Discrete Interaction Prediction Using Graph Neural Nets

Dagmawi Haile [*] **Sam Ubellacker** [*]

[1]Massachusetts Institute of Technology
77 Massachusetts Avenue Cambridge, Massachusetts 02139
dagmawi@mit.edu, subella@mit.edu

Abstract

Considering interactions between multiple agents in a traffic environment is critical for autonomous vehicles to operate safely. This paper aims to implement a learning based approach based on Graph Neural networks to predict interaction types for vehicles in a 4-way intersection. Specifically, we offer two main contributions: an automatic data generation pipeline developed in a custom simulator and a Graph Neural net model to predict pairwise interactions between agents. We further extend the capabilities of state-of-the-art approaches by explicitly accounting for emergency vehicles in our model. We provide empirical results in simulation for our model and show it is capable of reaching 85% accuracy on our test set.

Motivation

Multi-agent trajectory prediction has yielded three types of models: context representation, generative, and interaction. Exciting work has been presented on the overlap between these models. The model proposed in TrafficGraphNet offers a novel combination of context representation and interaction modeling.[6, 5] Although the results from TrafficGraphNet showed state-of-the-art performance, it presents an opportunity for extensions that could be implemented to improve its discrete interaction prediction/labeling.

Background

Initial solutions to trajectory prediction for autonomous vehicles primarily looked at the challenge through single-agent dynamics lenses. However, as the field progressed, it shifted to a multi-agent context in which single-agent dynamics were combined with multi-agent scene information to achieve more accurate trajectory prediction. The shift to a multi-agent viewpoint in solving the autonomous trajectory vehicle prediction led to three paradigms - context representation models, generative models, and interaction models.

Context representation models solve the multi-agent trajectory prediction problem by predicting a single optimal trajectory for a vehicle through learning about agents' states and contextual information in an agent's environment. Data is encoded via recurrent dynamics for agent states and dense raster maps for contextual information. Some of the core works in this paradigm include RasterNet, CoverNet, and ChauffeurNet, which use CNNs to learn and predict agent trajectories using rasters.

Interaction models solve the multi-agent trajectory prediction by predicting trajectories based on the interactions between agents in a multi-agent setting. Much of this work stems from human interaction in SocialLSTM and Convolutional Social Pooling (CSP) models.[1, 4] Advancements in this paradigm have led to models that combine context representation with interaction information, as seen in SpAGNN and Implicit Latent Variable Model. [3, 2]

Recent work has been dominated by the interplay between context representation and interaction models. Models such as SpAGNN and CSP implicitly factor in discrete agent interaction using social poling, attention, and message passing.[2, 4] Other models such as TrafficGraphNet combine context representation with explicitly supervised discrete interactions.

TrafficGraphNet proposes a robust model for combining context representation and explicit discrete interaction modeling through a dual goal loss function predicated on trajectory and pairwise interaction labels.[5] However, the model on predicts discrete interaction labels for an entire scene rather than at each time step of a scene. We believe there remains room for exploring the impact using a more granular set of discrete interaction labels as it may improve the accuracy of trajectory predictions when combined with dynamics.

Problem statement

We consider a simplified traffic scenario that involves two agents i and j at a 4-way intersection. Each agents motion is constrained to follow the road along only one dimension. We seek to predict interaction labels for the two agents in real time. In particular, at each time step our algorithm should assign each agent a label of either going, yielding, or ignoring. Here, going means the agent will continue moving forward, yielding means the agent will wait for the other agent to cross, and ignoring means the agent does not consider the actions of the agent. Furthermore, we extend traditional approaches by explicitly accounting for how interactions change in the presence of emergency vehicles.

[*]These authors contributed equally.

Our problem consists of two main stages: data generation and interaction prediction. We present a simulation environment and heuristic-based algorithm which allows for training data to be rapidly generated. This data is fed into a Graph Neural Net (GNN) which can then perform predictions on interactions in our simulation environment.

Method

Data Generation

Learning based approaches require substantial amounts of training data to generalize well to various scenarios. Because collecting this data can be very time consuming for a human, we seek to automate the process by utilizing a heuristic based algorithm described in [**?** 5]. To summarize, the algorithm assigns interaction labels by comparing the future trajectories of agents i and j according to the cases below:

- **GOING** - Agent i is **going** with respect to agent j if the two future trajectories intersect and agent i arrives at the intersection point first.
- **YIELDING** - Agent i is **yielding** with respect to agent j if the two future trajectories intersect and agent i arrives at the intersection point second.
- **IGNORING** - Agent i is **ignoring** with respect to agent j if the two future trajectories do not intersect.

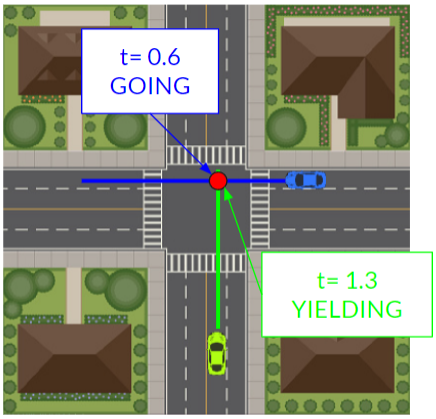

Figure 1: A visualization of our heuristic algorithm. The blue and green lines represent the future trajectories of the blue and green car, respectively. The intersection time is determined for each agent. Here, we have $t^{int}_{blue} < t^{int}_{green}$, so the blue car is labelled as going while the green car yields.

Contrasting to [6] we define future trajectories by considering distance elapsed rather than time. This provides a more robust labelling method that can properly label cases when the agent is stationary. Formally, we define the future trajectory of agent i as $s_i^{t_0:t_1}$, where t_0 is the current time and t_1 is solved by computing the discrete trajectory integral:

$$\mathbf{d} = \sum_{t=t_0}^{t_1} ||p_i^{t+1} - p_i^t||_2 \tag{1}$$

where $\mathbf{d}$, the lookahead, is a constant defining how much distance the future trajectory should span, and p_i^t represents just the x, y positions of s_i^t. The equation can be iteratively solved for an approximate value of t_1 quickly. To accommodate for emergency vehicles, the lookahed $\mathbf{d}$ is increased when a siren is active. This is to reflect the fact that drivers will react to active emergency vehicles from a much further distance away than they would for normal vehicles.

Given future trajectories $s_i^{t_0:t_1}$ and $s_j^{t_0:t1}$, we leverage our 1-Dimensional motion assumption to quickly compute if trajectories intersect. For each agent, a line segment is created from p^{t_0} to p^{t_1}. It then becomes trivial to compare if future trajectories intersect by checking the line segments for intersections. To determine the time of intersection for each agent, we consider p_{ij}^{int}, the point of intersection, and determine the time of the nearest points to p_{ij}^{int}. For example, to compute the time of intersection t_i^{int} for agent i, we compute:

$$t_i^{int} = \arg\min_{p_i} ||p_i^t - p_{ij}^{int}||_2 \tag{2}$$

Thus, we simply compare t_i^{int} and t_j^{int} to determine the interaction label.

Hybrid Graph

In this variant of TrafficGraphNet, we focus on a lightweight model with only traffic actors that serve as the nodes in our graph, represented as a vector t^i. There are two types of traffic actors: police cars and cars. There is a directed edge e_{ij} in the graph connecting agent i to j; the edge e_{ji} also exists. All traffic actors are connected, thus making the graph fully connected. Like TrafficGraphNet, a state encoded f_s encodes the past four states of an actor; therefore, $h_i^s = f_s(s_i^{t-H:t})$, where H is the number of prior states.[5] Each state or timestep of an agent includes the following dynamics: x-position, y-position, x-velocity, and y-velocity. The type of actor will also be encoded into the node attributes. We concatenate t^i and h_i^s via a fuse layer to get the final node embedding h_f^i.

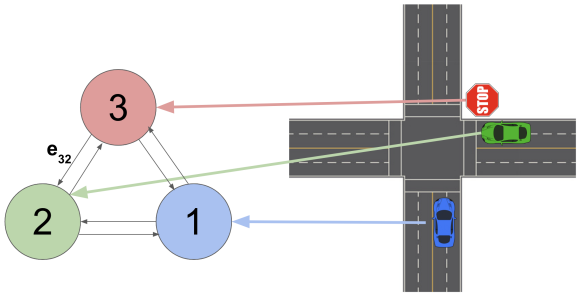

Figure 2: In this figure we see the transcription of an multi-agent interaction. Traffic agents are converted to nodes and are connected to each other via directed edges.

Graph Neural Net

In order to predict the labels for each edge, we will leverage a Graph Neural Net comprised of a node model f_v and an

edge model f_e for each layer. The GNN will first update the edge attributes and then update the node attributes using the following update equations.[6]

$$e_{ij}^k = f_e^k(e_{ij}^{k-1}, h_i^{k-1}, h_j^{k-1}) \tag{3}$$

$$h_{ij}^k = f_n^k(h_i^{k-1}, \sum_{j \in N_i} e_{ij}) \tag{4}$$

The output of the GNN will then be run through an output MLP to get a distribution over the possible embedding.

$$l_{ij} = f_o(e_{ij}^K) \tag{5}$$

Implementation

Data Generation

We designed a custom simulation environment Fig. 3 to streamline our data generation and prediction pipelines. The simulation was created using pygame, a python package which provides functionality for user input and graphics. Each agent in our simulation follows a simplified dynamics model:

$$\mathcal{F} = \mathbf{m}\ddot{p} - \mathbf{b}\dot{p} \tag{6}$$

where $\mathcal{F}$ represents the force on the agent, $\mathbf{m}$ is the agent's mass, $\mathbf{b}$ is a damping coefficent, $\ddot{p}$ is the agent's acceleration, and $\dot{p}$ is the agent's velocity. The user can control both vehicles through keyboard control, which applies a force $\mathcal{F}$ to the agent in the direction of the key pressed. The dynamics solution is then determined by using scipy's odeint function.

The simulation features two modes: data generation and prediction.

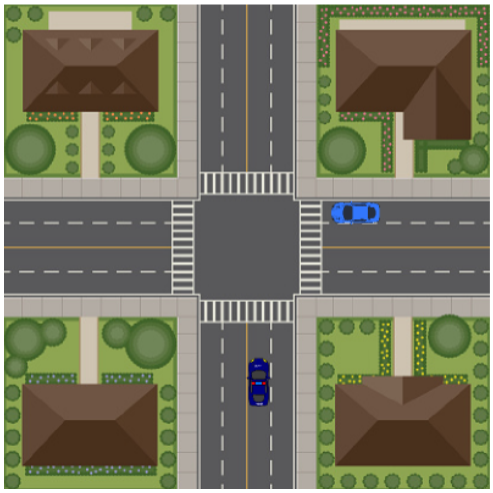

Figure 3: Our custom simulation environment developed using pygame.

The data generation mode provides a framework which allows us to rapidly create datasets for training. The user controls both vehicles with the keyboard to mimic how real drivers would perform for a particular traffic scenario. Once the user is finished, the simulation will enter "playback" mode, where the agent's states are replayed and assigned

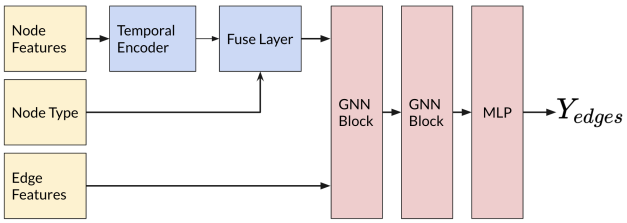

Figure 4: Node features are encoded via an temporal encoder, then passed through a two GNN layers and passed through a final classification 2-layer MLP

interaction labels at each time step using the heuristic described previously. In particular, Eqn. (1) can be iteratively solved and Eqn. (2) can be quickly solved using scipy's KDTree. The above process can be repeated for various scenarios until enough data is generated. The simulation will then write all the relevant simulation information in a json file which can be parsed and used to train the model.

After the model is trained, we can use the trained model to determine interaction labels in real time in the prediction mode of the simulator.

Model

The model used for our experimentation was built using the PyTorch Geometric (PyG) library, which is tailored for writing and training GNNs. In particular, we leverage the MetaLayer graph network, which is based on a similar model to TrafficGraphNet.

We represent our node features as a tensor with dimensions $(N, 5, 4)$, where N is the number of total node/actors, 5 is the number of previous states (including the current state), and 4 is the dynamics features. We represent our edge attributes as a tensor with dimension $(N, 4)$, where N is the number of edges and 4 is the relative dynamics features. These tensors are wrapped in a PyG Data object, which also contains an edge index tensor representing the hybrid graph's connectivity in COO format with dimensions $(2, |E|)$.

Due to the short temporal nature of the state encoder f_s, we implemented it as a 1-D CNN encoder with a weighting vector concatenated along the time dimension to add temporal influence to the convolution. We then took the mean across the different states to convert our $(N, 5, 4)$ tensor into an $(N, 128)$ node encoding.

We implemented the discrete interactions as the *edge attributes* parameter of the PyG Data tensor being passed into our GNN. The *edge attributes* tensor is a 4D tensor with the following relative attributes between actor i with respect to actor j: x-position, y-position, x-velocity, and y-velocity. We then use a two-layer GNN in our model which follows the message passing defined by Eqns. (3, 4). Each layer contains an edge model (f_e) and node model (f_n) that are 2-layer MLPs with an output size of 128 and a hidden dimension of 128. The output layer is a 2-layer MLP with an output of 3 and a softmax output layer. We use a cross-entropy loss with equal weights.

Evaluation

To evaluate the strength of our model we ran the following metrics: top-1 accuracy, confusion matrix, and AUPRC curve. Furthermore, we explored different architectures from those outlined in TrafficGraphNet by varying the amount of hidden units, history horizon, and batch size. These results are explored below.

Test Accuracy

We constructed a testing data set of interactions based on a different distribution of interaction to model the model's performance. Below, we observe the results considering variations in hidden units, history horizon, and batch size. When increasing hidden units from 128 to 256, we see a drop in accuracy across all variations. This suggests that 256 hidden units cause the model to over-fit and not generalize to new observations. Similarly, when increasing the history horizon from 2 seconds to 4 seconds, we see accuracy fall off; this is likely because we are letting dynamics too far away from an interaction influence the label. There was a positive trend as the batch size increased.

Hidden Units	History(s)	Batch Size	**Testing Accuracy**
128	2	32	0.8001
128	**2**	**64**	**0.8516**
128	4	32	0.7890
256	2	32	0.7921
256	2	64	0.8168
256	4	32	0.8184

Confusion Matrix

To better understand how the model was performing on each type of pair-wise interaction, we computed confusion matrices to identify cases of miss classification. Figure 1 shows that the model attained nearly 100% accuracy on IGNORING interactions. YIELDING and GOING interactions have a 68% accuracy, respectively. This highlights that the model is weak at identifying these interaction types even though testing accuracy is relatively high at 85%. The miss classifications of YIELDING have a 3:1 ratio for GOING and IGNORING, respectively. The GOING label is symmetric.

AUPRC Curve

The AUC measures a classifier's ability to distinguish between the three classes. The greater the AUC, the better performance. The AUC of our model is 94.6% which is very strong. However, we know this measure is not the whole image when taken with the confusion matrix. The data is dominated by IGNORING labels, and because our model is nearly perfect at classifying those, the AUC is raised significantly. However, we expect a much lower AUC if this was only classifying YIELDING and GOING.

Demonstration

Below we have provided links to two demonstration of of a two-agent interaction, where interaction labels are being predicted by the GNN for every time-step.

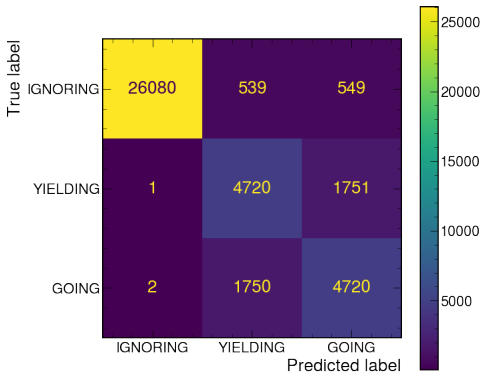

Figure 5: The confusion matrix shows a high accuracy on ignoring labels, and a mediocre performance on yielding and going labels. It highlights the flaws in the classification abilities of our model

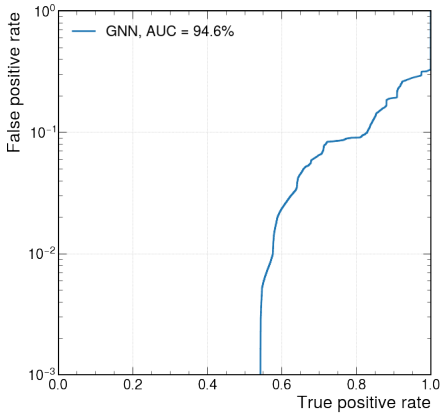

Figure 6: ROC-AUC Curve

1. Standard Vehicle Demo
2. Emergency Vehicle Demo

Discussion

As described in the evaluation, one of the significant insights of this experiment was discovering how an imbalanced data set can lead to high accuracy but a poor model. It is clear that even with nearly $60k$ data points for training, we still needed a richer data set with more YIELDING/GOING to improve the learning of our model. The nearly perfect accuracy on IGNORING interactions suggests that a richer/balanced data set would likely enhance the YIELDING/GOING accuracy.

One of the surprising discoveries of this project was how easy it is to over-fit with a Graph Neural Net structure. In our evaluation, one of the free parameters we explored was our architecture's number of GNN blocks. We saw that increasing the blocks from two to three caused the model to over-fit, which is surprising. However, research suggests that this is likely to occur because our graph only has nodes at most one distance away. Thus having three iterations of message passing would lead to possible over-smoothening, meaning the node/edge attributes of connected components are driven to

appear more similar. Thus the characteristics of the unique interaction types appear more similar, making them harder to distinguish.

Conclusion

This paper explored predicting interaction labels for an example 4-way intersection scenario. We presented a custom simulation environment explicitly designed to streamline data generation and training of our GNN. Our simulation allows users to control each vehicle through keyboard control to mimic a real life scenario, and our heuristic-based labelling algorithm automatically assigns labels. This data set is used to train our GNN model which was emperically shown to have 85% accuracy on our test set. Finally, we offered video demostrations of our approach working in simulation.

Contributions

Sam created the simulation environment and the automatic data generation pipeline. He also helped debug the GNN model and created the interface for it in the simulation

Dagmawi worked on creating the pipeline to convert the data from the simulations to PyG data. Additionally, he worked on researching, implementing, and training the GNN outlined in the Implementation. He also was responsible for creating the ROC-AUC curve and the confusion matrix.

References

[1] Alahi, A.; Goel, K.; Ramanathan, V.; Robicquet, A.; Fei-Fei, L.; and Savarese, S. 2016. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 961–971.

[2] Casas, S.; Gulino, C.; Liao, R.; and Urtasun, R. 2020. SpAGNN: Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data. 9491–9497.

[3] Casas, S.; Gulino, C.; Suo, S.; Luo, K.; Liao, R.; and Urtasun, R. 2020. Implicit Latent Variable Model for Scene-Consistent Motion Forecasting.

[4] Deo, N.; and Trivedi, M. M. 2018. Convolutional Social Pooling for Vehicle Trajectory Prediction.

[5] Kumar, S.; Gu, Y.; Hoang, J.; Haynes, G. C.; and Marchetti-Bowick, M. 2020. Interaction-Based Trajectory Prediction Over a Hybrid Traffic Graph.

[6] Lee, D.; Gu, Y.; Hoang, J.; and Marchetti-Bowick, M. 2019. Joint Interaction and Trajectory Prediction for Autonomous Driving using Graph Neural Networks.