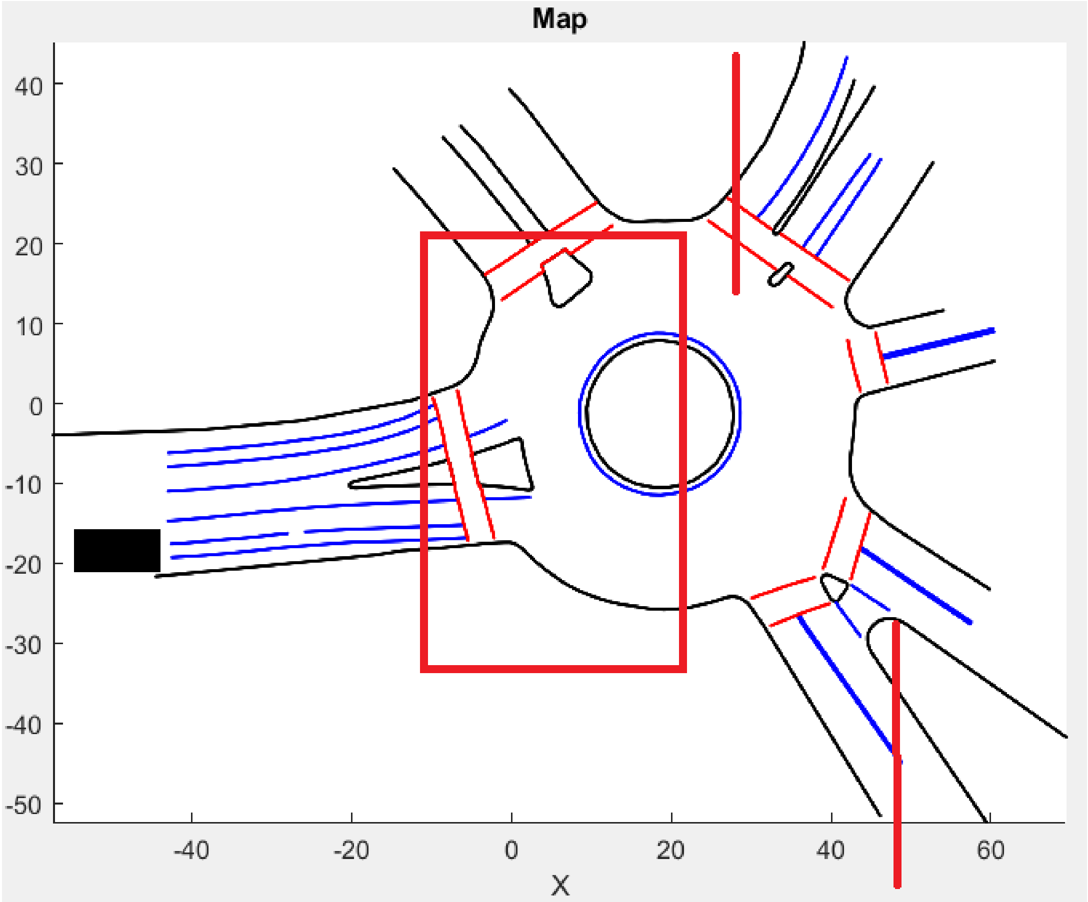
Vehicle Interaction Learning

Xiaosong Jia

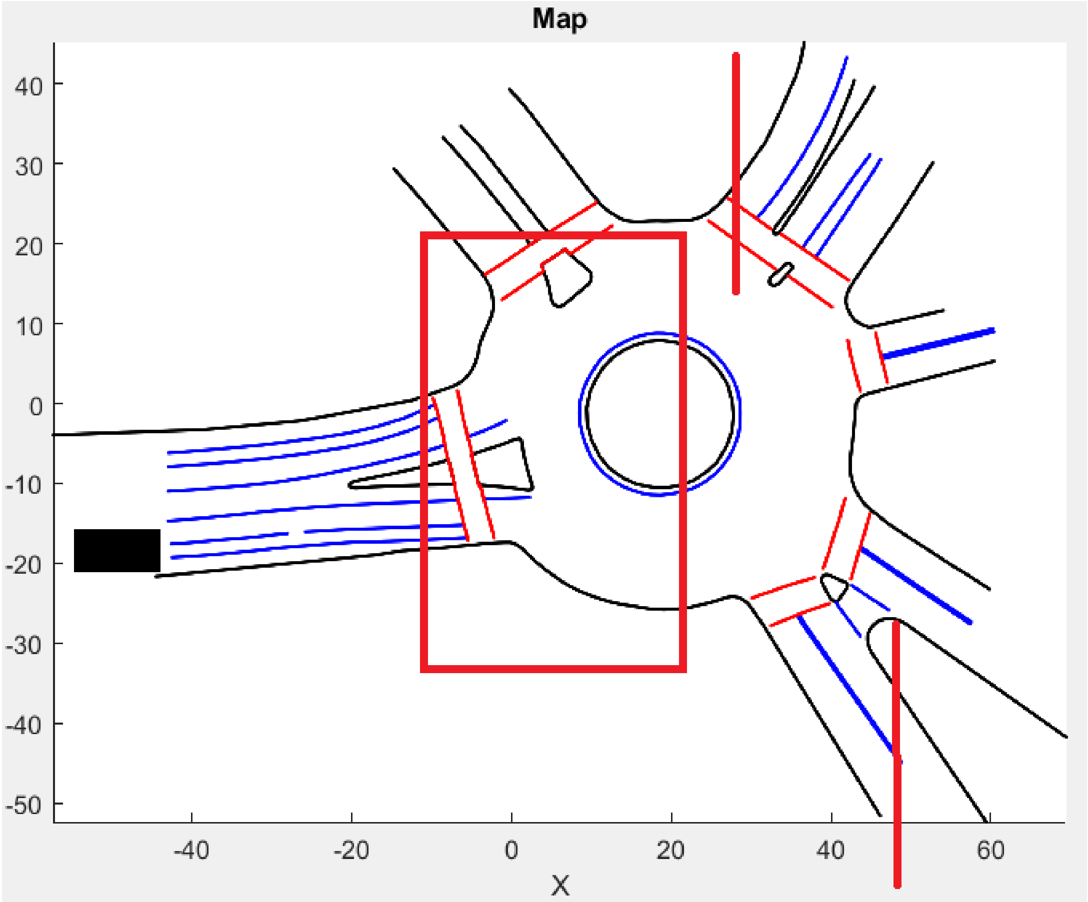
Generate Graph Data

1. For each entrance, generate a possible-neighbor-vehicle set.
2. Clip their appearance frames to make sure they are close enough.
3. Find all vehicles enter from this entrance and sort them by time order. (Main Vehicle)
4. For each Main Vehicle, at each time step, find their N nearest neighbors in the possible-neighbor-vehicle set.



Generate Graph Data

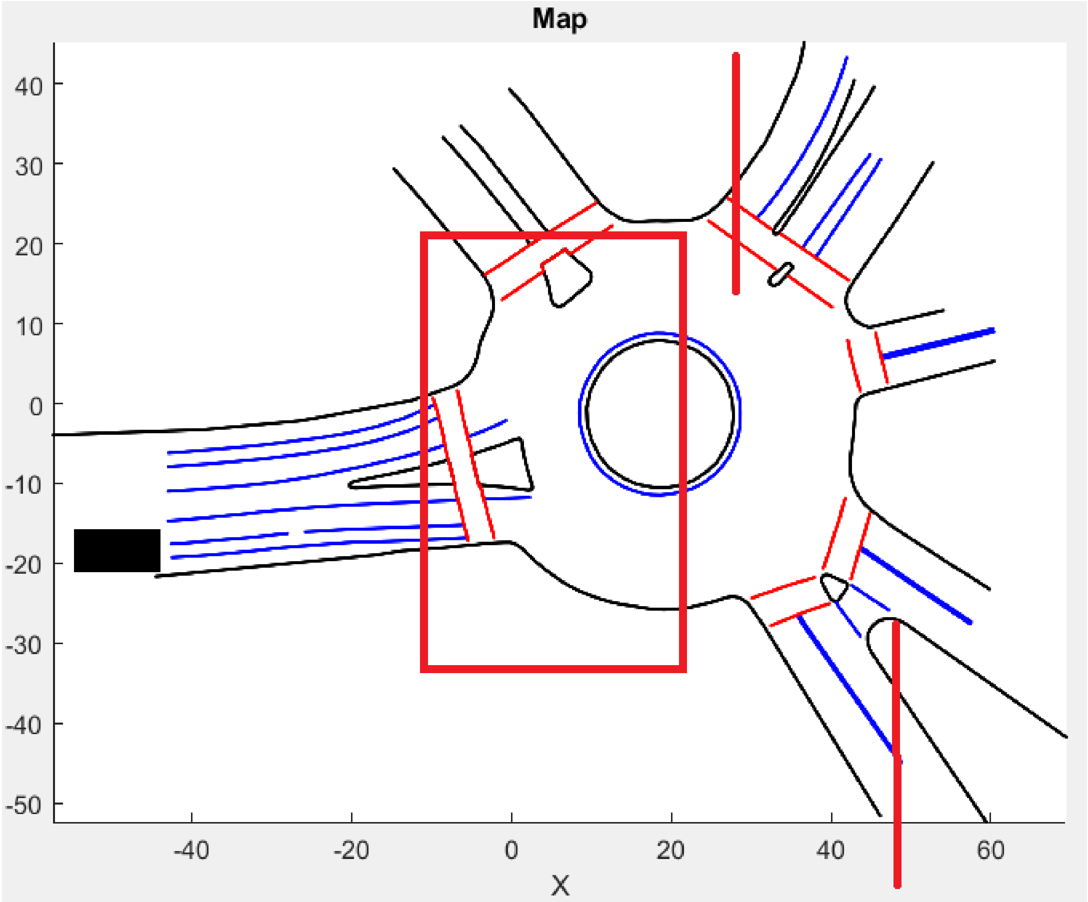
5. For all the contiguous frames with same neighbors and length > T, we keep it as a spatial-temporal graph. It is a N*T*D matrix.
6. For each timestep, we calculate the TTC of all pairs of vehicles, which forms a T*N*N matrix.
7. To generate a T*N*N label matrix: TTC<3s -> Positive, TTC > 8s -> Negative, otherwise -> Unknown



Generate Graph Data

Additional Processing

1. Delete all the duplicate samples.
2. Divide training-test set by folders.
3. FPS = 10, Max_number_of_neighbors=10
4. Features: coordinate, velocity



Graph data statistics

MaxLength	#Train Sample	# Test Sample	Average Length	Max Length
2s	43614	5436	3.60s	46.2s
4s	11037	1351	6.16s	46.2s
8s	1603	180	12.4s	46.2s

Max Length	#Object 2	#Object 3	#Object 4	#Object 5	#Object 6	#Object7	#Object 8	#Object 9
2s	17556	12984	6894	3449	1704	716	240	71
4s	6694	2450	1107	475	226	67	2	1
8s	1105	313	121	51	12	1	0	0

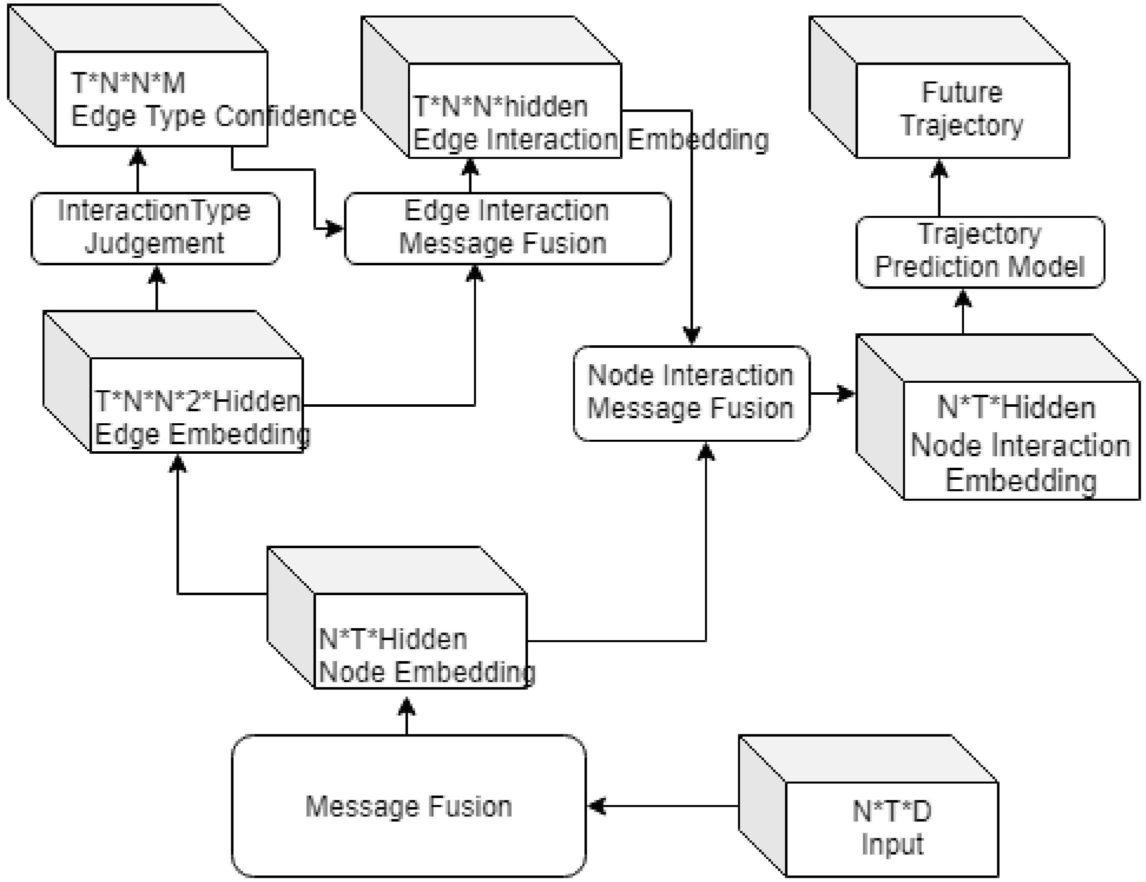
Multi-agent, Multi-type Interaction model

- Input: N objects, T frames, D features (coordinate and velocity)

-> N*T*D matrix

- Output: T frames, N*N 'pairs of objects, M confidences of each type of interactions -> T*N*N*M matrix

- Graph: Node- a vehicle at a timestep, Edge – the interaction type of the two corresponding vehicles (nodes)



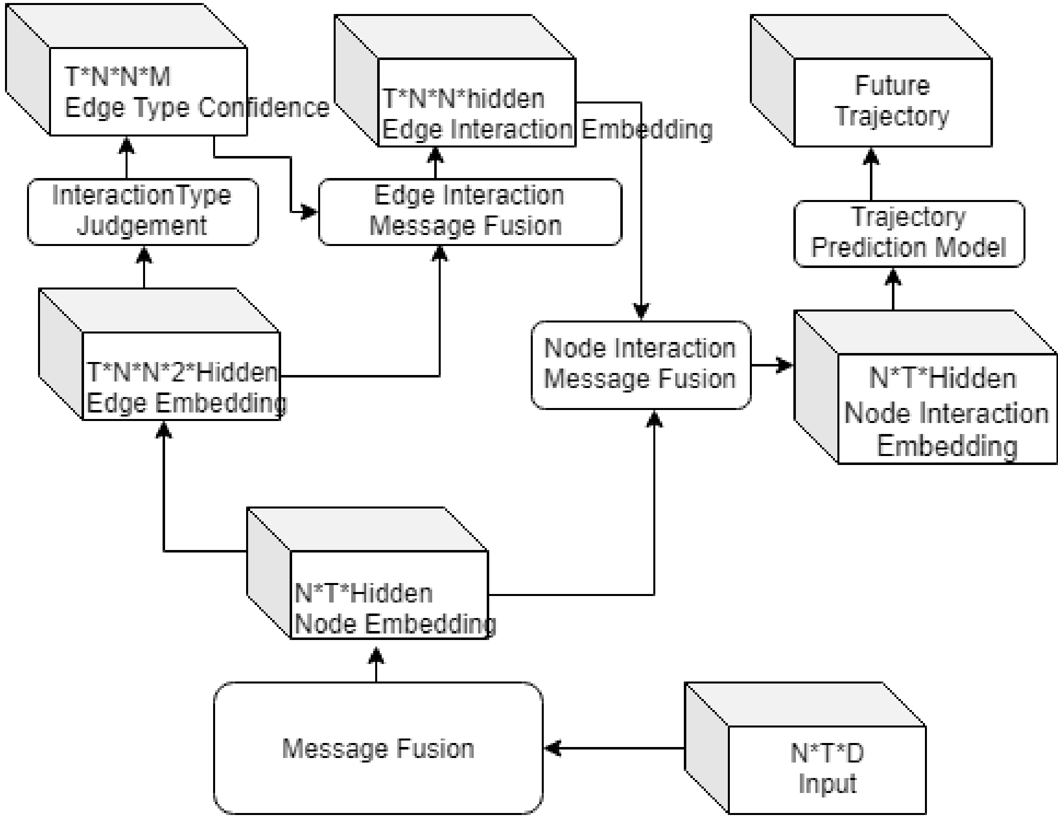
Spatial-Temporal Message Fusion

- At each time-step, let each vehicle ‘know’ other vehicles’ information and information about its past (no future information to avoid data leakage)
- Right now, fully connected graph.
- Fuse spatial and temporal information alternatively by transformers (spatial-temporal transformer)
- Transformer can fuse information from others with attention mechanism

```
## number_of_objects, seq_len, n_embed
x = x.transpose(0,1)  ## seq_len, number_of_objects,  n_embed
for i in range(len(self.spatial_module)):
    x, attn_weight = self.temporal_module[i](x.transpose(0,1),  position_enc=pos_enc, non_pad_mask=None, slf_attn_mask=slf_att_mask)  ##
    number_of_objects, seq_len, n_embed
    x, attn_weight = self.spatial_module[i](x.transpose(0,1),  position_enc=None, non_pad_mask=None, slf_attn_mask=None)##seq_len,
    number_of_objects, n_embed
```

Interaction Message Fusion

- Edge Embedding: Concatenate embeddings of two corresponding nodes -> T∗N∗N∗2hidden
- Interaction Type Judgement: A set of linear layers with Edge Embedding -> T∗N∗N∗M (M is the number of interaction types.)
- Interaction Embedding: M sets of linear layers with Edge Embedding -> M∗T∗N∗N∗hidden



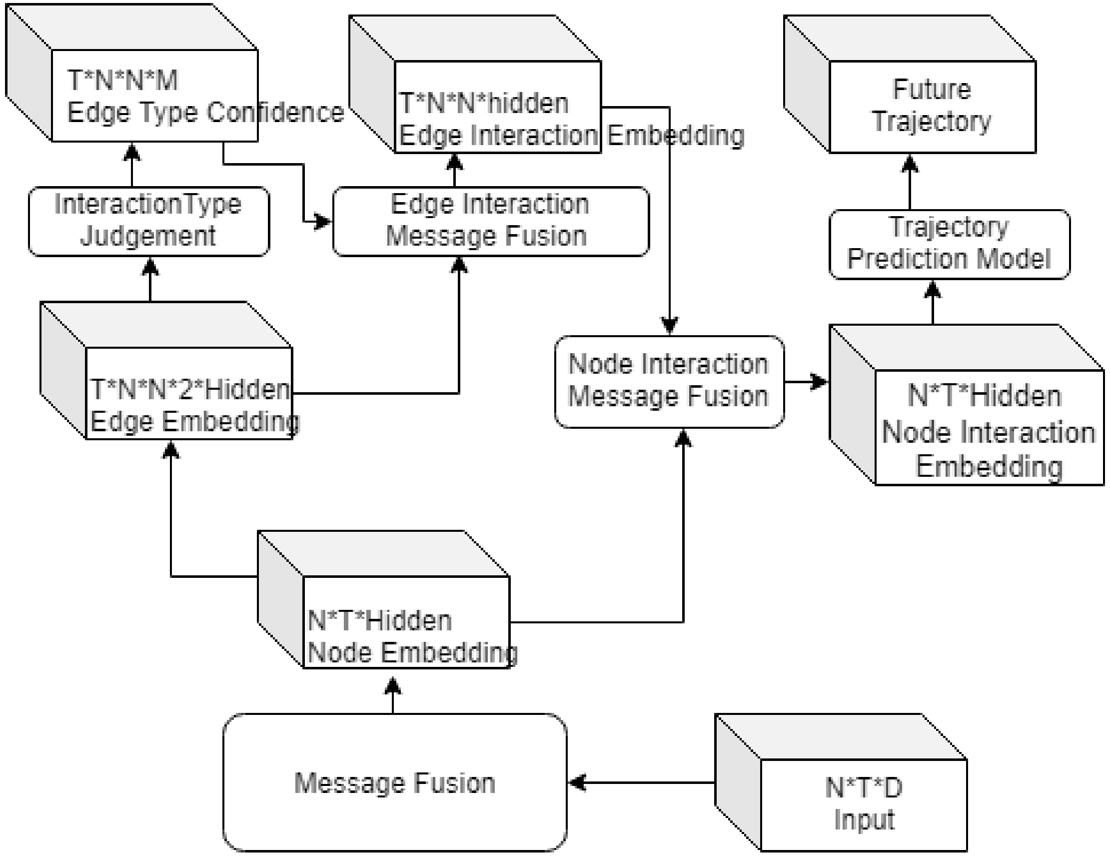
Interaction Message Fusion

- Interaction Edge Embedding: weighted sum of M Interaction Embedding -> M∗T∗N∗N∗hidden -> T∗N∗N∗hidden

Hard (one hot weight) vs Softmax weights?

- Interaction Node Embedding: fuse original Node Embedding and Interaction Edge Embeddings of all its edges.

Should no-interaction edge has its own embedding?



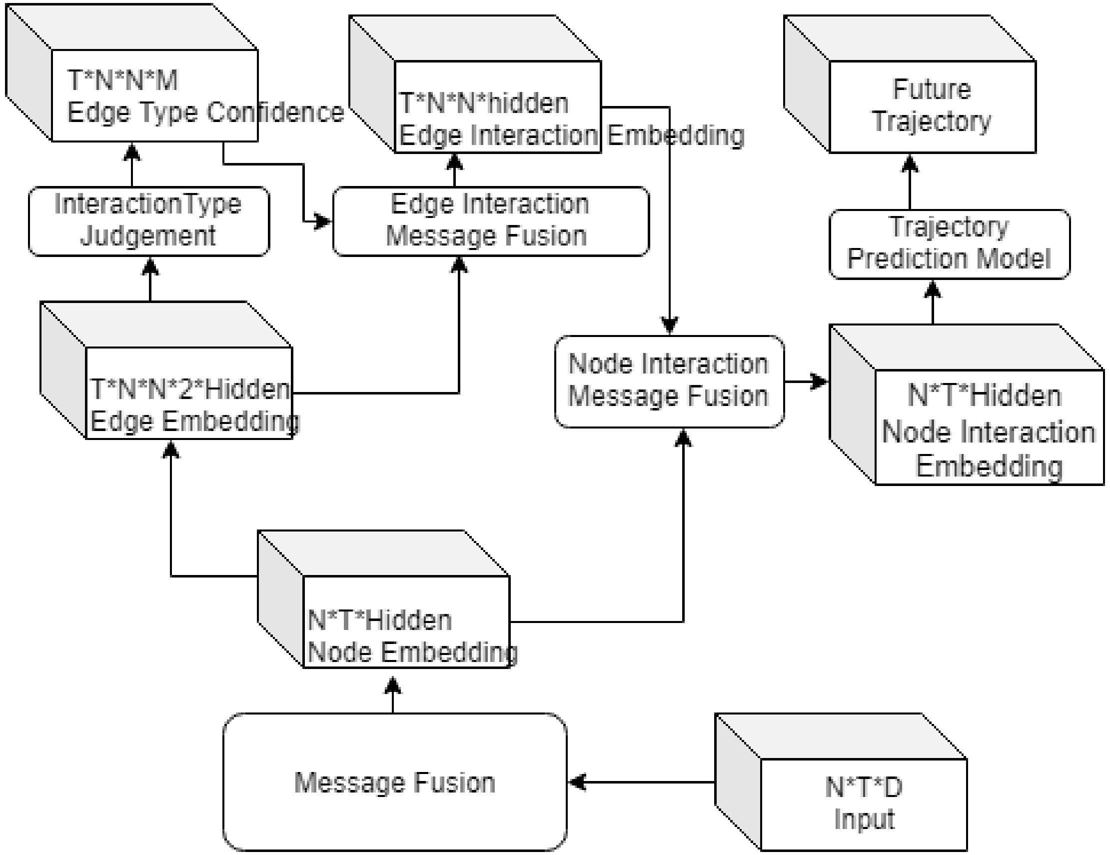
Interaction Message Fusion

- Interaction Node Embedding:

Soft-no-interaction: $\boldsymbol{v}_{\boldsymbol{i}}' = w_1 \ * \boldsymbol{v}_{\boldsymbol{i}} + w_2 \ * \sum_{j \in N(i)} v_{ij}$

connect_ratio: cr

Hard-no-interaction: $\boldsymbol{v}_{\boldsymbol{i}}' = (w_1 + 1 - cr) * \boldsymbol{v}_{\boldsymbol{i}} + (w_2 + cr) \ * \sum_{j \in N(i)} v_{ij}$



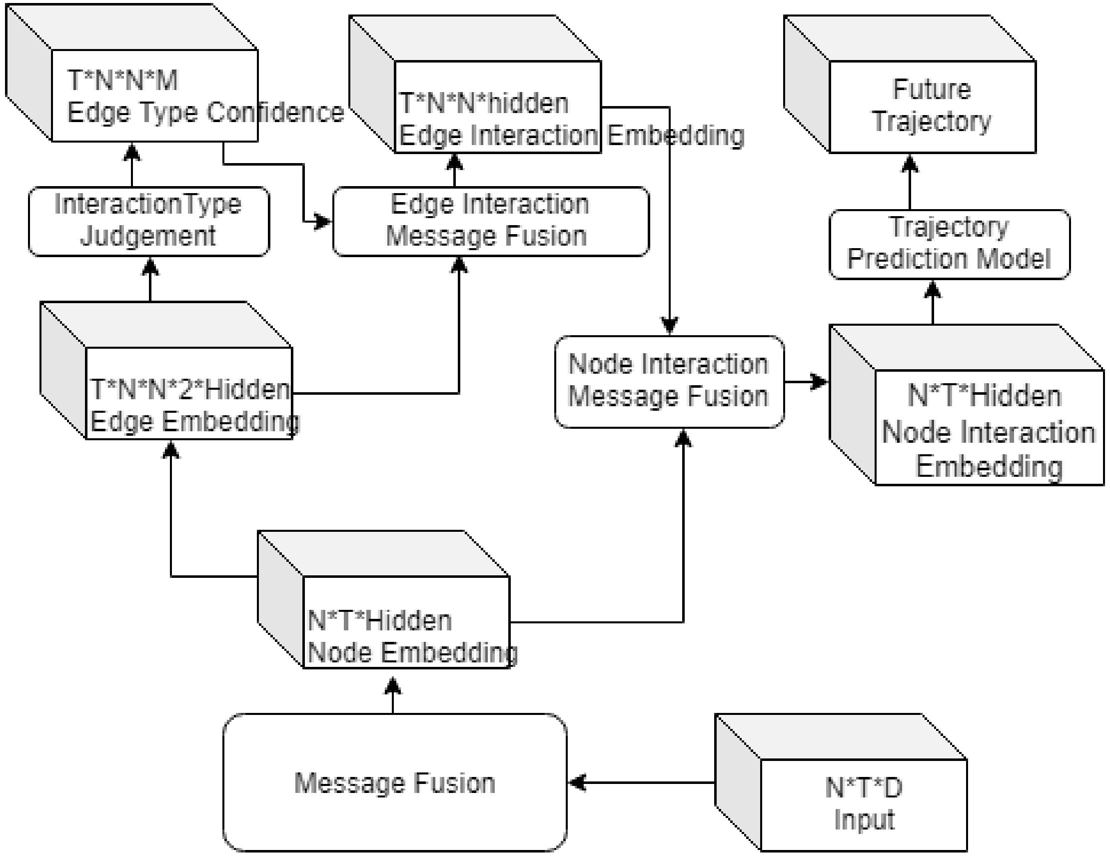
Trajectory Prediction

- Another spatial-temporal transformer: spread interaction information

-> Visualize Attention Matrix: at each timestep, each vehicle's attention point

- LSTM: at each time step, predict next t frames relative movement.

  T*N*t*2 matrix



Loss Function

- Trajectory Prediction Loss: predict next t step's relative movement

$$L_{traj} = \left|\left|pred - gt\right|\right|^2$$

- Prior Distribution (KL Loss)

Label Information: Positive:0.0252, Negative:0.9168, Unknown:0.05799

3 types of edges -> Prior: uniform for unknown: [0.9458, 0.0397, 0.0145]

$$L_{prior} = \sum_k p_k log \frac{p_k}{p_k'}$$

- Supervised Edge Label (Cross Entropy Loss):

$$L_{sup} = -\sum_k y_k log p_k'$$

Loss Function

- Unsupervised Edge Entropy Loss: make model more sure about edge types

$$L_{ent} = -\sum_k p'_k log p'_k$$

- Edge Diverse Loss: make edge embeddings more distinguished

$$L_{diverse} = -\sum_{i,j,k_1,k_2} \left| \frac{v_{ij,k_1} v_{ij,k_2}}{\left\|v_{ij,k_1}\right\| \, ||v_{ij,k_1}||} \right|$$

Some Results

- Supervised edge + Predict Trajectory + Prior Distribution

```
*** Epoch: [10][1351/1351], pred_losses 1.42214e-06, supervised_edge_losses
0.217, kl_losses 0.026, unsupervised_edge_entropy_losses 0.000,
edge_fc_diverse_losses 0.000, losses 0.081, supervised_edge_accs 0.935,
supervised_edge_precisions 0.185, supervised_edge_recalls 0.289 Edge0_num
0.8676789837388353, Edge1_num 0.12866615722138683, Edge2_num
0.0036548590397778 16
```

```
*** Epoch: [27][1351/1351], pred_losses 4.39581e-07, supervised_edge_losses
0.217, kl_losses 0.032, unsupervised_edge_entropy_losses 0.000,
edge_fc_diverse_losses 0.000, losses 0.083, supervised_edge_accs 0.912,
supervised_edge_precisions 0.172, supervised_edge_recalls 0.338 Edge0_num
0.8438336578992778, Edge1_num 0.15319512575903238, Edge2_num
0.002971216341689879
```

New Idea

- Pure transformer for prediction
- Check: at each layer, what each vehicle is pay attention to?
- Dense-transformer: each vehicle always observe low level and high level features of other vehicles

- Only supervised task?