Intellisys - Project Report

4th July 2021

Progresses

1. Debug pipelines for data extraction and parsing:
   - updated function to calculate edges weights, see gif
   - imported edges weights and intention into data object
   - renamed the nodes that define the edges in $edges_{index}$. (the numbering is always 0-9, even if there are 300 vehicles in the full simulation, because per time step there are max 10)

Figure 1: Data object

```
==================== 165 ====================

data_x              [[0.0, 14.44, 3.0], [90.0, 0.0, 1.0], [270.0, ...
data_pos            [[108.0, 163.77], [85.4, 92.0], [34.26, 104.8]...
data_edges          [[18, 16, 21, 13, 13, 22, 16, 16, 20, 18, 23, ...
data_edges_renamed  [[2, 1, 4, 0, 0, 5, 1, 1, 3, 2, 6, 5, 1, 0, 0,...
data_y              [[108.0, 199.46, 0.0], [85.4, 92.0, 90.0], [85...
data_edges_attr     [1.5, 3.5, 2.6, 1.6, 1.6, 2.0, 2.0, 1.5, 3.5, ...
training_row                                                    False
Name: 165, dtype: object
padded_data_x.shape=(10, 3)
padded_data_pos.shape=(10, 2)
padded_data_y.shape=(10, 3)
x=tensor([[  0.0000,  14.4400,   3.0000],
        [ 90.0000,   0.0000,   1.0000],
        [270.0000,  18.1500,   1.0000],
        [ 90.0000,   0.0000,   0.0000],
        [  0.0000,   7.5700,   3.0000],
        [270.0000,  15.6100,   1.0000],
        [  0.0000,   0.0000,   3.0000],
        [180.0000,  12.3100,   3.0000],
        [  0.0000,   0.0000,   0.0000],
        [  0.0000,   0.0000,   0.0000]])
pos=tensor([[108.0000, 163.7700],
        [ 85.4000,  92.0000],
        [ 34.2600, 104.8000],
        [ 85.4000,  98.4000],
        [108.0000, 127.1600],
        [ 61.7400, 104.8000],
        [108.0000,  85.4000],
        [ 92.0000, 150.4900],
        [  0.0000,   0.0000],
        [  0.0000,   0.0000]])
edge_index=tensor([[2, 1, 4, 0, 0, 5, 1, 1, 3, 2, 6, 5, 1, 0, 0, 2, 3, 3, 3, 0, 1, 0, 2, 4,
         2, 4, 0, 1],
        [5, 4, 5, 7, 6, 7, 7, 2, 4, 3, 7, 6, 6, 2, 5, 4, 7, 6, 5, 1, 5, 3, 7, 7,
         6, 6, 4, 3]])
edge_attr=tensor([1.5000, 3.5000, 2.6000, 1.6000, 1.6000, 2.0000, 2.0000, 1.5000, 3.5000,
        1.5000, 2.0000, 2.6000, 6.0000, 1.5000, 1.6000, 1.5000, 2.0000, 6.0000,
        2.6000, 1.6000, 2.6000, 1.6000, 1.5000, 2.0000, 1.5000, 3.5000, 1.6000,
        6.0000])
y=tensor([[108.0000, 199.4600,   0.0000],
        [ 85.4000,  92.0000,  90.0000],
        [ 85.4000,  98.4000,  90.0000],
        [108.0000, 149.3000,   0.0000],
        [ 25.4000, 104.8000, 270.0000],
        [108.0000,  85.4000,   0.0000],
        [ 92.0000, 125.1600, 180.0000],
        [  0.0000,   0.0000,   0.0000],
        [  0.0000,   0.0000,   0.0000],
        [  0.0000,   0.0000,   0.0000]])
```

2. Progress with the Pipeline for Training
   - Including Data Loader part
   - Updated and fixed the shape of inputs and output matrices. We use a matrix of fixed size Mx3 for the features and Mx2 for the positions, to predict a fixed size matrix Mx3, where M is the max number of vehicles at each timeframe in the simulation. If only 3 vehicles are present in the input or in the target, the remaining rows are all zeros.
     The non-zero input is the current state of the crossing described by:
     - $data_{pos}$ = positions (x, y) of all the vehicles in t
     - $data_x$ = features (yaw, speed, intention) of all the vehicles in t
     - $edges_{indexes}$ = list of outbound and inbound nodes
     - $edges_{attr}$ = weights of these edges, calculated as above

     The non-zero Output is the state at time t + δ_t, specifically:
     - the positions (x, y)

– the orientation (yaw)

of the vehicles in state t that are still present in state t + δ_t. We cannot predict features for vehicles that are non-existent in time step t. For example between time step 15 and time step 15+2 vehicle "0" exits the simulation and we don't look at other vehicles present in the time step 17 but not in 15:

Figure 2: example of target with less vehicles than the input

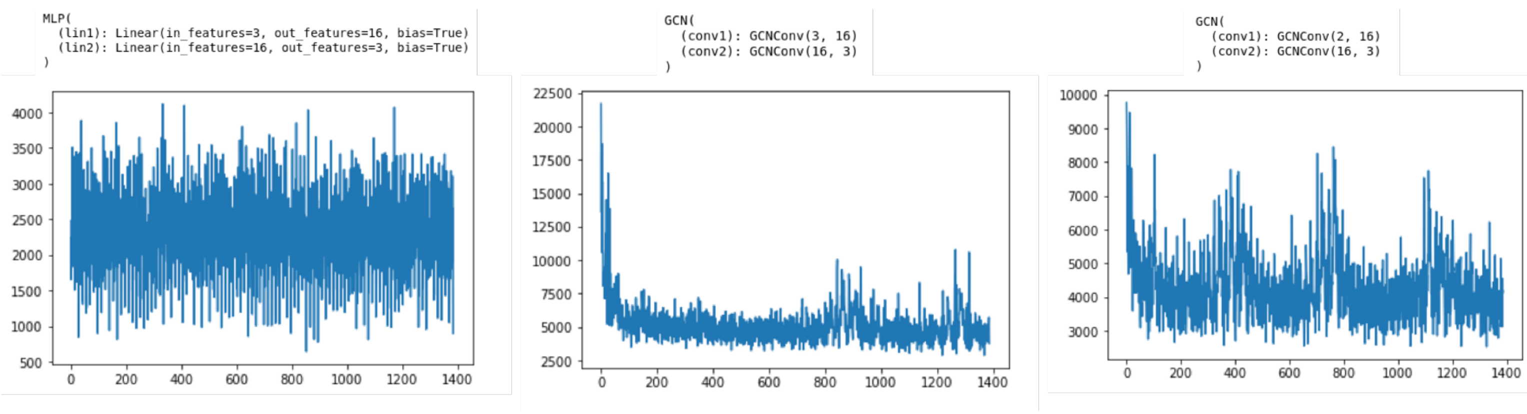
```
data_x                  [[90.0, 17.88, 2.0], [180.0, 10.69, 3.0], [90....
data_pos                   [[169.71, 95.2], [92.0, 121.97], [11.01, 98.4]]
data_edges                                          [[0, 0, 1], [1, 2, 2]]
data_edges_renamed                                  [[0, 0, 1], [1, 2, 2]]
data_y                    [[90.26, 109.2, 201.18], [24.81, 98.4, 90.0]]
data_edges_attr                                            [1.4, 1.1, 1.1]

Name: 15, dtype: object
```

- Current settings:

```
loss: MSELoss,
len(train loader.dataset)=134
len(val loader.dataset)=52
len(test loader.dataset)=6
batch size = 10,
epochs = 10
```

- Following PyTorch notebooks intro to GCN, we tried to have some quick training with two different basic architectures: GCN and MLP. First impression is that the GCN trained only with $data_x$ and $edges_{indexes}$ or only with $data_{pos}$ and

Figure 3: training losses of MLP and simple GCN



$edges_{indexes}$ is learning, whereas the MLP is stuck. We are using just a very limited amount of data, so this is not yet close to a final result.

Next Steps

– Structure properly the code for training

– Including $data_{pos}$ and $edges_{attr}$ as training input

– Play with architectures and settings in general, e.g. other loss

Questions

– Which loss should we use? Is the MCE good to start?

– Can we take this approach with padding the input and the output or is there a better way?

– Should we mask rows of zeros when computing the loss? How ? (we don't want just to mask the single element == 0, since it might be a legit value in some cases)