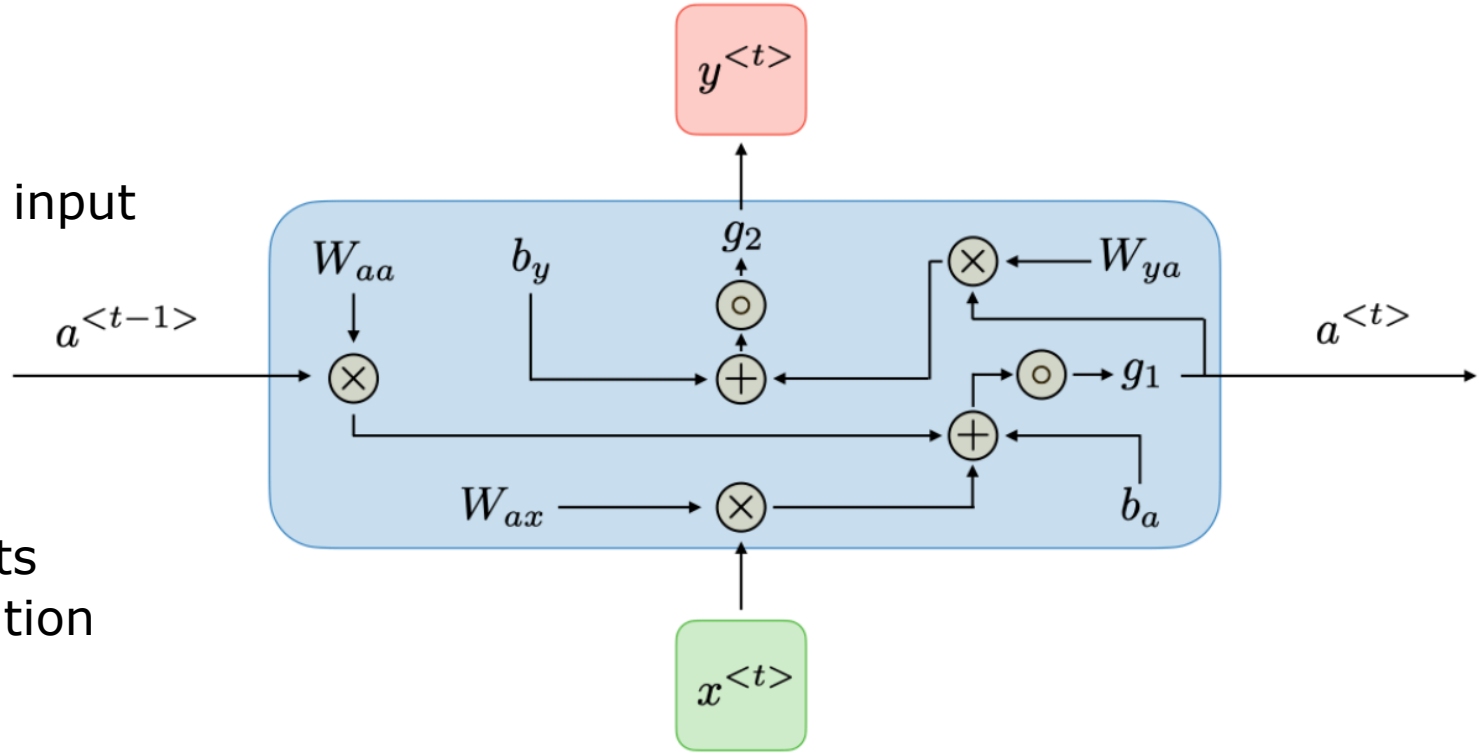
- RNN: a class of neural networks that allow previous outputs to be used as inputs while having hidden states.
- LSTM: a kind of Recurrent Neural Network (RNN)

Naïve RNN

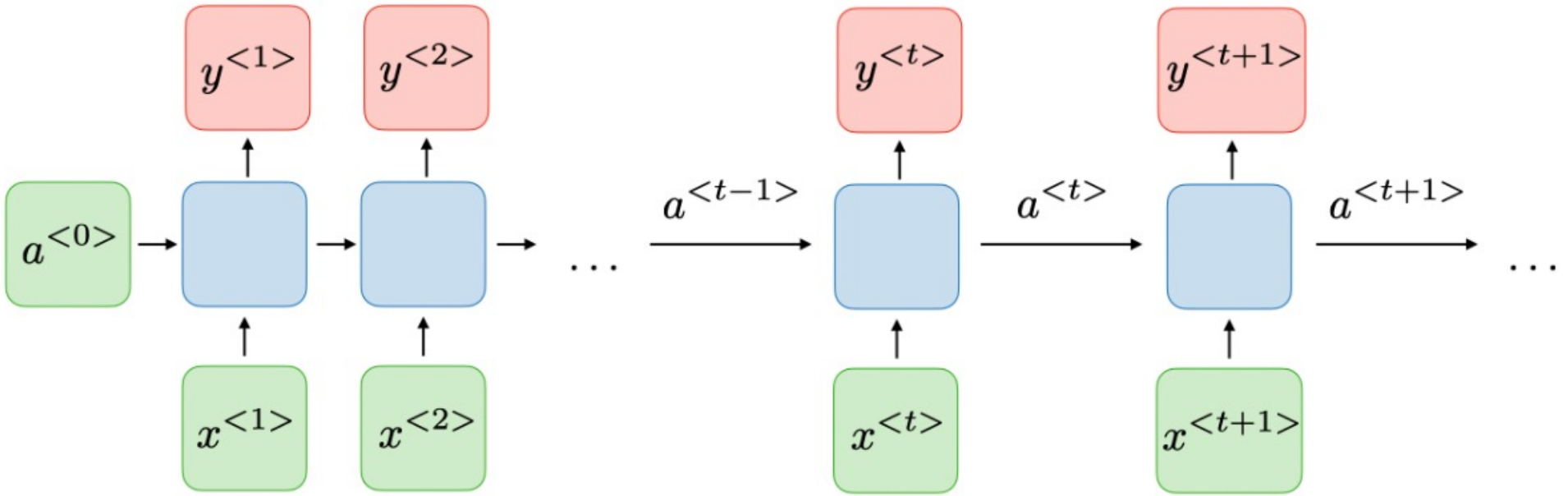
$$a^{<t>}, y^{<t>} = f(x^{<t>}, a^{<t-1>})$$

- $a^{<t>}$: hidden state used as the next input
  $a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$
- $y^{<t>}$: output
  $y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients shared temporally and g_1, g_2 are activation functions.



Naïve RNN

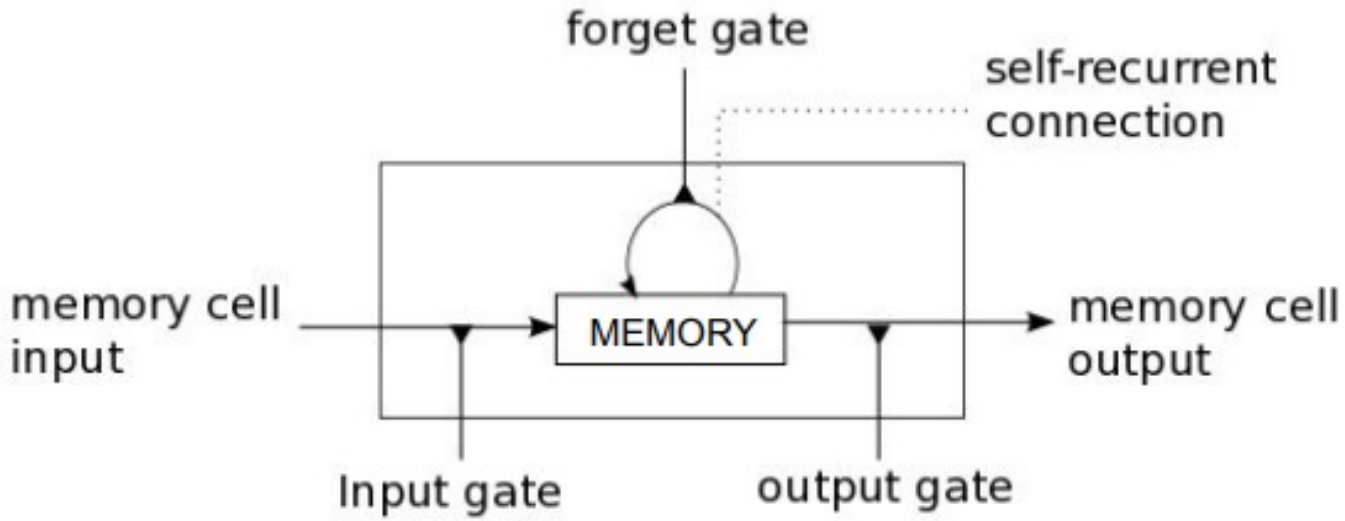


$$a^{<t>}, y^{<t>} = f(x^{<t>}, a^{<t-1>})$$

- $a^{<t>}$: hidden state used as the next input
  $a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$
- $y^{<t>}$: output
  $y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$

LSTM

- Central Idea: A memory cell (interchangeably block) which can maintain its state over time, consisting of an explicit memory (aka the cell state vector) and gating units which regulate the information flow into and out of the memory.

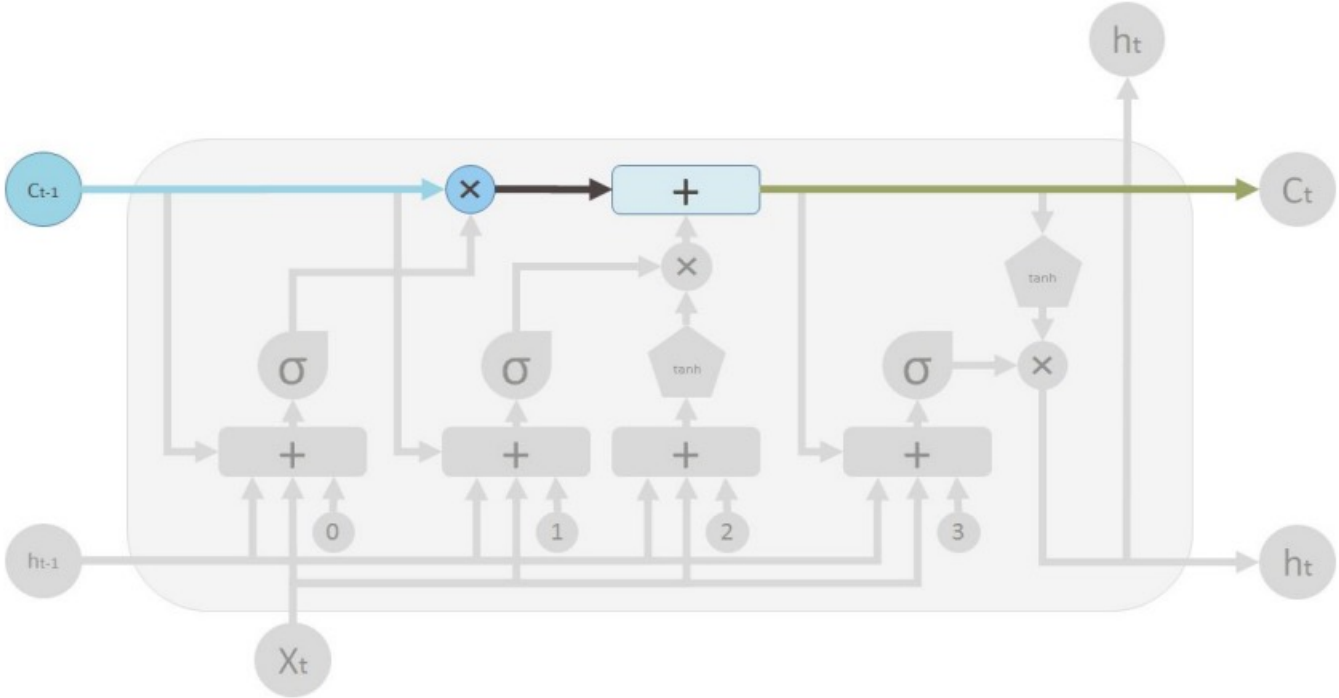


LSTM Memory Cell

LSTM

Cell state

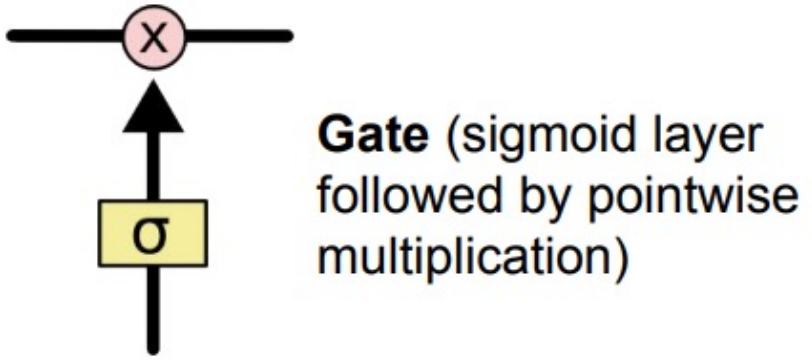
- Represents the memory of the LSTM
- Undergoes changes via forgetting of old memory (forget gate) and addition of new memory (input gate)



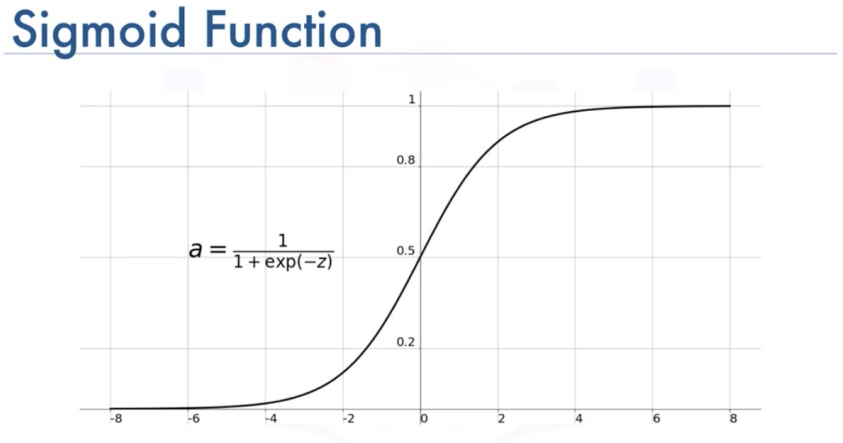
LSTM

Gate

- Gate: sigmoid neural net layer followed by pointwise multiplication operator
- Gates control the flow of information to/from the memory
- Gates are controlled by a concatenation of the output from the previous time step and the current input and optionally the cell state vector.

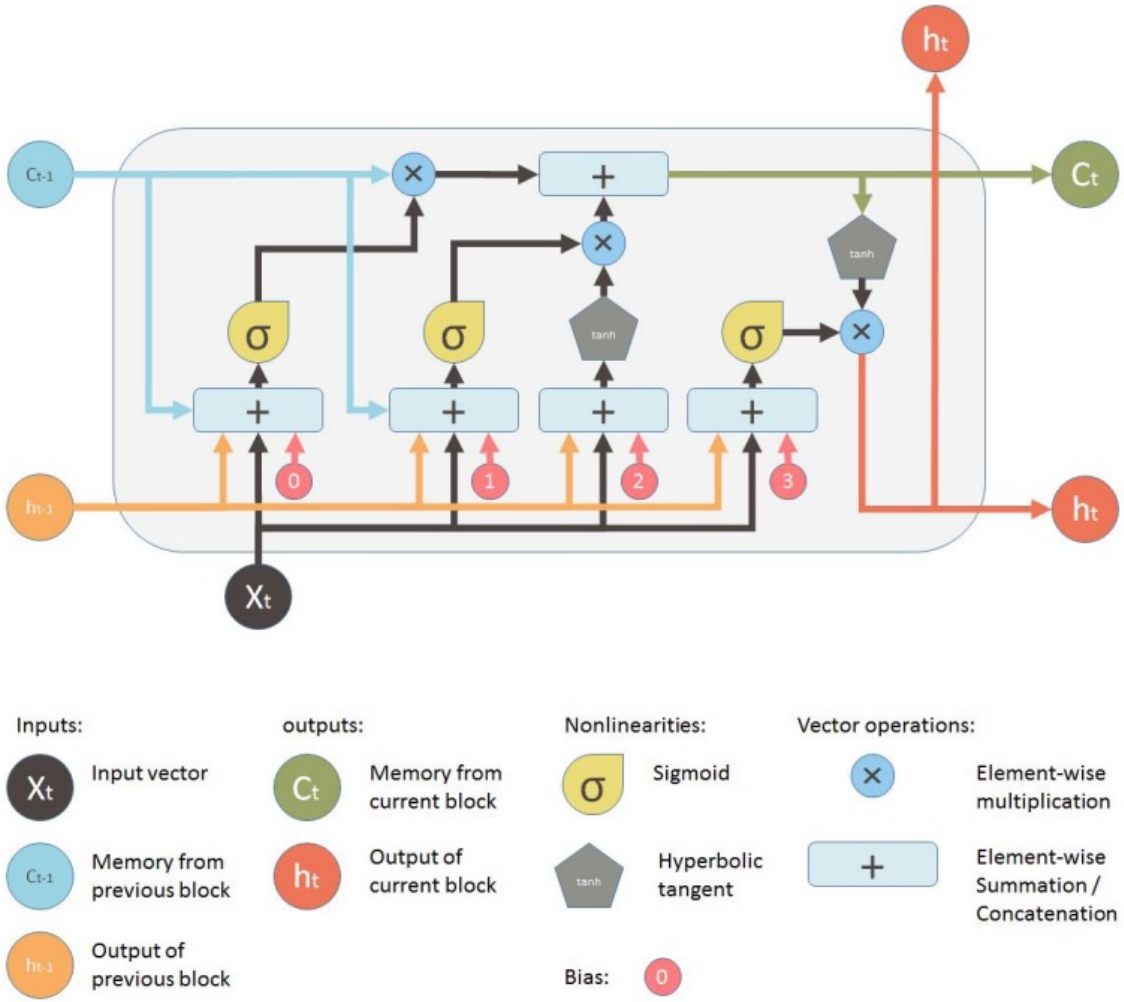


$$\Gamma_* = \sigma(W[x^{<t>}, h^{<t-1>}] + b)$$



LSTM

Cell overview



Passing two states:
the hidden state and the cell state
($h^{<t>}$ is the $a^{<t>}$ in the Naïve RNN)

- $\tilde{c}^{<t>} = \tanh(W_c[x^{<t>}, h^{<t-1>}] + b_c)$
- $c^{<t>} = \Gamma_i \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$
- $h^{<t>} = \Gamma_o \odot c^{<t>}$

where $\odot$ is the Hadamard product,
Γ_* are different gates defined by

$$\Gamma_* = \sigma(Wx^{<t>} + Ua^{<t>} + b)$$

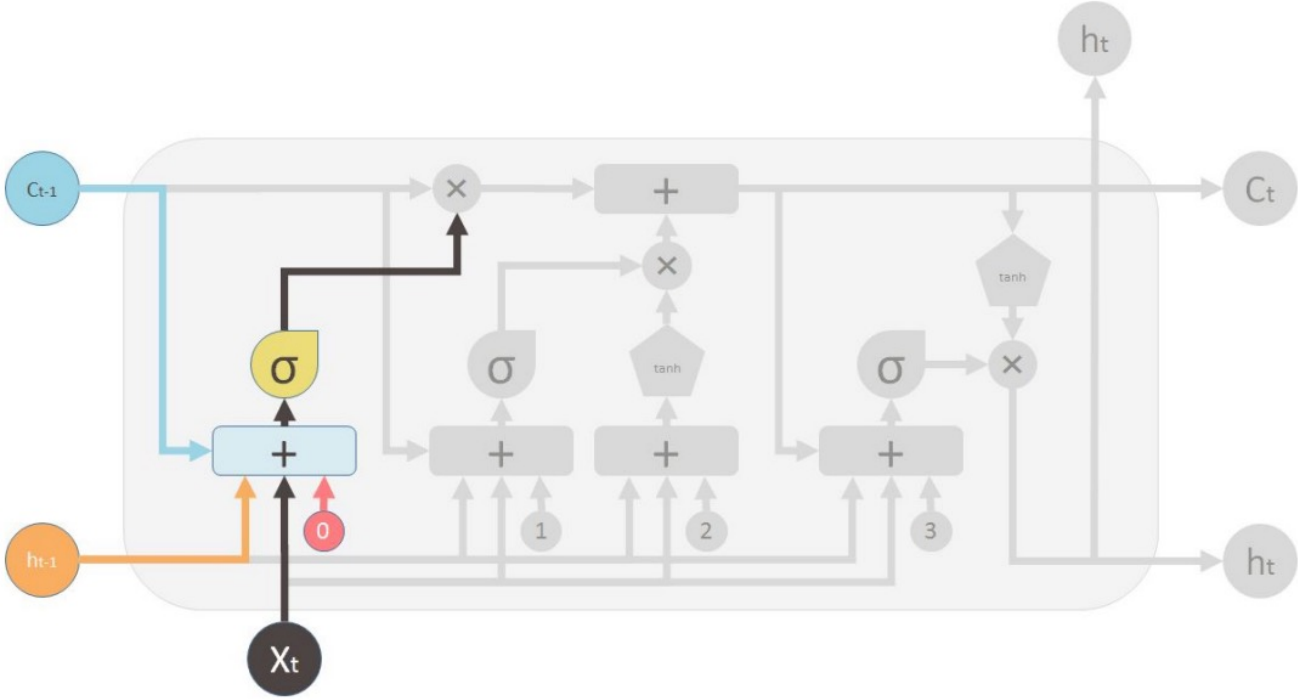
Specifically,

- Γ_f: forget gate;
- Γ_i: input gate;
- Γ_o: output gate.

LSTM

Forget Gate

- Decides what information to throw away from memory

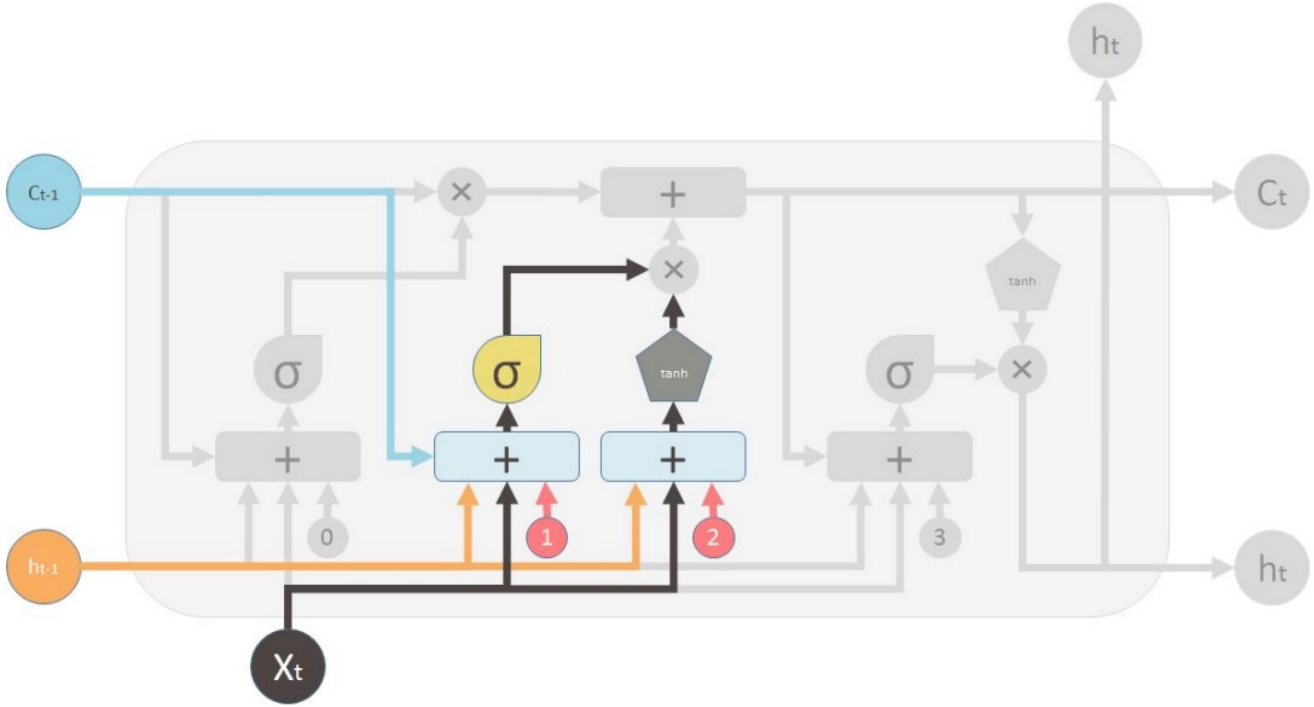


$$\Gamma_f = \sigma\left(W_f[x^{<t>}, h^{<t-1>}] + b_f\right)$$

LSTM

Input Gate

- Controls what new information is added to cell state from current input

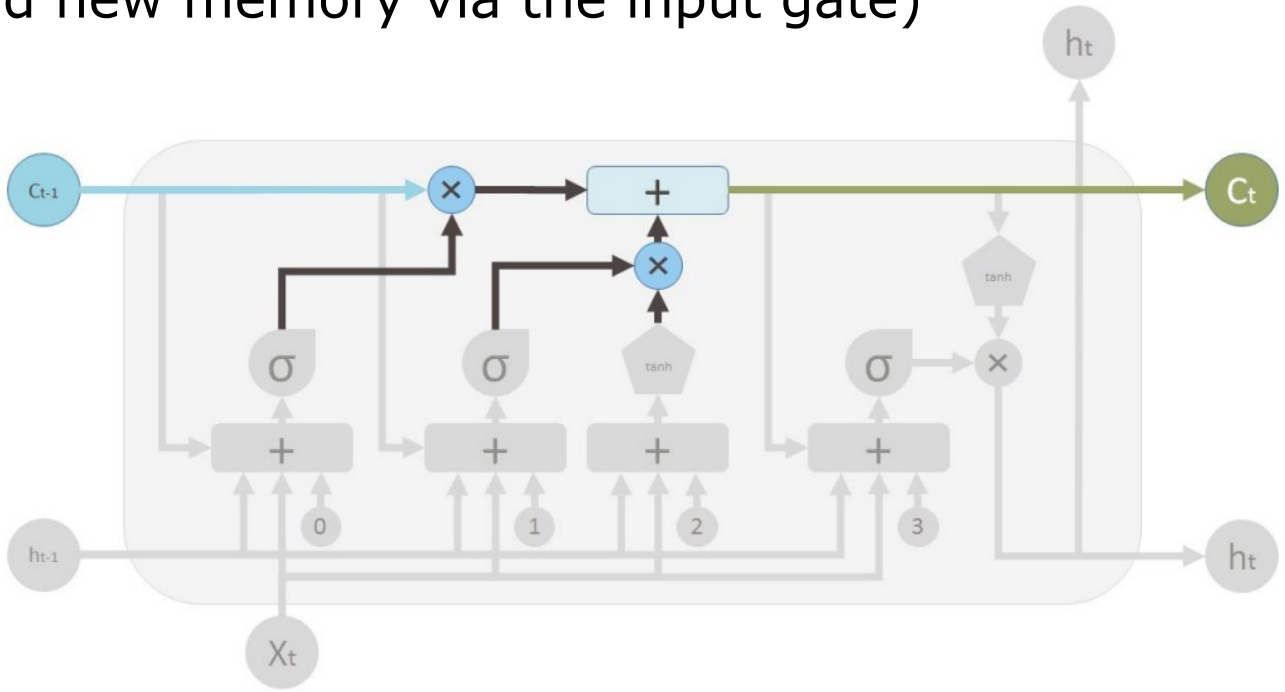


$$\Gamma_i = \sigma(W_i[x^{<t>}, h^{<t-1>}] + b_i)$$
$$\tilde{c}^{<t>} = \tanh(W_c[x^{<t>}, h^{<t-1>}] + b_c)$$

LSTM

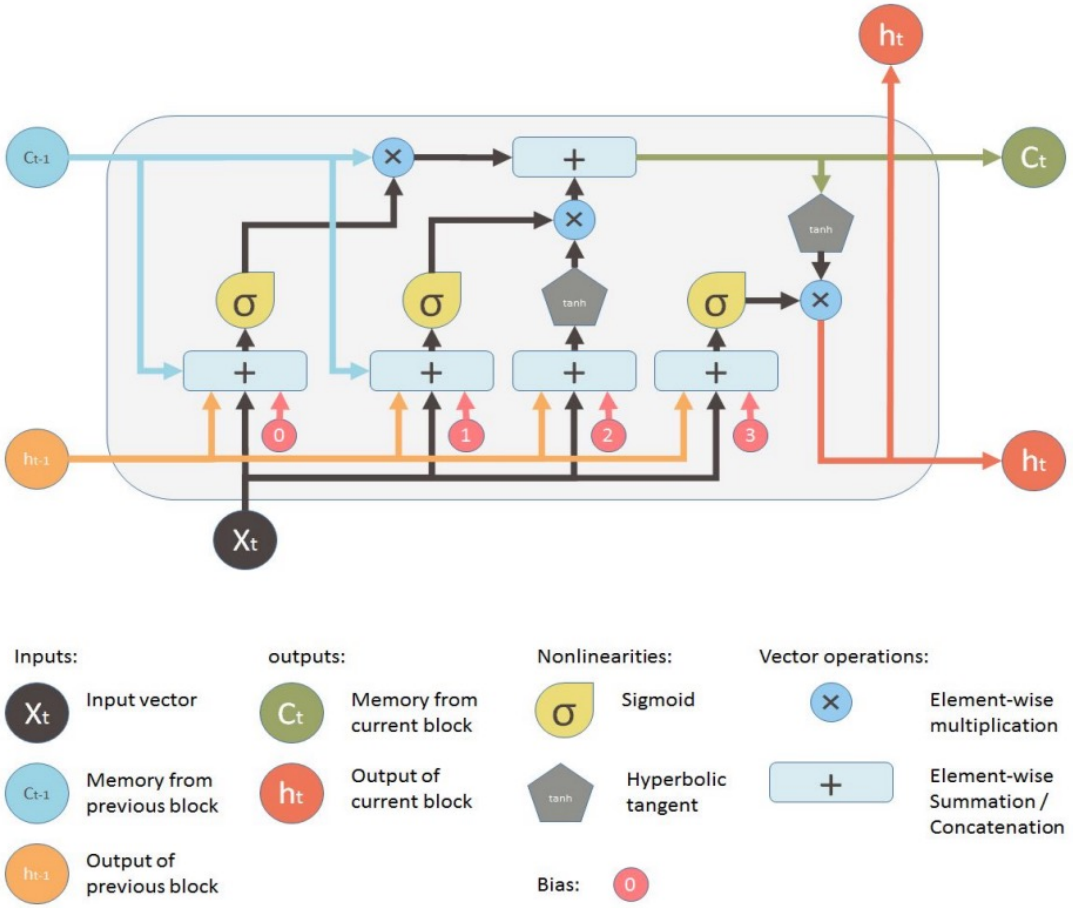
Memory update

- The cell state vector aggregates the two components (old memory via the forget gate and new memory via the input gate)



$$c^{<t>} = \Gamma_i \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

LSTM



- $\Gamma_f = \sigma\left(W_f[x^{<t>}, h^{<t-1>}] + b_f\right)$
- $\Gamma_i = \sigma(W_i[x^{<t>}, h^{<t-1>}] + b_i)$
- $\tilde{c}^{<t>} = \tanh(W_c[x^{<t>}, h^{<t-1>}] + b_c)$
- $c^{<t>} = \Gamma_i \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$
- $\Gamma_o = \sigma(W_o[x^{<t>}, h^{<t-1>}] + b_o)$
- $h^{<t>} = \Gamma_o \odot \tanh c^{<t>}$

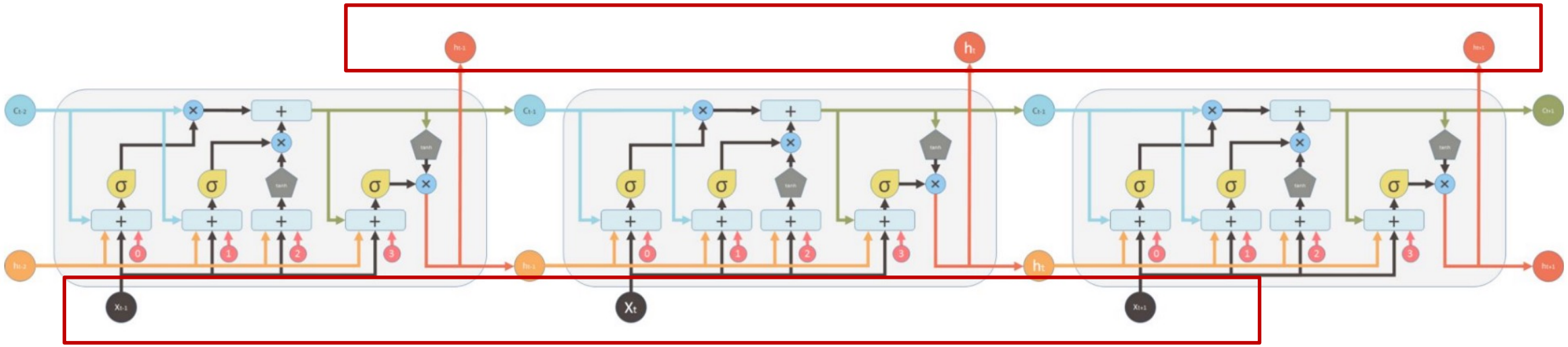
LSTM

LSTM-based trajectory prediction
(A many-to-one way of prediction)

Output trajectory sequence

FC layer

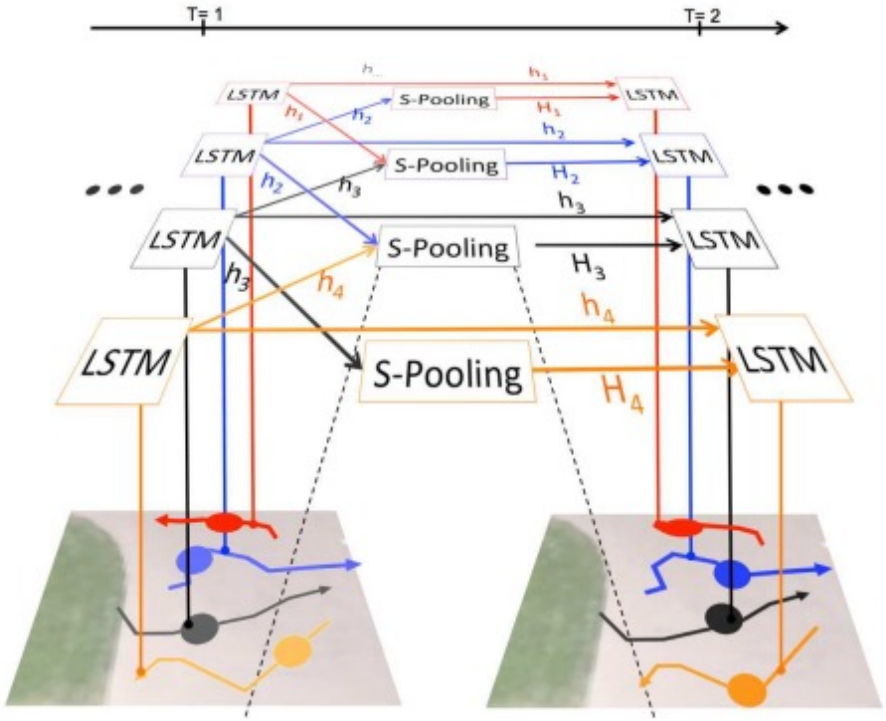
Output sequence feature



Input trajectory sequence

Social LSTM

- LSTM’s limitation: consider the behavior of each individual actor and ignore the rich interactions among multiple actors in a scene.
- Solution: introduce a ”Social” pooling layer which allows the LSTMs of spatially proximal sequences to share their hidden-states with each other.



Social LSTM

Denote the hidden state of the i^{th} person at time t as h_i^t.

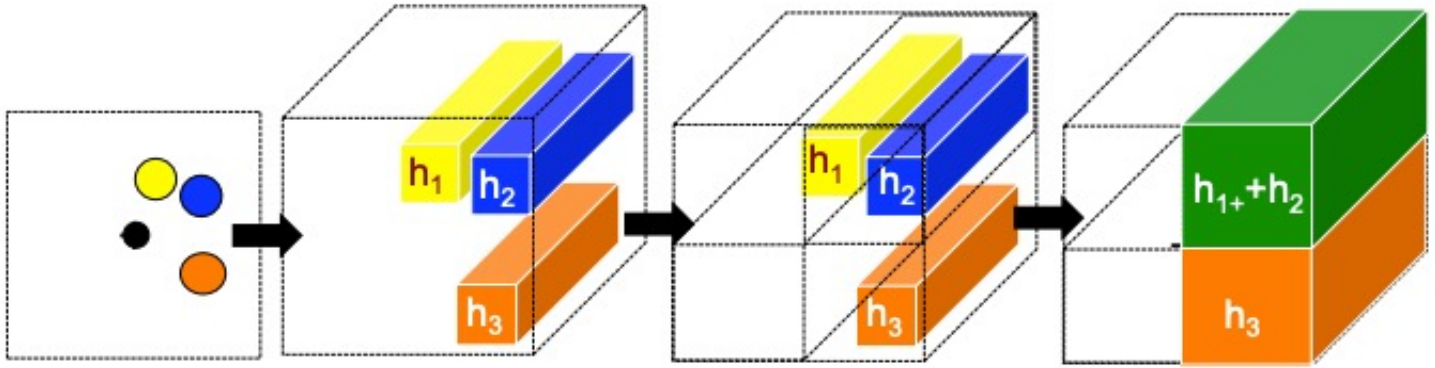
Given a hidden-state dimension D, and neighborhood size N_o, we construct a $N_o \times N_o \times D$ tensor for the i^{th} trajectory:

$$H_i^t(m,n,:) = \prod_{j \in \mathcal{N}_i} \mathbf{1}_{mn}\left[x_j^t - x_i^t, y_j^t - y_i^t\right] h_j^{t-1}$$

indicator

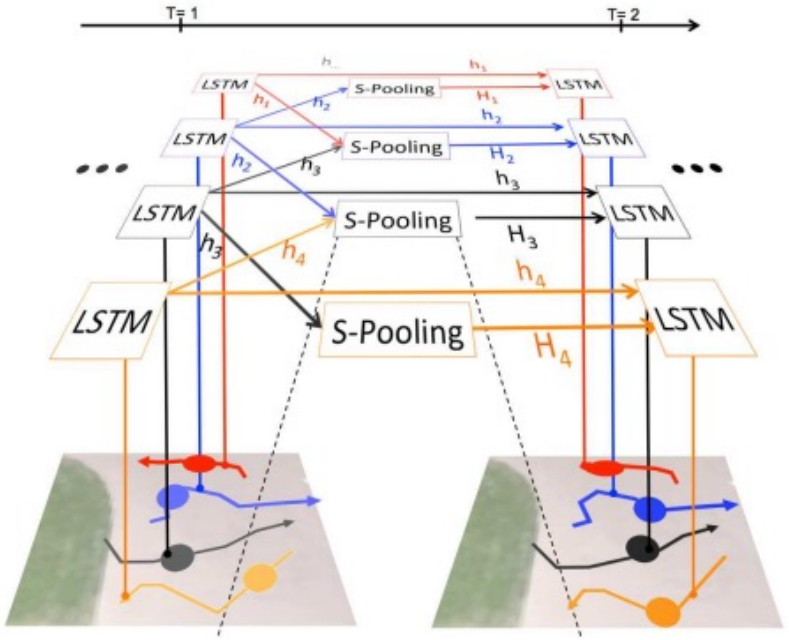
hidden state of the LSTM

Consider neighbors of different grid (m,n)



Social LSTM

Hidden state update:



Coordinate embedding $e_i^t = \phi(x_i^t, y_i^t; W_e)$

Social embedding $a_i^t = \phi(H_i^t; W_a)$

Hidden state update $h_i^t = LSTM(h_i^{t-1}, e_i^t, a_i^t; W_l)$

Social LSTM

Position estimation: use hidden state h_i^t to predict the distribution of the trajectory position $(\hat{x}, \hat{y})_i^{t+1}$ at time step $t+1$.

Assumption: follows a bivariate Gaussian distribution, i.e.

$$(\hat{x}, \hat{y})_i^t \sim \mathcal{N}\left(\mu_i^t, \sigma_i^t, \rho_i^t\right)$$

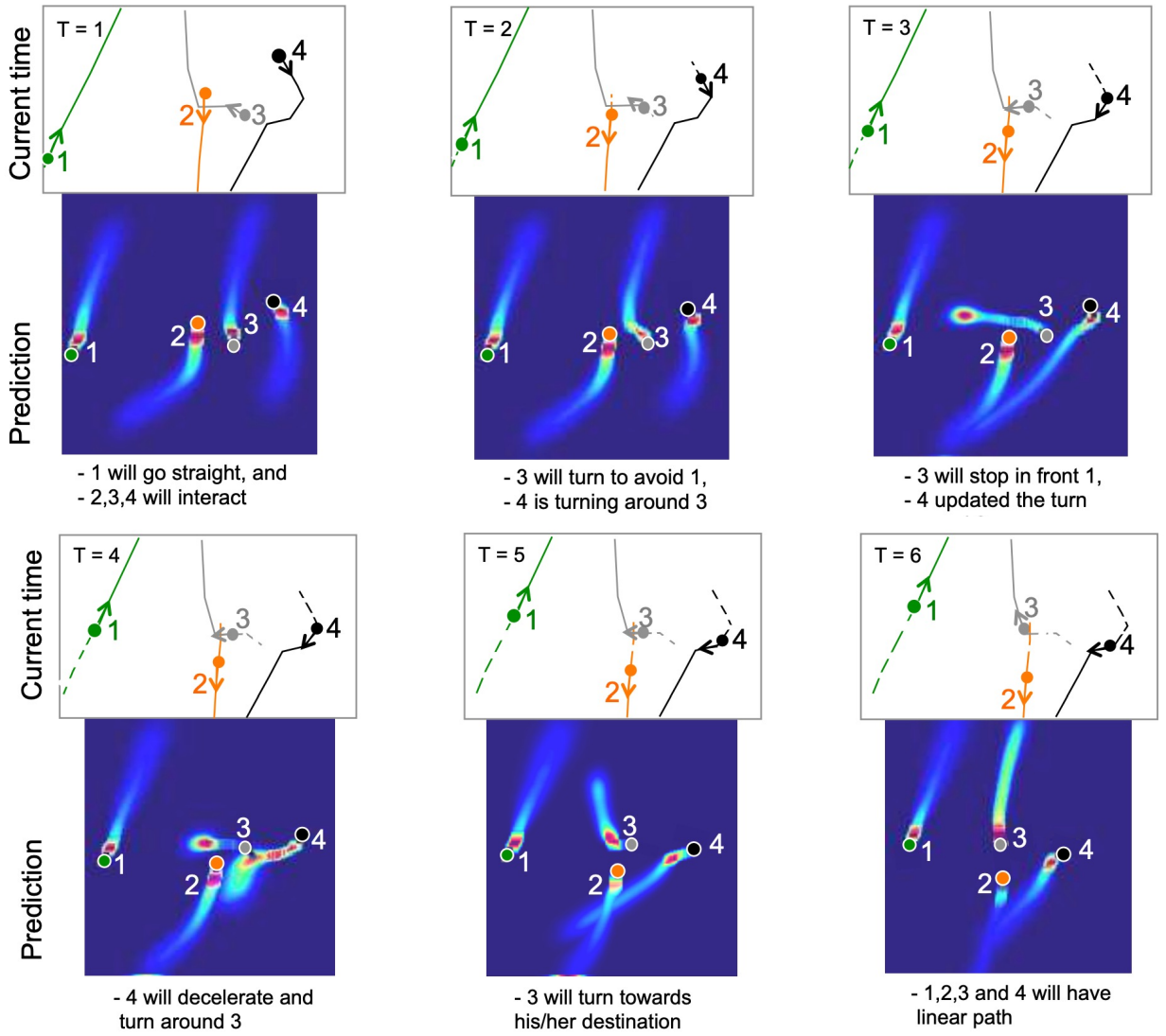
mean std correlation coefficient

The parameters of the LSTM model are learned by minimizing the negative log-likelihood loss

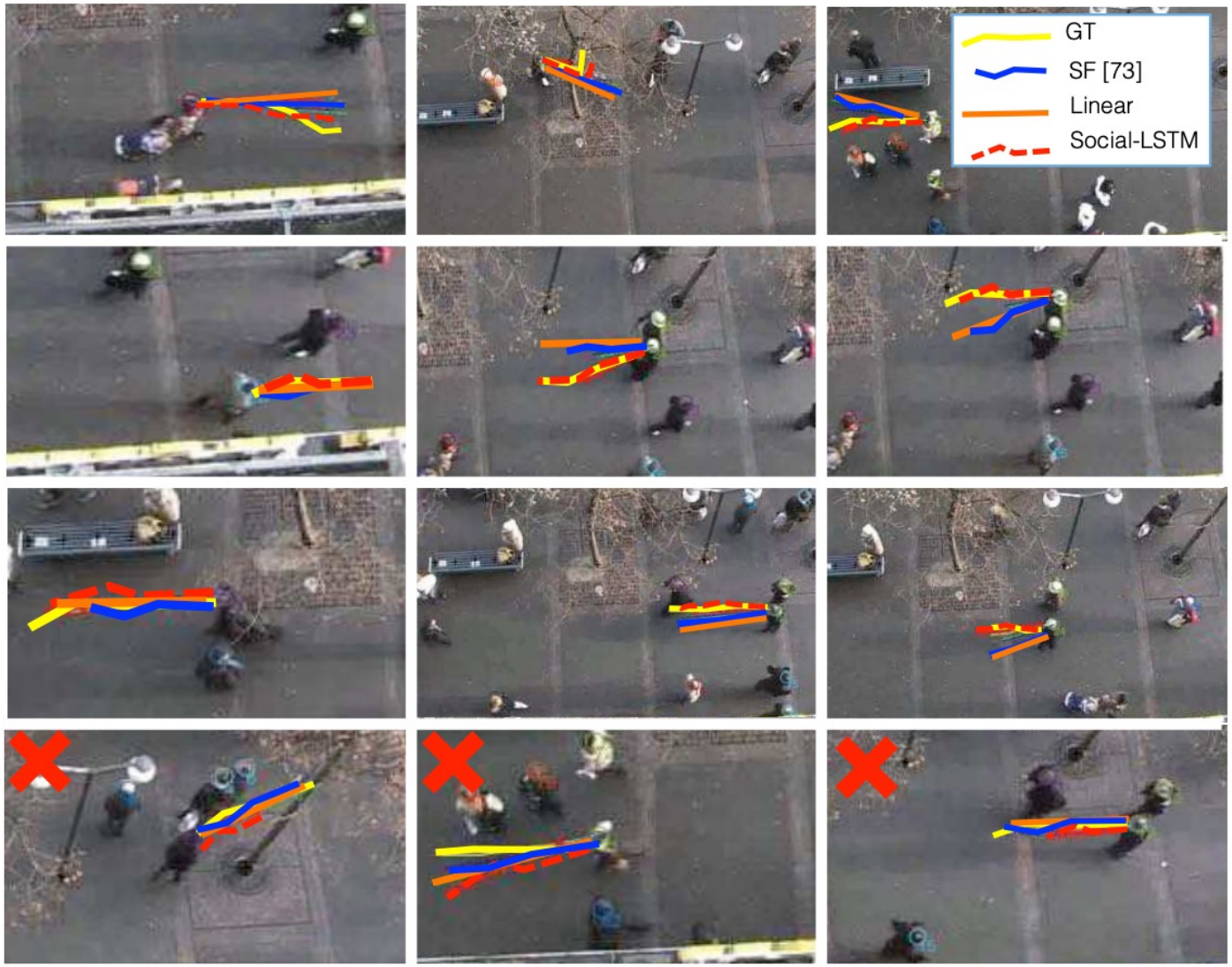
$$\left[\mu_i^t, \sigma_i^t, \rho_i^t\right] = W_p h_i^{t-1}$$

$$L^i\left(W_e, W_l, W_p\right) = -\sum_{t=T_{obs}+1}^{T_{pred}} \log\left(\mathbb{P}\left(x_i^t, y_i^t | \mu_i^t, \sigma_i^t, \rho_i^t\right)\right)$$

Performance of Social LSTM



Performance of Social LSTM



03

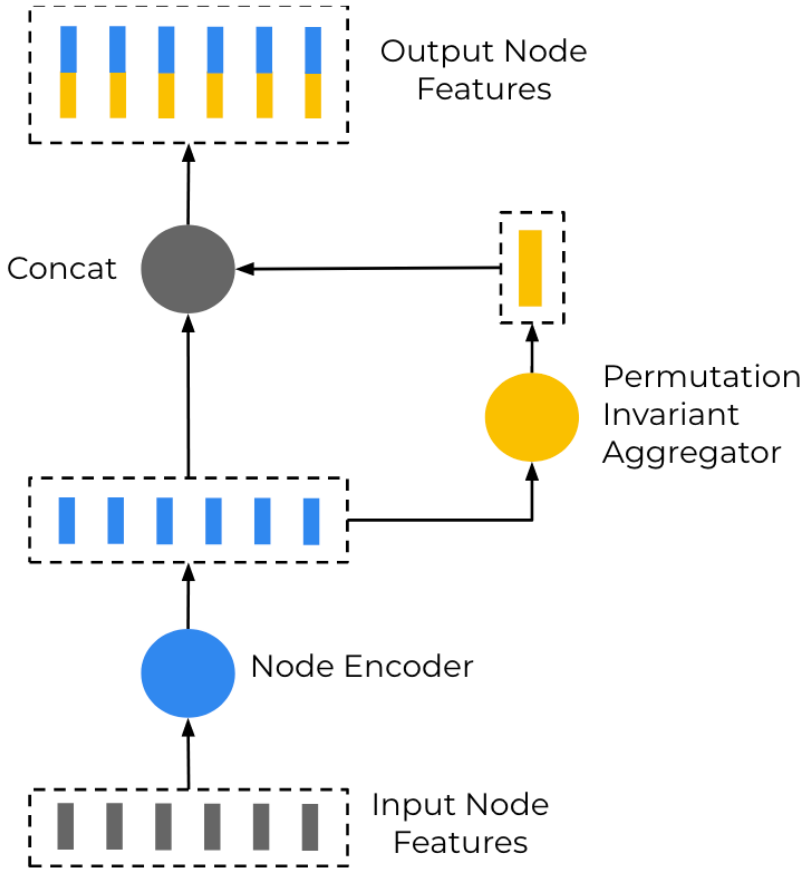
VectorNet

Encoding HD Maps and Agent Dynamics from Vectorized Representation

VectorNet

VectorNet: Encoding HD Maps and Agent Dynamics From Vectorized Representation

Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, Cordelia Schmid;
Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 11525-11533

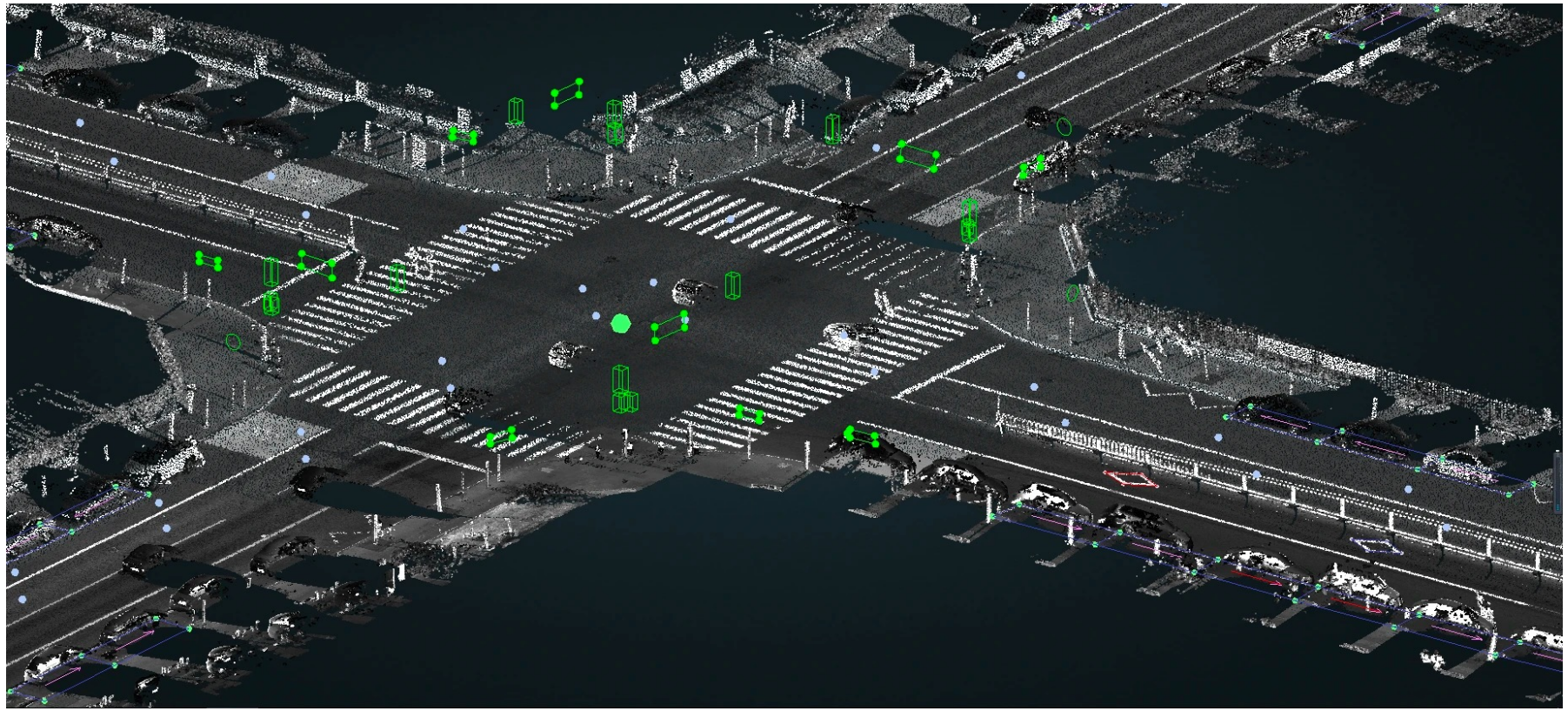


Abstract

Behavior prediction in dynamic, multi-agent systems is an important problem in the context of self-driving cars, due to the complex representations and interactions of road components, including moving agents (e.g. pedestrians and vehicles) and road context information (e.g. lanes, traffic lights). This paper introduces VectorNet, a hierarchical graph neural network that first exploits the spatial locality of individual road components represented by vectors and then models the high-order interactions among all components. In contrast to most recent approaches, which render trajectories of moving agents and road context information as bird-eye images and encode them with convolutional neural networks (ConvNets), our approach operates on the primitive vector representation. By operating on the vectorized high definition (HD) maps and agent trajectories, we avoid lossy rendering and computationally intensive ConvNet encoding steps. To further boost VectorNet's capability in learning context features, we propose a novel auxiliary task to recover the randomly masked out map entities and agent trajectories based on their context. We evaluate VectorNet on our in-house behavior prediction benchmark and the recently released Argoverse forecasting dataset. Our method achieves on par or better performance than the competitive rendering approach on both benchmarks while saving over 70% of the model parameters with an order of magnitude reduction in FLOPs. It also obtains state-of-the-art performance on the Argoverse dataset.

VectorNet

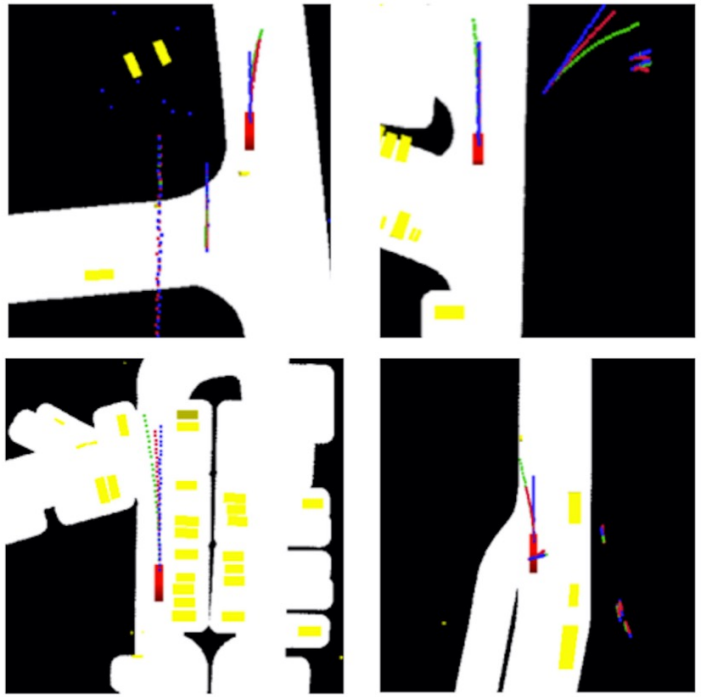
HD map: a highly accurate map used in autonomous driving, containing details not normally present on traditional maps. Such maps can be precise at a centimeter level.

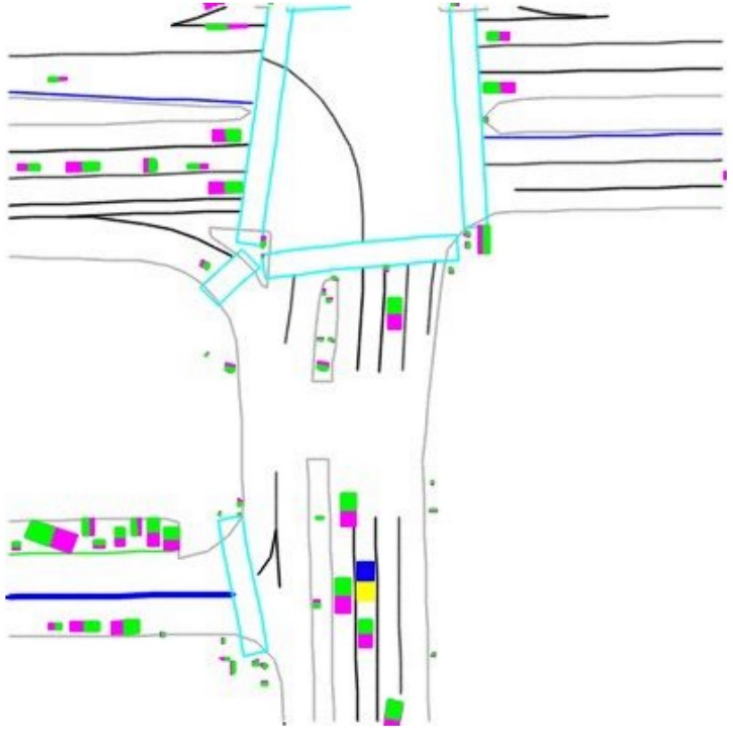


VectorNet

HD map representation – rasterized map

Complex, requires manual specifications, ConvNets on images have limited receptive fields

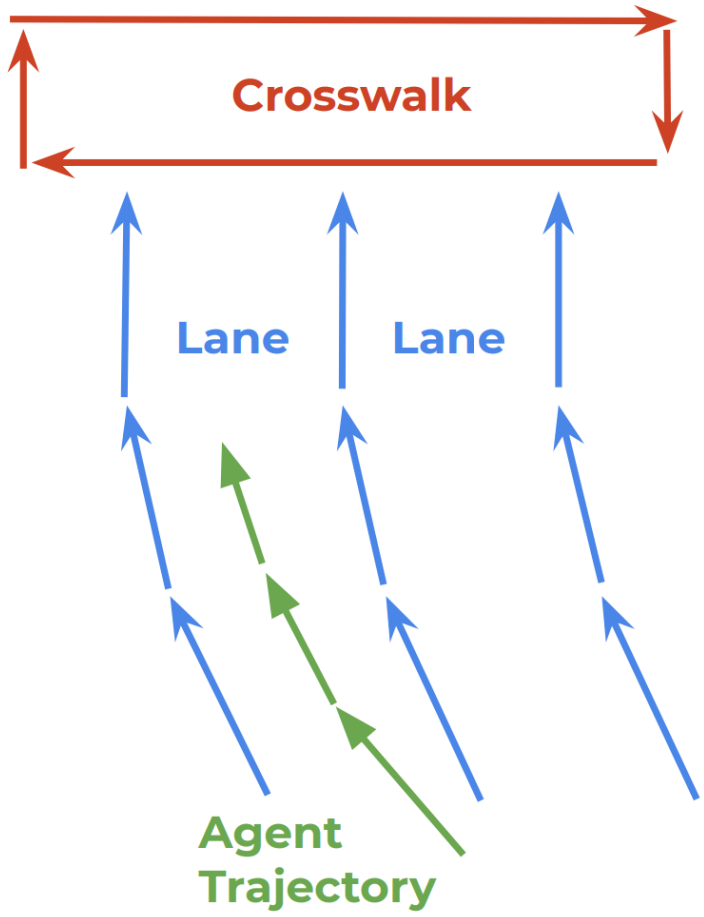




VectorNet

Q: Can we learn a meaningful context representation directly from the structured HD maps?

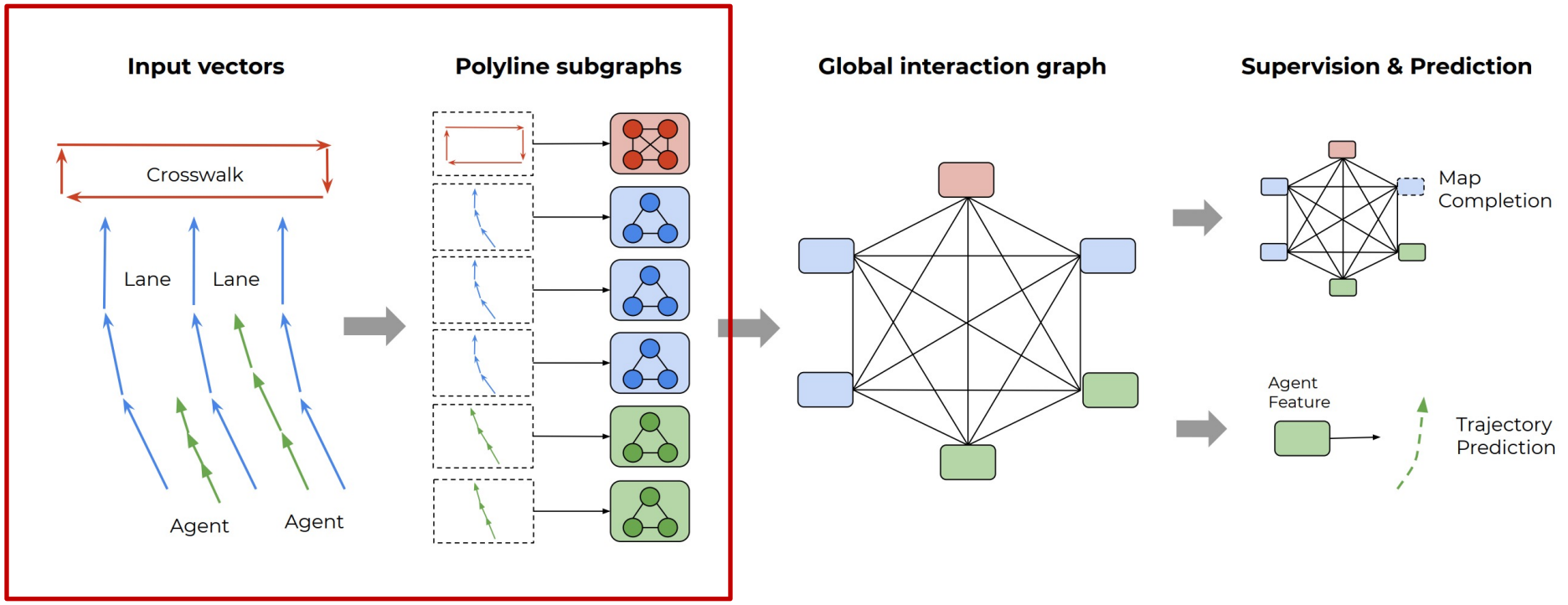
A: Yes, use vectorized form!



Vectorized Representation

VectorNet

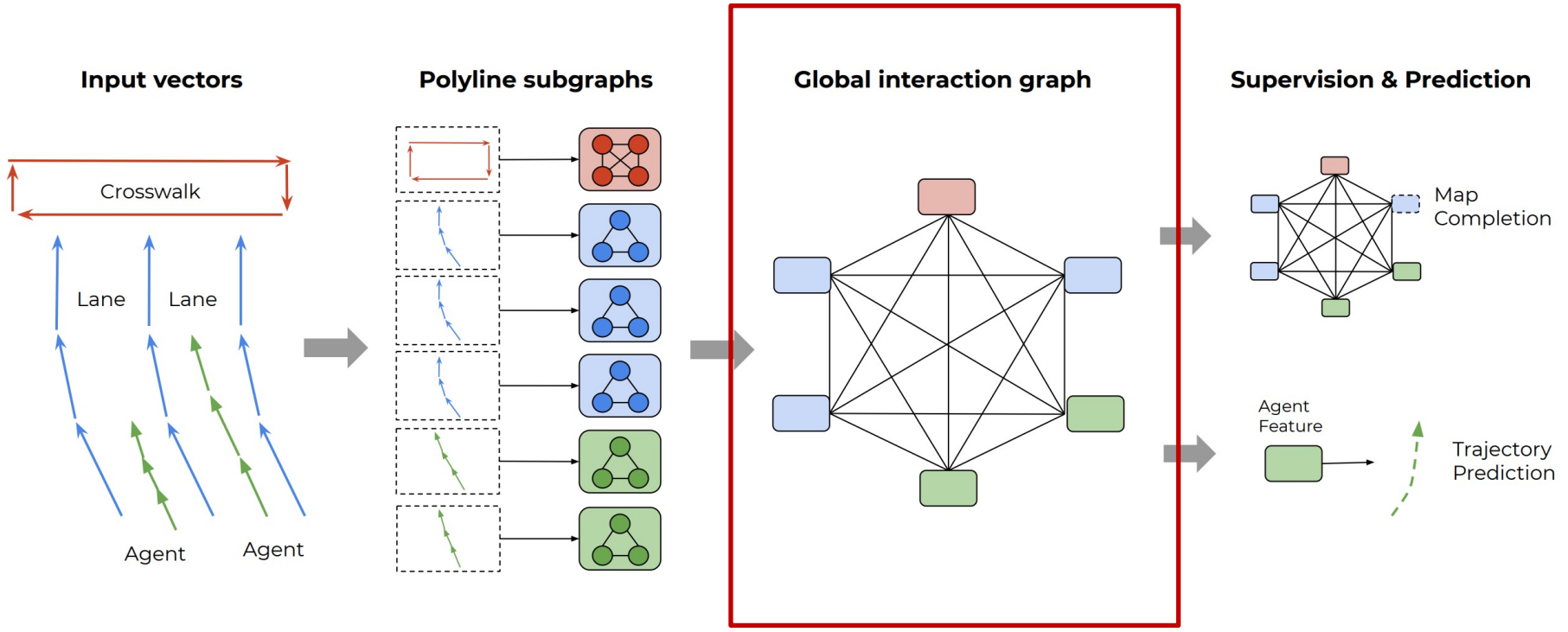
Overview



build vector map/calculate map features

VectorNet

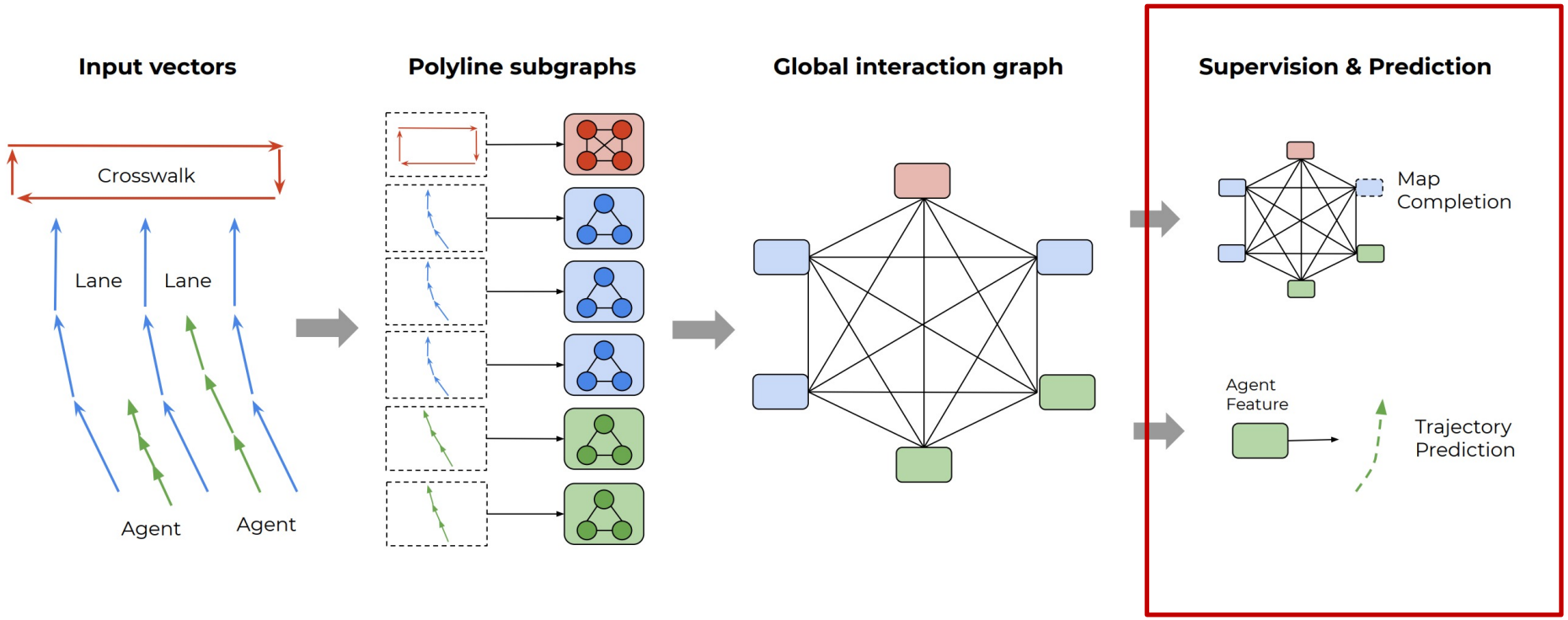
Overview



Trajectory/map interaction

VectorNet

Overview



Prediction

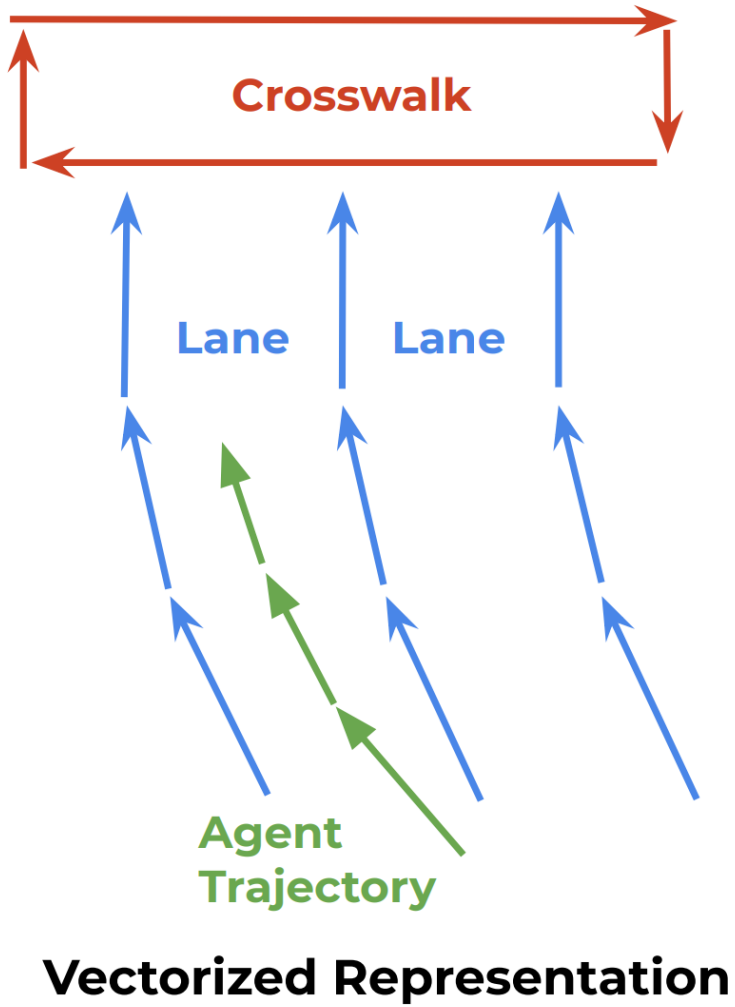
VectorNet

Q: How to build vectorized maps?

- Treat each vector v_i belonging to a polyline $\mathcal{P}_j$ as a node in the graph with node features given by:

$$v_i = [d_i^s, d_i^e, a_i, j]$$

start, end, attribute features, integer id



Vectorized Representation

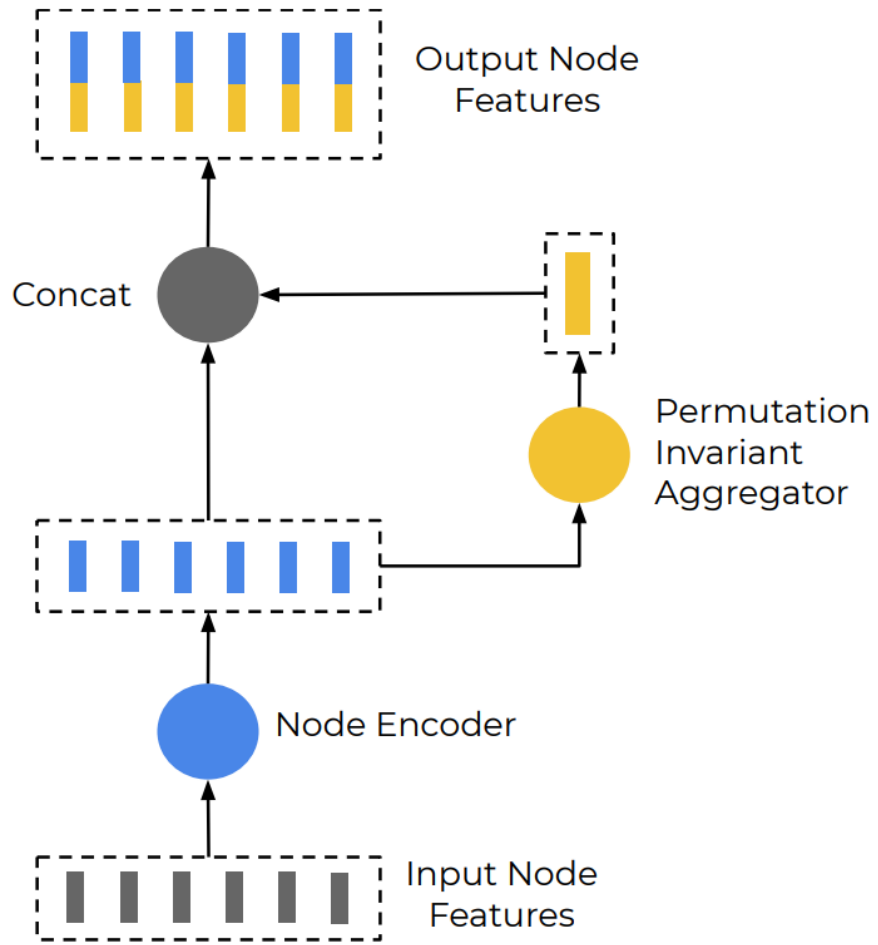
VectorNet

- Considering a polyline $\mathcal{P}$ with its nodes $\{v_1, v_2, \ldots, v_P\}$, we can construct the polyline subgraphs

$$v_i^{(l+1)} = \varphi_{\text{rel}}\left(g_{\text{enc}}\left(v_i^{(l)}\right), \varphi_{\text{agg}}\left(\left\{g_{\text{enc}}\left(v_i^{(l)}\right)\right\}\right)\right)$$

- To obtain polyline level features, we compute

$$\boldsymbol{p} = \varphi_{\text{agg}}\left(\left\{v_i^{(L_p)}\right\}\right)$$

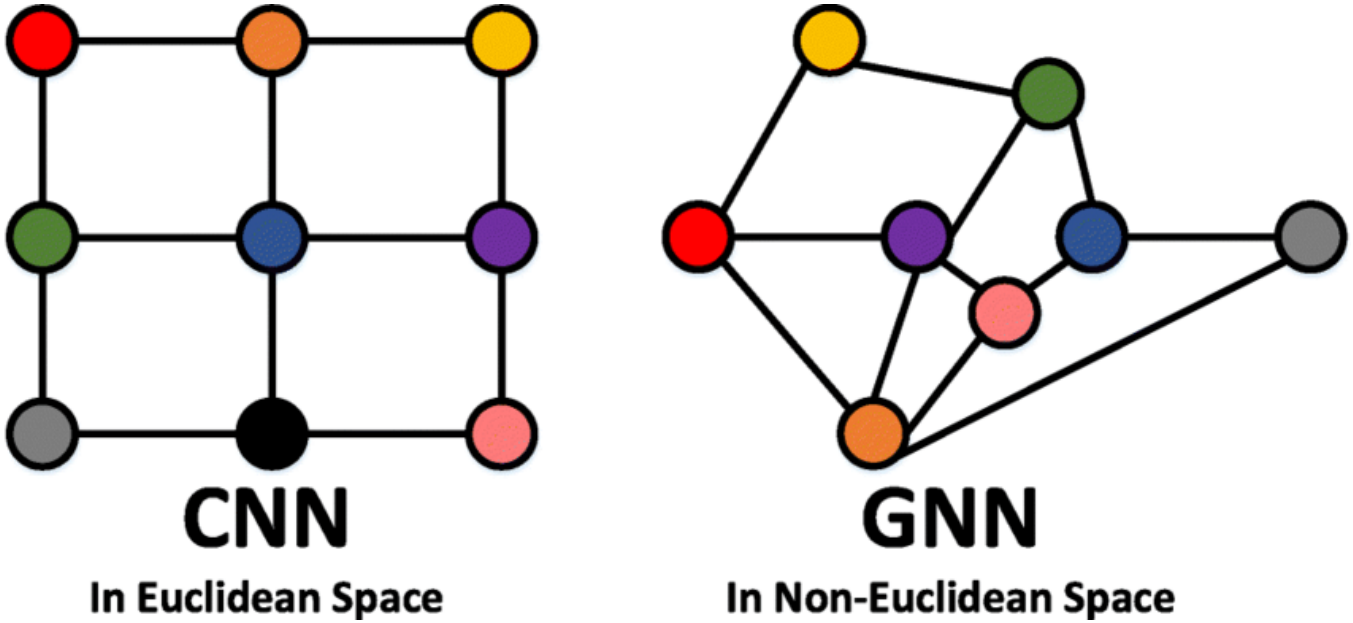


GNN

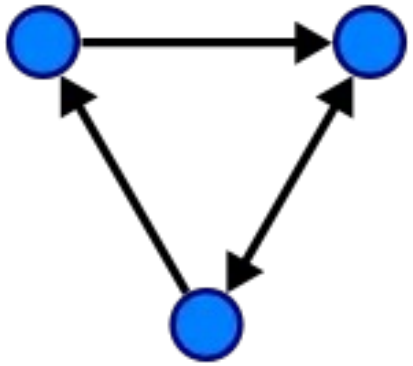
Q: How to use vectorized maps?

A: By implementing a Graph Neural Network (GNN)

GNN is a type of Neural Network which directly operates on the Graph structure.



GNN

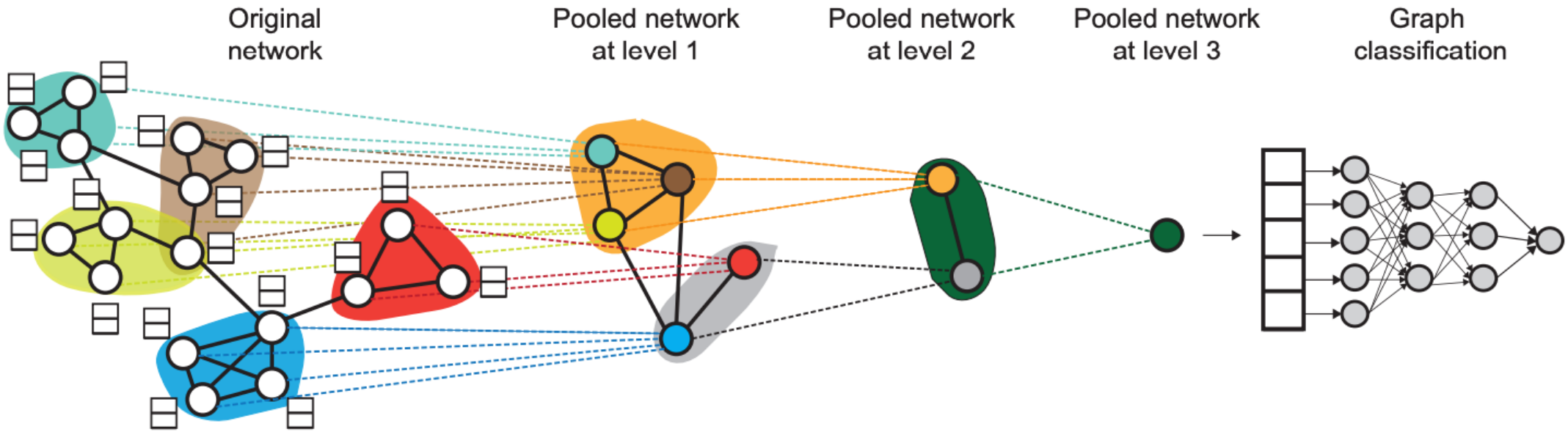


- Graph in discrete math: $G = (V, E)$
- In the original GNN:

$$h_v = f\left(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]}\right)$$

  - x_v: feature of node v
  - t_v: ground-truth label
  - h_v: a vector contains the information of its neighbors
  - $x_{co[v]}$: the features of the edges connecting with v
  - $h_{ne[v]}$: the embedding of the neighbors of v
  - $x_{ne[v]}$: the features of the neighbors of v
- Aggregation: $H^{t+1} = F(H^t, X)$
- Output: $o_v = g(h_v, x_v)$
- Loss: $L = \mathrm{func}(o_v, t_v)$

GNN


Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

VectorNet

- Model the high-order interactions on the polyline node features $\{p_1, p_2, \ldots, p_P\}$ with a global interaction graph:

$$\left\{p_i^{(l+1)}\right\} = \mathrm{GNN}\left(\left\{p_i^{(l)}, \mathcal{A}\right\}\right)$$

  - $\mathcal{A}$: the adjacency matrix for the set of polyline nodes.
- GNN: $\mathrm{GNN}(P) = \mathrm{softmax}\left(P_Q P_K^T\right) P_V$
  - P: is the node feature matrix
  - P_Q, P_K and P_V are its linear projections.
- Decode the future trajectories from the nodes corresponding the moving agents:

$$v_i^{\mathrm{future}} = \varphi_{\mathrm{traj}}\left(p_i^{(L_t)}\right)$$

VectorNet

- The input node feature

$$\widehat{p_i} = \varphi_{\text{node}}\left(p_i^{(L_t)}\right)$$

- To identify an individual polyline node when the corresponding feature of p_i is masked out

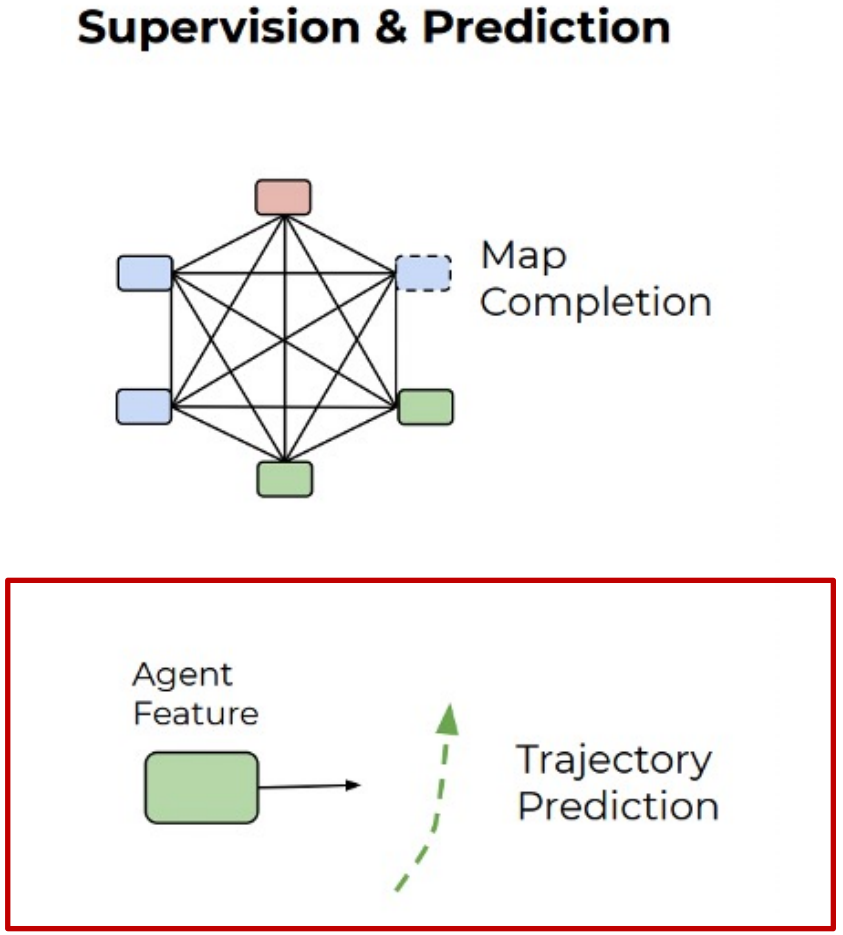
$$p_i^{(0)} = \left[p_i; p_i^{\text{id}}\right]$$

- Loss

$$\mathcal{L} = \mathcal{L}_{\text{traj}} + \alpha\mathcal{L}_{\text{node}}$$

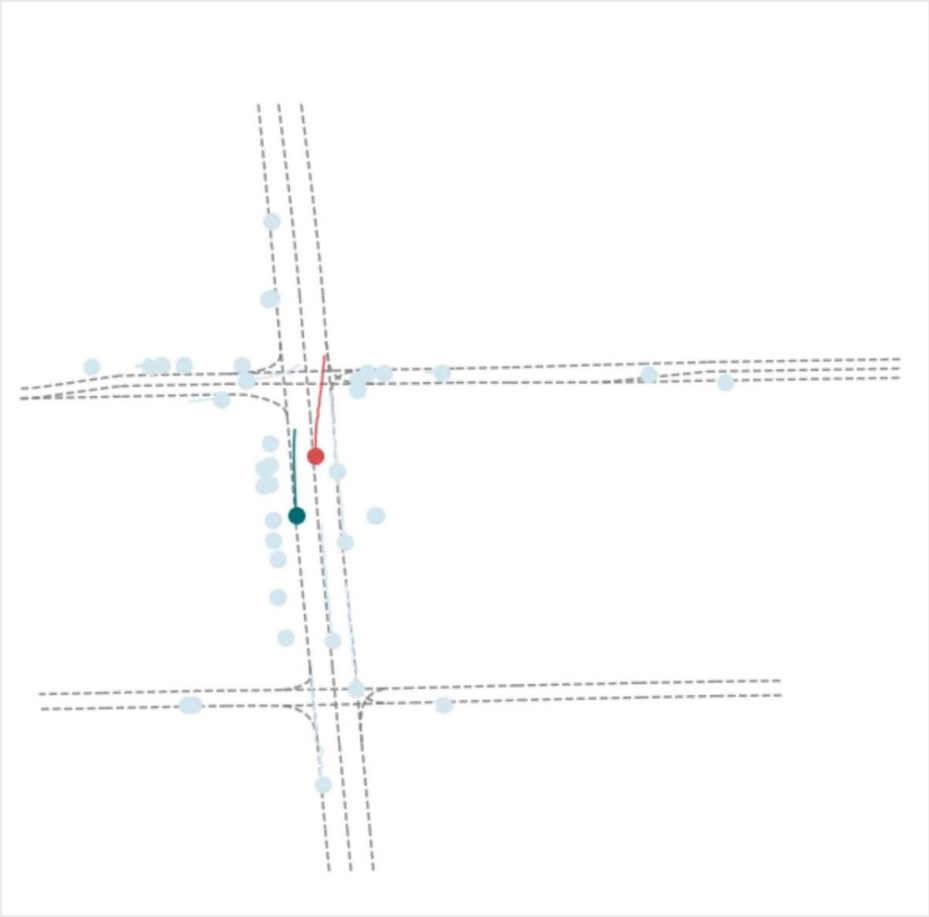
VectorNet

Q: How to predict?



Use MLP to predict final trajectory

Dataset - Argoverse



A still frame from one of the motion forecasting sequences, showing the trajectories for the agent of interest (red), self-driving vehicle (green), and all other objects of interest in the scene (light blue).

Argoverse 1 Motion Forecasting Dataset

A dataset to train and validate motion forecasting models

Argoverse 1 Motion Forecasting is a curated collection of 324,557 scenarios, each 5 seconds long, for training and validation. Each scenario contains the 2D, birds-eye-view centroid of each tracked object sampled at 10 Hz.

To create this collection, we sifted through more than 1000 hours of driving data from our fleet of self-driving test vehicles to find the most challenging segments — including segments that show vehicles at intersections, vehicles taking left or right turns, and vehicles changing lanes.

What makes this dataset stand out?

This dataset is far larger than what can currently be mined from publicly available self-driving datasets, and our HD maps make it easier to predict the motion of objects.

SEGMENT DURATION:
5 seconds

TOTAL NUMBER OF SEGMENTS:
323,557

TOTAL TIME:
320 hours

Restults

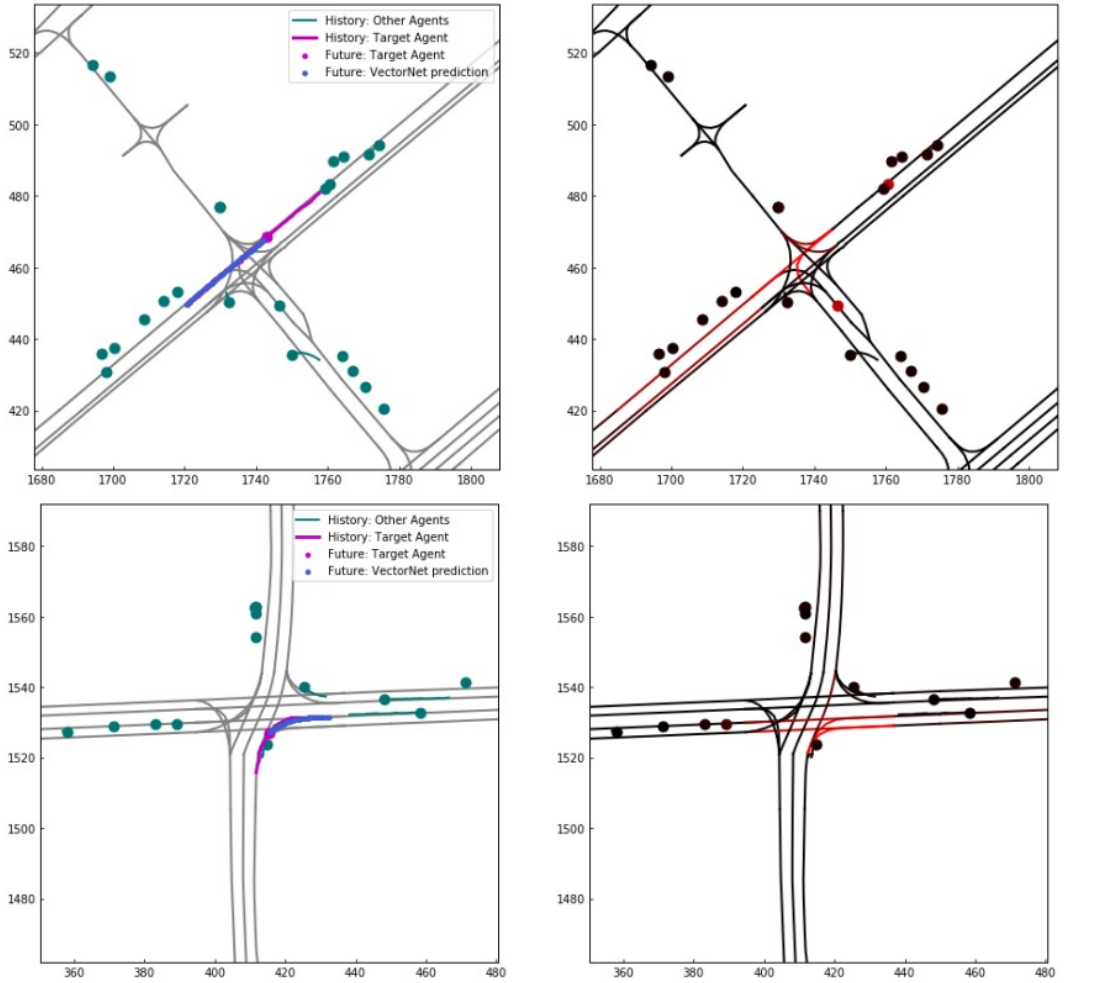
Resolution	Kernel	Crop	In-house dataset				Argoverse dataset			
			DE@1s	DE@2s	DE@3s	ADE	DE@1s	DE@2s	DE@3s	ADE
100×100	3×3	1×1	0.63	0.94	1.32	0.82	1.14	2.80	5.19	2.21
200×200	3×3	1×1	0.57	0.86	1.21	0.75	1.11	2.72	4.96	2.15
400×400	3×3	1×1	0.55	0.82	1.16	0.72	1.12	2.72	4.94	2.16
400×400	3×3	3×3	0.50	0.77	1.09	0.68	1.09	2.62	4.81	2.08
400×400	3×3	5×5	0.50	0.76	1.08	0.67	1.09	2.60	4.70	2.08
400×400	3×3	traj	0.47	0.71	1.00	0.63	1.05	2.48	4.49	1.96
400×400	5×5	1×1	0.54	0.81	1.16	0.72	1.10	2.63	4.75	2.13
400×400	7×7	1×1	0.53	0.81	1.16	0.72	1.10	2.63	4.74	2.13

Table 1. Impact of receptive field (as controlled by convolutional kernel size and crop strategy) and rendering resolution for the ConvNet baseline. We report DE and ADE (in meters) on both the in-house dataset and the Argoverse dataset.

Context	Node Compl.	In-house dataset				Argoverse dataset			
		DE@1s	DE@2s	DE@3s	ADE	DE@1s	DE@2s	DE@3s	ADE
none	-	0.77	0.99	1.29	0.92	1.29	2.98	5.24	2.36
map	no	0.57	0.81	1.11	0.72	0.95	2.18	3.94	1.75
map + agents	no	0.55	0.78	1.05	0.70	0.94	2.14	3.84	1.72
map	yes	0.55	0.78	1.07	0.70	0.94	2.11	3.77	1.70
map + agents	yes	**0.53**	**0.74**	**1.00**	**0.66**	**0.92**	**2.06**	**3.67**	**1.66**

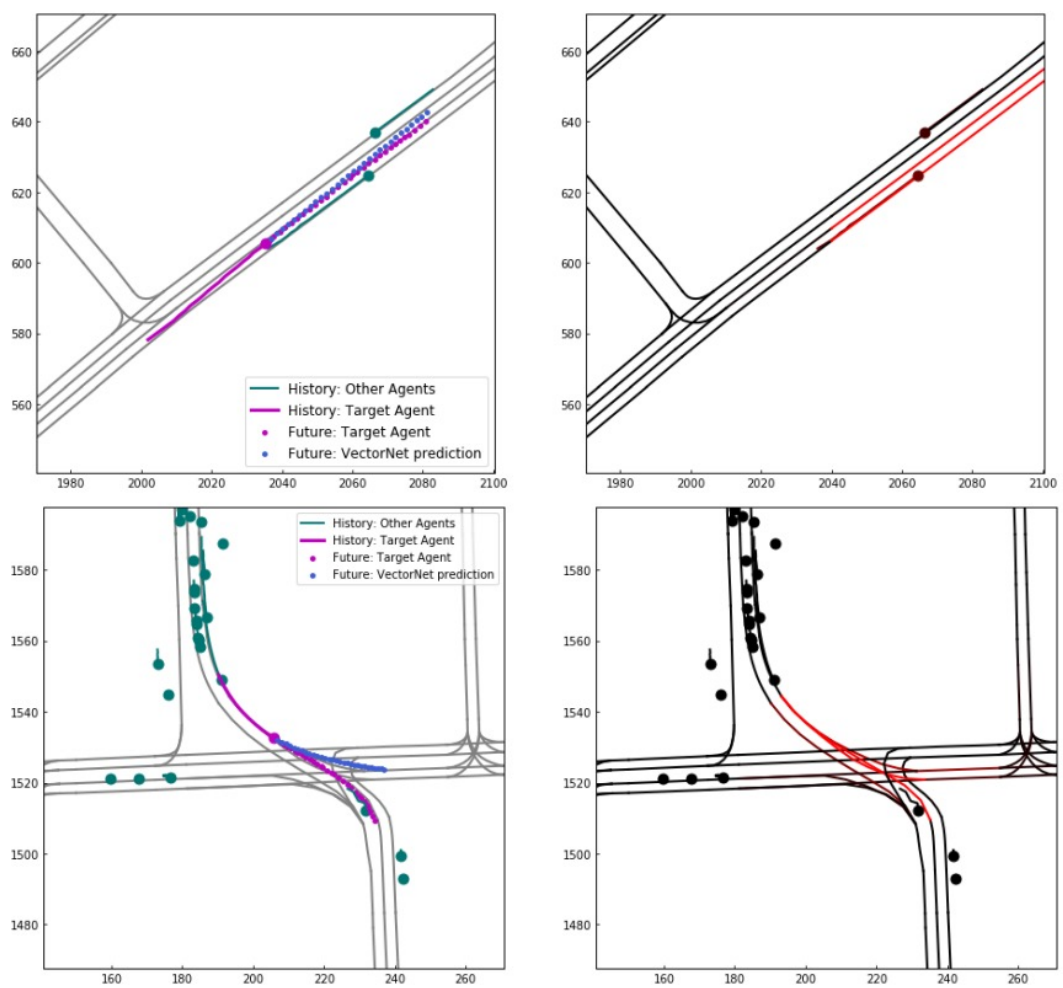
Table 2. Ablation studies for VectorNet with different input node types and training objectives. Here "map" refers to the input vectors from the HD maps, and "agents" refers to the input vectors from the trajectories of non-target vehicles. When "Node Compl." is enabled, the model is trained with the graph completion objective in addition to trajectory prediction. DE and ADE are reported in meters.

Results



prediction

attention for road and agent



prediction

attention for road and agent

05

Q & A

Recommend Reading

- Hu, Yue, et al. "Collaborative motion prediction via neural motion message passing." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020. APA
- Wang, Jingke, et al. "LTP: Lane-Based Trajectory Prediction for Autonomous Driving." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.
- https://paperswithcode.com/task/trajectory-prediction
- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#
- https://distill.pub/2021/gnn-intro/