Learning Lane Graph Representations
for Motion Forecasting

Ming Liang[1]([⊠]), Bin Yang[1,2], Rui Hu[1], Yun Chen[1], Renjie Liao[1,2], Song Feng[1],
and Raquel Urtasun[1,2]

[1] Uber ATG, Pittsburgh, USA
{ming.liang,byang10,rui.hu,yun.chen,rjliao,songf,urtasun}@uber.com
[2] University of Toronto, Toronto, Canada

Abstract. We propose a motion forecasting model that exploits a novel structured map representation as well as actor-map interactions. Instead of encoding vectorized maps as raster images, we construct a lane graph from raw map data to explicitly preserve the map structure. To capture the complex topology and long range dependencies of the lane graph, we propose LaneGCN which extends graph convolutions with multiple adjacency matrices and along-lane dilation. To capture the complex interactions between actors and maps, we exploit a fusion network consisting of four types of interactions, actor-to-lane, lane-to-lane, lane-to-actor and actor-to-actor. Powered by LaneGCN and actor-map interactions, our model is able to predict accurate and realistic multi-modal trajectories. Our approach significantly outperforms the state-of-the-art on the large scale Argoverse motion forecasting benchmark.

Keywords: HD map · Motion forecasting · Autonomous driving

1 Introduction

Autonomous driving has the potential to revolutionize transportation. Self-driving vehicles (SDVs) have to accurately predict the future motions of other traffic participants in order to safely operate. High Definition maps (HD-maps) provide extremely useful geometric and semantic information for motion forecasting, as the behaviors of actors largely depend on the map topology. For example, a vehicle is unlikely to take a left turn when there is not a left turn lane nearby. Effectively exploiting HD maps is essential for motion forecasting models to produce plausible and accurate trajectories.

First attempts exploit HD maps as heuristics [42]. Actors are first associated with lanes and all candidate motion paths are then generated based on map topology. In this way, the prediction results are constrained by the map. However,
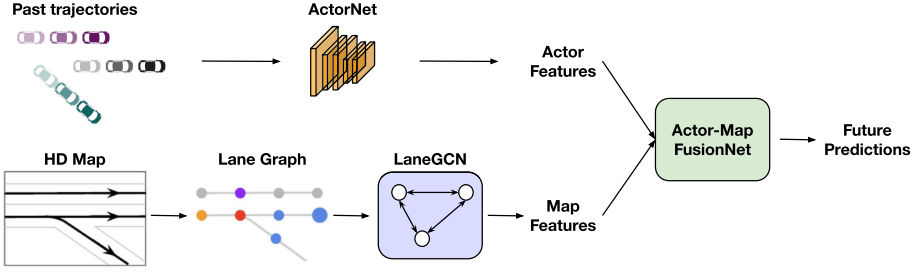
Fig. 1. Our approach: We construct a lane graph from raw map data and use LaneGCN to extract map features. In parallel, ActorNet extracts actor features from observed past trajectories. We then use FusionNet to model the Interactions between actors themselves and the map, and predict the future trajectories.

this approach can not capture rare and non-compliant behaviours, which while not very likely, might be safety critical.

Recent works [3,5–7,14,23,29,38] use machine learning to learn semantic representations from maps. To enable HD maps to be processed by neural networks the map data is rasterized to create image-like raster inputs. Map topology is implicitly encoded as lines, masks or colours, which are then processed by a 2D Convolutional Neural Network (CNN). These learned map features were shown to provide useful context information for motion forecasting. However, these approach has two disadvantages. First, the rasterization process inevitably results in information loss. Second, maps have a graph structure with complex topology which 2D convolution may be very inefficient to capture. For example, a lane of interest may extend for a long range in the lane direction. To capture this information, the receptive field has to be very large, covering not only the intended area, but also large areas outside the lane. Furthermore, lane pairs in the same or opposite directions have completely different semantic meanings and dependencies, although the lanes in both pairs are spatially close to each other.

In this paper we made three main contributions: (1) Instead of using rasterization, we construct a lane graph from vectorized map data, thus avoiding information loss. We then propose the Lane Graph Convolutional Network (LaneGCN), which effectively captures the complex topology and long range dependencies of the lane graph. (2) Based on LaneGCN, our motion forecasting model captures all possible actor-map interactions. In particular, we represent both actors and lanes as nodes in the graph and use a 1D CNN and LaneGCN to extract the features for the actor and lane nodes respectively, and then exploit spatial attention and another LaneGCN to model four types of interactions: *actor-to-lane*, *lane-to-lane*, *lane-to-actor* and *actor-to-actor*. We refer the reader to Fig. 1 for an illustration of our approach. (3) We conduct experiments on the large-scale Argoverse motion forecasting benchmark [9], and show significant improvements over the state-of-the-art.

2   Related Work

In this section, we review work on map representations, learning map representations for autonomy tasks, and graph convolutional networks.

Map Representations: HD maps capture both the lane geometry as well as their connectivity. [21] proposes to parameterize the lane boundaries as a set of polylines, and exploit a Recurrent Neural Network (RNN) to extract them from sensor data. [28] further extends the polyline representation to a more structured para3ization. Instead of modelling the geometry of each lane, [22] proposes to parameterize the unknown lane graph as a Directed Acyclic Graphical model (DAG), which is more robust and able to handle more complex topology like branching. In addition to modelling the geometry, [32,33] encode different lane types in a graphical model to better exploit their appearance features. [11] parameterizes the road layout using an undirected graph, showcasing outstanding performance in large-scale city scale road topology.

Learning Map Representations for Autonomy: Rasterization based map representations have been extensively used. [10,12,14] rasterize map elements (roads, crosswalks) as layers and encode the lane direction with different colors. [3,8] encode roadmap, traffic lights and speed limits in rasterized bird's eye view images. [23] encodes the history of static entities, dynamic entities and semantic map information in a top-down spatial grid. HDNet [38] exploits the road mask as input feature to improve object detection performance. Rasterized maps have been fused with LiDAR point clouds to perform joint perception and prediction [4,27,29] as well as end-to-end motion planning [35,40,41]. While raster map representations are popular, an alternative is to use vectorized map features. [9] uses the distance along the centerlines and offset from the centerlines as input to their nearest neighbours regression and LSTM [20] models. [1,34] use 1D CNN and LSTM to encode lane features. In contrast, our model constructs a lane graph from vectorized map data, and extracts multi-scale topology features using the proposed LaneGCN. In concurrent work VectorNet[16], two graph networks are used to extract actor/lane features and model global interactions, respectively. There are two major differences between VectorNet and LaneGCN. First, VectorNet uses vanilla graph networks with undirected full connections, while we build a sparsely connected lane graph following the map topology and propose task specific multi-type and dilated graph operators. Second, VectorNet uses polyline-level nodes for interaction, while our LaneGCN uses polyline segments as map nodes to capture higher resolution. Note that in our approach nodes in different polylines can interact with each other through dilated connections.

Graph Convolutional Networks: Graph Convolutional Networks (GCNs) [13,15,19,26,30,36] have been shown to be effective for graph representation learning. They generalize the 2D convolution on grids to arbitrary graphs via the so called graph convolution. Different from 2D convolution, which operates on neighbors in a local grid, graph convolution operates on the neighboring nodes defined by the graph structure, typically described in the form of an adjacency matrix. We draw inspiration from GCNs and propose LaneGCN, which is a
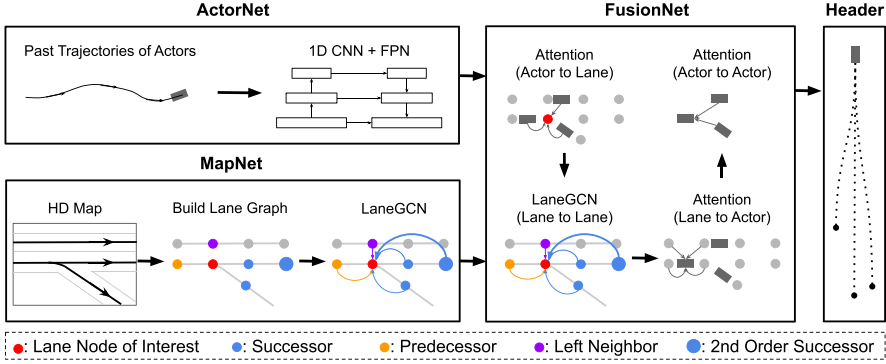
Fig. 2. Overall architecture: Our model is composed of four modules. (1) ActorNet receives the past actor trajectories as input, and uses 1D convolution to extract actor node features. (2) MapNet constructs a lane graph from HD maps, and uses a LaneGCN to exact lane node features. (3) FusionNet is a stack of 4 interaction blocks. The actor to lane block fuses real-time traffic information from actor nodes to lane nodes. The lane to lane block propagates information over the lane graph and updates lane features. The lane to actor block fuses updated map information from lane nodes to actor nodes. The actor to actor block performs interactions among actors. We use another LaneGCN for the lane to lane block, and spatial attention layers for the other blocks. (4) The prediction header uses after-fusion actor features to produce multi-modal trajectories.

specialized version designed for lane graphs. In our model, we introduce multiple adjacency matrices and multi-scale dilated convolutions, which are effective in capturing the complex topology and long-range dependencies of the lane graph.

3   Lane Graph Representations for Motion Forecasting

In this section, we propose a novel motion forecasting model that learns structured map representations and fuses the information of traffic actors and HD maps taking into account their interactions. In the following, we explain the four modules that compose our model, *i.e.,* how to compute actor features with **ActorNet**, how to represent the map via **MapNet**, how to fuse the information from both actors and the map with **FusionNet**, and finally how to predict the final motion forecasting trajectories through the **Prediction Header**. We refer the reader to Fig. 2 for an illustration of the overall architecture.

3.1   ActorNet: Extracting Traffic Participant Representations

We assume actor data is composed of the observed past trajectories of all actors in the scene. Each trajectory is represented as a sequence of displacements $\{\Delta\mathbf{p}_{-(T-1)}, \ldots, \Delta\mathbf{p}_{-1}, \Delta\mathbf{p}_0\}$, where $\Delta\mathbf{p}_t$ is the 2D displacement from time step $t-1$ to t, and T is the trajectory size. All coordinates are defined in the Bird's Eye View (BEV), as this is the space of interest for traffic agents. For trajectories with sizes smaller than T, we pad them with zeros. We add a binary $1 \times T$ mask
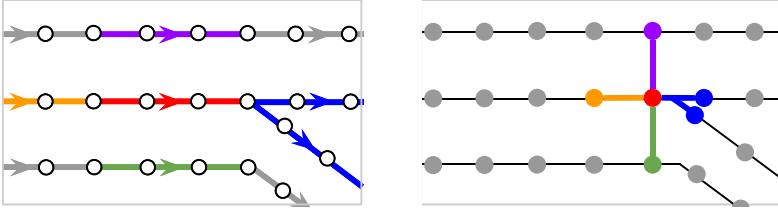
Fig. 3. Lane graph construction from vectorized map data. **Left**: The lane centerline of interest, its predecessor, successor, left and right neighbor are denoted with red, orange, blue, purple, and green lines, respectively. Each centerline is given as a sequence of BEV points (hollow circles). Right: Derived lane graph with an example lane node. The lane node of interest, its predecessor, successor, left and right neighbor are denoted with red, orange, blue, purple and green circles respectively. See Sect. 3.2 for more information. (Color figure online)

to indicate if the element at each step is padded or not and concatenate it with the trajectory tensor, resulting in an input tensor of size $3 \times T$.

While both CNNs and RNNs can be used for temporal data, here we use an 1D CNN to process the trajectory input for its effectiveness in extracting multi-scale features and efficiency in parallel computing. The output of ActorNet is a temporal feature map, whose element at $t = 0$ is used as the actor feature. The network has 3 groups/scales of 1D convolutions. Each group consists of 2 residual blocks [18], with the stride of the first block as 2. We then use a Feature Pyramid Network (FPN) [31] to fuse the multi-scale features, and apply another residual block to obtain the output tensor. For all layers, the convolution kernel size is 3 and the number of output channels is 128. Layer normalization [2] and the Rectified Linear Unit (ReLU) [17] are used after each convolution.

3.2 MapNet: Extracting Structured Map Representation

We use a novel deep model, called MapNet, to learn structured map representations from vectorized map data. This contrasts previous approaches, which encode the map as a raster image and apply 2D convolutions to extract features. MapNet consists of two steps: (1) building a lane graph from vectorized map data; (2) applying our novel LaneGCN to the lane graph to output the map features.

Map Data: In this paper, we adopt a simple form of vectorized map data as our representation of HD maps. Specifically, the map data is represented as a set of lanes and their connectivity. Each lane contains a centerline, *i.e.,* a sequence of 2D BEV points, which are arranged following the lane direction (see Fig. 3, top). For any two lanes which are directly reachable, 4 types of connections are given: *predecessor*, *successor*, *left neighbour* and *right neighbour*. Given a lane A, its predecessor and successor are the lanes which can directly travel to A and from A respectively. Left and right neighbours refer to the lanes which can be directly reached without violating traffic rules. This simple map format provides essential geometric and semantic information for motion forecasting, as vehicles generally plan their routes by reference to lane centerlines and their connectivity.

Lane Graph Construction: Instead of encoding maps as raster images, we derive a lane graph from the map data as the input. In designing the lane graph, we expect its nodes to have a fine resolution. Given any actor location, we query the lane graph and find its nearest nodes to retrieve accurate map information. From this point of view, it is not an optimal choice to directly use the lane centerlines as the nodes.

We refer the reader to Fig. 3 for an example of the lane graph construction. We first define a lane node as the straight line segment formed by any two consecutive points (grey circles in Fig. 3) of the centerline. The location of a lane node is the averaged coordinates of its two end points. Following the connections between lane centerlines, we also derive 4 connectivity types for the lane nodes, *i.e., predecessor, successor, left neighbour* and *right neighbour*. For any lane node A, its predecessor and successor are defined as the neighbouring lane nodes that can travel to A or from A respectively. Note that one can reach the first lane node of a lane l_A from the last lane node of lane l_B if l_B is the predecessor of l_A. Left and right neighbours are defined as the spatially closest lane node measured by ℓ_2 distance on the left and on the right neighbouring lane respectively. We denote the lane nodes with $V \in \mathbb{R}^{N \times 2}$, where N is the number of lane nodes and the ith row of V is the BEV coordinates of the ith node. We represent the connectivity with 4 adjacency matrices $\{A_i\}_{i \in \{\text{pre,suc,left,right}\}}$, with $A_i \in \mathbb{R}^{N \times N}$. We denote $A_{i,jk}$, as the element in the jth row and kth column of A_i. Then $A_{i,jk} = 1$ if node k is an i-type neighbor of node j.

LaneConv Operator: A natural operator to handle lane graphs is the graph convolution [36]. The most widely used graph convolution operator [26] is defined as $Y = LXW$, where $X \in \mathbb{R}^{N \times F}$ is the node feature, $W \in \mathbb{R}^{F \times O}$ is the weight matrix, and $Y \in \mathbb{R}^{N \times O}$ is the output. The graph Laplacian matrix $L \in \mathbb{R}^{N \times N}$ takes the form $L = D^{-1/2}(I + A)D^{-1/2}$, where I, A and D are the identity, adjacency and degree matrices respectively. I and A account for self connection and connections between different nodes. All connections share the same weight W, and the degree matrix D is used to normalize the output. However, this vanilla graph convolution is inefficient in our case due to the following reasons. First, it is not clear what kind of node feature will preserve the information in the lane graphs. Second, a single graph Laplacian can not capture the connection type, *i.e.,* losing the directional information carried by the connection type. Third, it is not straightforward to handle long range dependencies , *e.g.,* akin dilated convolution, within this form of graph convolution. Motivated by these challenges, we introduce our novel specially designed operator for lane graphs, called *LaneConv*.

Node Feature: We first define the input feature of the lane nodes. Each lane node corresponds to a straight line segment of a centerline. To encode all the lane node information, we need to take into account both the shape (size and orientation) and the location (the coordinates of the center) of the corresponding line segment. We parameterize the node feature as follows,

$$\mathbf{x}_i = \text{MLP}_{\text{shape}}\left(\mathbf{v}_i^{\text{end}} - \mathbf{v}_i^{\text{start}}\right) + \text{MLP}_{\text{loc}}\left(\mathbf{v}_i\right), \tag{1}$$

where MLP indicates a multi-layer perceptron and the two subscripts refer to shape and location, respectively. $\mathbf{v}_i$ is the location of the ith lane node, *i.e.,* the center between two end points, $\mathbf{v}_i^{\text{start}}$ and $\mathbf{v}_i^{\text{end}}$ are the BEV coordinates of the node i's starting and ending points, and $\mathbf{x}_i$ is the ith row of the node feature matrix X, denoting the input feature of the ith lane node.

LaneConv: The node feature above only captures the local information of a line segment. To aggregate the topology information of the lane graph at a larger scale, we design the following LaneConv operator

$$Y = XW_0 + \sum_{i \in \{\text{pre,suc,left,right}\}} A_i X W_i, \tag{2}$$

where A_i and W_i are the adjacency and the weight matrices corresponding to the ith connection type respectively. Since we order the lane nodes from the start to the end of the lane, A_{suc} and A_{pre} are matrices obtained by shifting the identity matrix one step towards upper right (non-zero superdiagonal) and lower left (non-zero subdiagonal). A_{suc} and A_{pre} can propagate information from the forward and backward neighbours whereas A_{left} and A_{right} allow information to flow from the cross-lane neighbours. It is not hard to see that our LaneConv builds on top of the general graph convolution and encodes more geometric (*e.g.,* connection type/direction) information. As shown in our experiments this improves over the vanilla graph convolution.

Dilated LaneConv: Since motion forecasting models usually predict the future trajectories of actors with a time horizon of several seconds, actors with high speed could have moved a long distance. Therefore, the model needs to capture the long range dependency along the lane direction for accurate prediction. In regular grid graphs, a dilated convolution operator [39] can effectively capture the long range dependency by enlarging the receptive field. Inspired by this operator, we propose the *dilated LaneConv* operator to achieve a similar goal for irregular graphs.

In particular, the k-dilation LaneConv operator is defined as follows,

$$Y = XW_0 + A_{\text{pre}}^k X W_{\text{pre},k} + A_{\text{suc}}^k X W_{\text{suc},k}, \tag{3}$$

where A_{pre}^k is the kth matrix power of A_{pre}. This allows us to directly propagate information along the lane for k steps, with k a hyperparameter. Since A_{pre}^k is highly sparse, one can efficiently compute it using sparse matrix multiplication. Note that the dilated LaneConv is only used for predecessor and successor, as the long range dependency is mostly along the lane direction.

LaneGCN: Based on the dilated LaneConv, we further propose a multi-scale LaneConv operator and use it to build our LaneGCN. Combining Eq. (2) and (3) with multiple dilations, we get a multi-scale LaneConv operator with C dilation sizes as follows

$$Y = XW_0 + \sum_{i \in \{\text{left,right}\}} A_i X W_i + \sum_{c=1}^{C} \left(A_{\text{pre}}^{k_c} X W_{\text{pre},k_c} + A_{\text{suc}}^{k_c} X W_{\text{suc},k_c} \right), \tag{4}$$
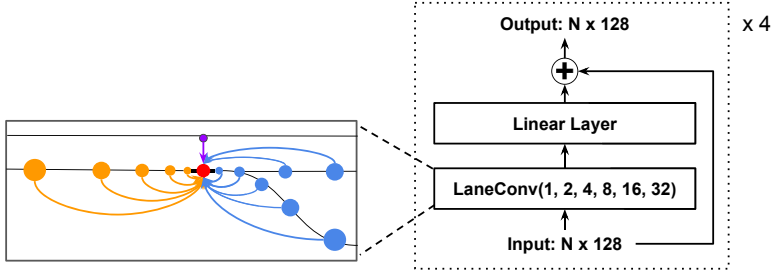
Fig. 4. LaneGCN architecture. Our LaneGCN is a stack of 4 multi-scale LaneConv residual blocks, each of which consists of a LaneConv(1,2,4,8,16,32) and a linear layer with a residual connection [18]. All layers have 128 feature channels.

where k_c is the cth dilation size. We denote LaneConv$(k_1, \cdots, k_C)$ this multi-scale layer. The architecture of LaneGCN is shown in Fig. 4. The network is composed of 4 LaneConv residual [18] blocks, which are the stack of a LaneConv(1, 2, 4, 8, 16, 32) and a linear layer, as well as a shortcut. All layers have 128 feature channels. Layer normalization [2] and ReLU [17] are used after each LaneConv and linear layer.

3.3 FusionNet

In this section we propose a network to fuse the information of the actor and lane nodes given by ActorNet and MapNet, respectively. The behaviour of an actor strongly depends on its context, *i.e.,* other actors and the map. Although the interactions between actors has been explored by previous work, the interactions between the actors and the map, and map conditioned interactions between actors have received much less attention. In our model, we use spatial attention and LaneGCN to capture a complete set of actor-map interactions (see Fig. 2).

We build a stack of four fusion modules to capture all information flows between actors and lane nodes, *i.e.,* actors to lanes (A2L), lanes to lanes (L2L), lanes to actors (L2A) and actors to actors (A2A). Intuitively, A2L introduces real-time traffic information to lane nodes, such as blockage or usage of the lanes. L2L updates lane node features by propagating the traffic information over the lane graph. L2A fuses updated map features with real-time traffic information back to the actors. A2A handles the interactions between actors and produces the output actor features, which are then used by the prediction header for motion forecasting.

We implement L2L using another LaneGCN, which has the same architecture as the one used in our MapNet (see Sect. 3.2). In the following we describe the other three modules in detail. We exploit a spatial attention layer [37] for A2L, L2A and A2A. The attention layer applies to each of the three modules in the same way. Taking A2L as an example, given an actor node i, we aggregate the features from its context lane nodes j as follows

$$\mathbf{y}_i = \mathbf{x}_i W_0 + \sum_j \phi(\text{concat}(\mathbf{x}_i, \Delta_{i,j}, \mathbf{x}_j)W_1)W_2, \tag{5}$$

with $\mathbf{x}_i$ the feature of the ith node, W a weight matrix, ϕ the composition of layer normalization and ReLU, and $\Delta_{ij} = \text{MLP}(\mathbf{v}_j - \mathbf{v}_i)$, where $\mathbf{v}$ denotes the node location. The context nodes are defined to be the lane nodes whose ℓ_2 distance from the actor node i is smaller than a threshold. The thresholds for A2L, L2A and A2A are set to $7\,\text{m}$, $6\,\text{m}$, and $100\,\text{m}$ respectively. Each of A2L, L2A and A2A has two residual blocks, which consist of a stack of the proposed attention layer and a linear layer, as well as a residual connection. All layers have 128 output feature channels.

3.4   Prediction Header

Taking the after-fusion actor features as input, a multi-modal prediction header outputs the final motion forecasting. For each actor, it predicts K possible future trajectories and their confidence scores. The header has two branches, a regression branch to predict the trajectory of each mode and a classification branch to predict the confidence score of each mode. For the mth actor, we apply a residual block and a linear layer in the regression branch to regress the K sequences of BEV coordinates:

$$O_{m,\text{reg}} = \{(\mathbf{p}_{m,1}^k, \mathbf{p}_{m,2}^k, ..., \mathbf{p}_{m,T}^k)\}_{k \in [0, K-1]} \tag{6}$$

where $\mathbf{p}_{m,i}^k$ is the predicted mth actor's BEV coordinates of the kth mode at the ith time step. For the classification branch, we apply an MLP to $\mathbf{p}_{m,T}^k - \mathbf{p}_{m,0}$ to get K distance embeddings. We then concatenate each distance embedding with the actor feature, apply a residual block and a linear layer to output K confidence scores, $O_{m,\text{cls}} = (c_{m,0}, c_{m,1}, ..., c_{m,K-1})$.

3.5   Learning

As all the modules are differentiable, we can train the model in an end-to-end way. We use the sum of classification and regression losses to train the model

$$L = L_{\text{cls}} + \alpha L_{\text{reg}}, \tag{7}$$

where $\alpha = 1.0$. Given K predicted trajectories of an actor, we find a positive trajectory $\hat{k}$ that has the minimum final displacement error, $i.e.$, the Euclidean distance between the predicted and ground truth locations at the final time step.

For classification, we use the max-margin loss:

$$L_{\text{cls}} = \frac{1}{M(K-1)} \sum_{m=1}^{M} \sum_{k \neq \hat{k}} \max(0, c_{m,k} + \epsilon - c_{m,\hat{k}}) \tag{8}$$

where ϵ is the margin and M is the total number of actors. For regression, we apply the smooth $\ell 1$ loss on all predicted time steps:

$$L_{\text{reg}} = \frac{1}{MT} \sum_{m=1}^{M} \sum_{t=1}^{T} \text{reg}(\mathbf{p}_{m,t}^{\hat{k}} - \mathbf{p}_{m,t}^{*}) \tag{9}$$

where $\mathbf{p}_t^*$ is the ground truth BEV coordinates at time step t, $\text{reg}(\mathbf{x}) = \sum_i d(x_i)$, x_i is the ith element of $\mathbf{x}$, and $d(x_i)$ is the smooth $\ell 1$ loss defined as

$$d(x_i) = \begin{cases} 0.5x_i^2 & \text{if } \|x_i\| < 1 \\ \|x_i\| - 0.5 & \text{otherwise,} \end{cases} \tag{10}$$

where $\|x_i\|$ denotes the ℓ_1 norm of x_i.

4   Experimental Evaluation

We evaluate our model on the large scale Argoverse [9] motion forecasting benchmark, which is publicly available and provides vectorized map data. We first compare our model with the state-of-the-art and show significant improvements in all metrics. We then conduct ablation studies on the architecture and LaneConv operators, and show the advantage of our model design choices. Finally, we show qualitative results and discuss future directions.

4.1   Experimental Settings

Dataset: Argoverse [9] is a motion forecasting benchmark with over 30K scenarios collected in Pittsburgh and Miami. Each scenario is a sequence of frames sampled at 10 HZ. Each sequence has an interesting object called "agent", and the task is to predict the future locations of agents in a 3 s future horizon. The sequences are split into training, validation and test sets, which have 205942, 39472 and 78143 sequences respectively. These splits have no geographical overlap. For the training and validation sets, each sequence lasts for 5 s. The first 2 s are used as input data and the other 3 s are used as ground truth for models to predict. For the test set, only the first 2 s are provided. Each frame is given as the centroid coordinates of all objects in the scene. The actor data is a trajectory of 20 time steps. The map data is a set of lane centerlines and their connectivity. We use both actor and map data in the way described in Sects. 3.1 and 3.2, without any other preprocessing step. We did not use the other map data such as the rasterized drivable area map and ground height map provided with the benchmark.

Metrics: We employ two extensively used motion forecasting metrics, *Average Displacement Error* (ADE) is defined as the ℓ_2 distance between the predicted and ground truth locations, averaged over all steps. *Final Displacement Error* (FDE) is defined as the ℓ_2 distance between the predicted and ground truth locations at the last step in the predicted horizon. As motion forecasting is by nature multi-modal, Argoverse uses the minimum ADE (minADE) and minimum FDE (minFDE) of the top K predictions as the metrics. When K=1, minADE

Table 1. Results on Argoverse motion forecasting benchmark (test set)

Model	K = 1			K = 6		
	minADE	minFDE	MR	minADE	**minFDE**	MR
Argoverse Baseline [9]	2.96	6.81	0.81	2.34	5.44	0.69
Argoverse Baseline (NN) [9]	3.45	7.88	0.87	1.71	3.29	0.54
Holmes (*7th*) [24]	2.91	6.54	0.82	1.38	2.66	0.42
cxx (*3rd*) [1]	1.91	4.31	0.66	0.99	1.71	0.19
uulm-mrm (*2nd*) [12,14]	1.90	4.19	0.63	0.94	1.55	0.22
Jean (*1st*) [1,34]	1.86	4.18	0.63	0.93	1.49	0.19
Our model	**1.71**	**3.78**	**0.59**	**0.87**	**1.36**	**0.16**

and minFDE are equal to the deterministic ADE and FDE. Argoverse benchmark allows up to 6 predictions, and the online server ranks the entries with minFDE with K = 6. We use minADE and minFDE for K = 1 and K = 6 as the main metrics. When comparing our model with top entries on the leaderboard, we also show *Miss Rate* (MR), which is the ratio of predictions (the best mode) whose final location is more than 2.0 m away from the ground truth.

Implementation Details: We use all actors and lanes whose distance from the agent is smaller than 100 m as the input. The coordinate system in our model is the BEV centered at the agent location at $t = 0$. We use the orientation from the agent location at $t = -1$ to the agent location at $t = 0$ as the positive x axis. We train the model on 4 TITAN-X GPUs using a batch size of 128 with the Adam [25] optimizer with an initial learning rate of 1×10^{-3}, which is decayed to 1×10^{-4} at 32 epochs. The training process finishes at 36 epochs and takes about 11.5 h. All our results are based on the same model, whose architecture and hyper-parameters are described in Sect. 3.

4.2 Results

Comparison with the State-of-the-Art: We compare our model with four top entries and two official baselines on the Argoverse motion forecasting leaderboard. We submit our result at the time of ECCV submission (2020/03/15). The metrics are minADE, minFDE and MR for K = 1 and K = 6, and the leaderboard is ranked by minFDE for K = 6. As shown in Table 1, our model significantly outperforms all other models in all metrics. Among the compared methods, uulm-mrm encodes the input data using a rasterization approach [12,14]. They represent actor states, lanes and the drivable area with a synthesized image, which is then processed by a 2D CNN. In this approach, map topology and actor-map interactions are both implicitly learned by 2D convolution. In contrast, our model explicitly learns structured map features and performs actor-map fusion. Jean and cxx encode actors and lanes with 1D CNN and/or LSTM, and use attention [37] to fuse the features. In their models, lanes are encoded independently so the global map topology is not captured. Moreover, there is no actor

Table 2. Ablation study results of modules

Backbone		FusionNet				K = 1		K = 6	
ActorNet	MapNet	L2A	A2L	L2L	A2A	minADE	minFDE	minADE	minFDE
✓						1.90	4.38	0.91	1.66
✓					✓	1.58	3.61	0.79	1.29
✓	✓	✓				1.55	3.52	0.76	1.23
✓	✓	✓	✓	✓		1.39	3.05	0.72	1.10
✓	✓	✓	✓	✓	✓	**1.35**	**2.97**	**0.71**	**1.08**

to lane and lane to lane fusion. In contrast, our model learns the lane features using the LaneConv, which captures the multi-scale topology of the lane graph.

Importance of Each Module: In Table 2, we show the results of using Actor-Net as the baseline and progressively adding more modules. Three observations can be drawn from the results. First, all modules improve the performance of the model, demonstrating the effectiveness of both LaneGCN and our overall architecture. Second, the information flow from actors to maps brings useful traffic information which benefits the motion forecasting performance, as the incorporation of A2L and L2L significantly outperforms L2A only. Third, A2L, L2L and L2A also facilitates the interaction between actors, which can be seen from the smaller gain of adding A2A to this combination (from 4th row to 5th row) compared to adding A2A to ActorNet alone (from 1st row to 2nd row). Intuitively, the information of different actors is propagated over the lane graph and leads to effective map conditioned interactions.

Lane Graph Operators: In Table 3, we show the results of the ablation study on lane graph operators. The baseline model uses the combination of A2L, L2L and L2A. We start from the vanilla graph convolution (GraphConv), and evaluate the effect of adding each component of the LaneConv block (see Fig. 4), including the residual block, multi-type connections and dilation. The last row is the LaneConv used in our model (fourth row of Table 2). All these components significantly improve the performance. The residual block only adds about 7% parameters, but effectively facilitates the training. Both multi-type connections and dilation significantly boost the performance, demonstrating the clear advantage of LaneConv over vanilla graph convolution.

Qualitative Results: In Fig. 5, we compare qualitatively our model to other methods on 4 hard cases. The results of other models are adapted from the slides of Argoverse motion forecasting competition [1]. As the examples are from the test set and we have no access to the labels, in our results we did not show the ground truth trajectory. The first row shows a case where the baselines miss the mode. While the other methods fail to capture the right turn prediction, our model produces a mode which nicely follows the right turn centerline. The second row shows a case where the agent is waiting to perform an unprotected left turn for the first 2 s. Due to the lack of actor motion history, maps are important

Table 3. Ablation study results of lane graph operators

Component				K=1		K=6	
GraphConv	Residual	Multi-Type	Dilate	minADE	minFDE	minADE	minFDE
✓				1.72	3.93	0.82	1.41
✓		✓		1.59	3.59	0.77	1.24
✓			✓	1.46	3.29	0.74	1.16
✓	✓			1.53	3.48	0.79	1.33
✓	✓	✓		1.48	3.33	0.74	1.19
✓	✓		✓	1.41	3.12	0.73	1.14
✓	✓	✓		**1.39**	**3.05**	**0.72**	**1.10**

Trajectories end with a circle. ■ Ground truth ■ Predicted trajectory ■ Past trajectory
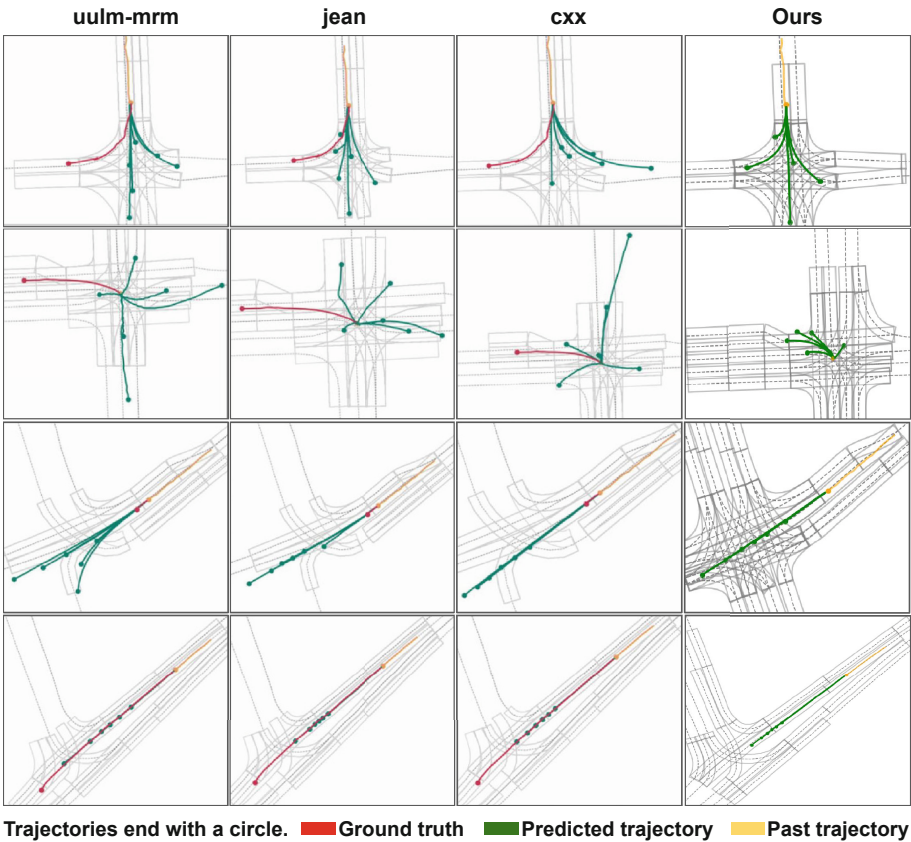
Fig. 5. Qualitative results on hard cases. From top to bottom, these hard cases involve missing the right turn mode, lacking history information, extreme deceleration and acceleration, respectively. See the text for more information.

for the model to produce reasonable trajectories. The other models produce divergent trajectories, some of which are non-traffic-rule compliant. In contrast, our model produces reasonable trajectories following the lane topology. The third row shows a case of a car decelerating and coming to a stop at the intersection. Our model produces a mode with more deceleration then the baselines and all the modes reasonably follow the lane. The fourth row shows a case of extreme acceleration. None of the models captures this case well, possibly because there is not enough information to make this prediction.

Overall, these results suggest that LaneGCN effectively learns structured map representations, which are used by the model to predict realistic trajectories. One potential way to improve our model is to incorporate more map information into the lane graph. Currently our model uses the centerlines and their connectivity. Other map information, such as traffic lights and traffic signs, provides useful information for motion forecasting, which is well illustrated by the second and third cases in Fig. 5. To account for new map data, our model can be easily extended by introducing new nodes and connections. We will explore this direction in future work.

5   Conclusion

In this paper, we propose a novel motion forecasting model to learn lane graph representations and perform a complete set of actor-map interactions. Instead of using a rasterized map as input, we construct a lane graph from vectorized map data and propose the LaneGCN to extract map topology features. We use spatial attention and the LaneGCN to fuse the information of both actors and lanes. We conduct experiments on the large scale Argoverse motion forecasting benchmark. Our model significantly outperforms the state-of-the-art. In the future we plan to explore the incorporation of other map data.

References

1. ArgoAI challenge. NeurIPS Workshop on Machine Learning for Autonomous Driving. https://slideslive.com/38923162/argoai-challenge (2019)
2. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv:1607.06450 (2016)
3. Bansal, M., Krizhevsky, A., Ogale, A.: Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. arXiv:1812.03079 (2018)
4. Casas, S., Gulino, C., Liao, R., Urtasun, R.: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In: ICRA (2020)
5. Casas, S., Gulino, C., Suo, S., Luo, K., Liao, R., Urtasun, R.: Implicit latent variable model for scene-consistent motion forecasting. In: Proceedings of the European Conference on Computer Vision (ECCV) (2020)
6. Casas, S., Gulino, C., Suo, S., Urtasun, R.: The importance of prior knowledge in precise multimodal prediction. In: IROS (2020)
7. Casas, S., Luo, W., Urtasun, R.: Intentnet: Learning to predict intention from raw sensor data. In: Conference on Robot Learning, pp. 947–956 (2018)

8. Chai, Y., Sapp, B., Bansal, M., Anguelov, D.: Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. arXiv:abs/1910.05449 (2019)
9. Chang, M.F., et al.: Argoverse: 3D tracking and forecasting with rich maps. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8748–8757 (2019)
10. Chou, F.C., et al.: Predicting motion of vulnerable road users using high-definition maps and efficient convnets. arXiv:abs/1906.08469 (2019)
11. Chu, H., et al.: Neural turtle graphics for modeling city road layouts. In: ICCV (2019)
12. Cui, H., et al.: Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 2090–2096 (2018)
13. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
14. Djuric, N., et al.: Motion prediction of traffic actors for autonomous driving using deep convolutional networks. arXiv:1808.05819 (2018)
15. Duvenaud, D.K., et al.: Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems, pp. 2224–2232 (2015)
16. Gao, J., et al.: 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)
17. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
19. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data. arXiv:1506.05163 (2015)
20. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
21. Homayounfar, N., Ma, W.C., Lakshmikanth, S.K., Urtasun, R.: Hierarchical recurrent attention networks for structured online maps. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3417–3426 (2018)
22. Homayounfar, N., Ma, W.C., Liang, J., Wu, X., Fan, J., Urtasun, R.: Dagmapper: Learning to map by discovering lane topology. In: ICCV (2019)
23. Hong, J., Sapp, B., Philbin, J.: Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8454–8462 (2019)
24. Huang, X., et al.: Diversity-aware vehicle motion prediction via latent semantic sampling. arXiv:1911.12736 (2019)
25. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
26. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
27. Li, L., et al.: End-to-end contextual perception and prediction with interaction transformer. In: IROS (2020)
28. Liang, J., Homayounfar, N., Ma, W.C., Wang, S., Urtasun, R.: Convolutional recurrent network for road boundary extraction. In: CVPR (2019)

29. Liang, M., et al.: PnPNet: End-to-end perception and prediction with tracking in the loop. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11553–11562 (2020)
30. Liao, R., Zhao, Z., Urtasun, R., Zemel, R.S.: LanczosNet: Multi-scale deep graph convolutional networks. arXiv:1901.01484 (2019)
31. Lin, T.Y., Dollár, P., Girshick, R.B., He, K., Hariharan, B., Belongie, S.J.: Feature pyramid networks for object detection. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 936–944 (2016)
32. Máttyus, G., Wang, S., Fidler, S., Urtasun, R.: Enhancing road maps by parsing aerial images around the world. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1689–1697 (2015)
33. Máttyus, G., Wang, S., Fidler, S., Urtasun, R.: HD maps: Fine-grained road segmentation by parsing ground and aerial images. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3611–3619 (2016)
34. Mercat, J., Gilles, T., Zoghby, N.E., Sandou, G., Beauvois, D., Gil, G.P.: Multi-head attention for multi-modal joint vehicle motion forecasting. arXiv:1910.03650 (2019)
35. Sadat, A., Casas, S., Ren, M., Wu, X., Dhawan, P., Urtasun, R.: Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In: Proceedings of the European Conference on Computer Vision (ECCV) (2020)
36. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag. **30**(3), 83–98 (2013)
37. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
38. Yang, B., Liang, M., Urtasun, R.: HDNET: Exploiting HD maps for 3D object detection. In: Conference on Robot Learning, pp. 146–155 (2018)
39. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. arXiv:1511.07122 (2015)
40. Zeng, W., et al.: End-to-end interpretable neural motion planner. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2019)
41. Zeng, W., Wang, S., Liao, R., Chen, Y., Yang, B., Urtasun, R.: DSDNet: Deep structured self-driving network. In: ECCV (2020)
42. Ziegler, J., et al.: Making bertha drive—an autonomous journey on a historic route. IEEE Intell. Transp. Syst. Mag. **6**(2), 8–20 (2014)