DLAD: Multi-Agent Vehicle Control
August 2021

Nachbar, Luise
s0017

Taormina, Arianna
s0014

1 Introduction and Motivation

The scope of the project is to investigate how a neural network can control multiple vehicles at one specific intersection, not using images or videos, but information that is light-weight and easy to retrieve, such as position, orientation and speed of the vehicles. This numerical information is transmitted to a central unit, whose role is to ensure the optimal passing of the cars over the intersection. The central unit performs all the analysis and controls the future positions of each vehicle, sparing on computational resources, since there is no need for one GPU per car. This central unit gains a complete overview of the situation at the crossing and therefore can help to avoid traffic jams as well as collisions between vehicles. At the same time, it can minimize the waiting time of each vehicle, since either it could optimally control the traffic lights or the traffic lights could even be omitted completely.

This task can be solved either as a classification task, predicting the next position of each vehicle out of a finite set of possibilities, or as a regression task, i.e. computing the exact values of the new positions. The second approach is the one investigated more thoroughly, where the predicted target consists of the location and orientation of each car two seconds later (this delta time value is an arbitrary, but fixed choice). At the end of the project, further attempt to perform classification is started, but is not yet completed.

The report is structured as follows: in Chapter 2 a quick overview of related work and publications is given, in Chapter 3 is described the basic theoretical background to implement the methods used to solve the task. Chapter 4 provides a step-by-step description of the entire process. In Chapter 5 the retrieved results of the experiment are presented and finally, in Chapter 6 a conclusion is drawn, and possible future work is proposed.

2 Related work

In the field of intelligent transportation systems, a lot of focus has been on the application of Reinforcement Learning (RL) to solve the problem of defining an autonomous driving policy (Wang, Wang, and Cui, 2021) or to control and coordinate traffic lights (Chen et al., 2020), (Wei et al., 2019). In most cases highly complex visual data is used, and the models fail to generalize well. Similarly, in the field of Optic Transport Networks (OTN), Deep Reinforcement Learning (DRL) algorithms have been used to solve routing optimization problems for agents, considering the overall traffic demand in a network. Again, exhibiting poor results when applied to more complex problems, such as flow-based routing (Suárez, 2019).

Some novel approaches try to overcome these limitations in generalization by combining Graph Convolutional Networks (GCN) with DRL. The idea is that graphs capture the interplay between agents by their relation representations (Jiang et al., 2020), therefore achieving a much better combinatorial generalization than other existing methods (Almasan et al., 2020).

While recognizing the potential of combining GCN with RL, this project, due to time constraints, focuses on using exclusively GCN and compare the achieved results with a basic Multi Layer Perceptron Network.

3 Methods

In this chapter, the different networks used to perform the task are presented. This includes Multi Layer Perceptron as well as Graph Convolutional Neural Networks.

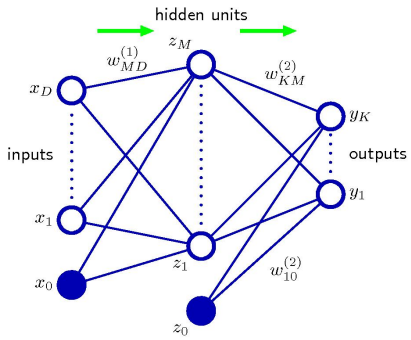
3.1 Multi Layer Perceptron

The Multi Layer Perceptron (MLP) is a very general neural net architecture that can in principle approximate a wide variety of functions and does not require strong assumption on the "shape" of the input. MLP consists of one input and one output layers, plus one or more hidden fully connected layers. It is also referred to as "Vanilla Neural Network", for its basic structure, or "Feed Forward Neural Network" (FFNN), being acyclic, in contrast to e.g. more complex Recurrent Neural Networks (RNN) (Gardner and Dorling, 1998).

In Fig 1, taken from Bishop (1995), an example of a simple MLP is given:

- Input data $x_1..x_D$, and its bias x_0
- One hidden layer with $z_1..z_M$ hidden units, and its bias z_0
- Outputs $y_1..y_K$
- Two layers of adaptable parameters (weights) denoted by $w = w^{(1)}_{MD}, w^{(2)}_{KM}$

Figure 1: Structure of simple MLP.



3.2 Graph Convolutional Neural Network

Graph Convolutional Network (GCNs) and its variants are established as a standard when it comes to learning a graph structure. GCNs are a variant of Convolutional Neural Networks (CNNs), that can work on graphs (Wu et al., 2019).

As explained in pytorch documentation, given a graph G = (V, E), where V are the vertices and E are the edges, a GCN takes as input (Paszke et al., 2017):

- An input feature matrix X of dimension $N \times F^0$ where:
  - N is the number of nodes
  - F^0 is the number of input features for each node
- A N × N matrix representation of the graph structure (e.g. the adjacency matrix A of G)

A hidden layer in the GCN can be written as

$$H^i = f(H^{i-1}, A))$$

where:

- $H^0 = X$
- f is a propagation rule

Each layer H^i is a $N \times F^i$ feature matrix and each row is a node feature representation. The propagation rule f aggregates the feature representation of each node to form the next layer features. By changing the propagation rule, one gets different variants of GCN (see Appendix A.1).

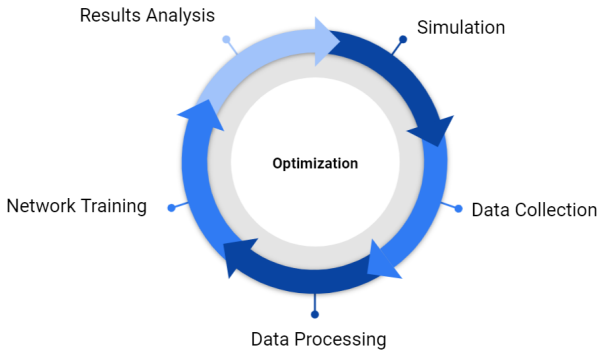
By representing the data as graphs, the structural information can be encoded to model the relations among entities, and furnish more promising insights about the underlying data. For example, in a transportation network, nodes are often the sensors and edges represent the spatial proximity among sensors (Paszke et al., 2017).

The core idea behind the use of GCN in the context of that task is that, through message passing, the features of each vehicle would gain aggregated information from all the other connected cars.

4 Process

In this chapter, the process followed to solve the task, motivated in Chapter 1, is described. This process consists of multiple steps, beginning with the simulation, where the setting of the crossing is defined. This crossing is used in the second step to collect data out of the simulation. The raw data is then processed and cleaned in the third step, so that a defined network can be trained with it in the fourth step. And in the last step, the obtained results are analysed. As Fig 2 shows, this process is iterative, as the results had to be optimized repeatedly in one or more steps. In the following, each of these steps are described in detail.

Figure 2: Cycle of the iterative process that is followed to solve the task.



4.1 Simulation

In order to get the data needed for the task, the traffic flow simulation SUMO is used. The SUMO simulation can be used as a stand alone or in combination with CARLA. In the course of this project, both variants are tested. The advantage of using the combination of both software products is that it allows using more complex towns defined in CARLA, within the SUMO simulation. However, since SUMO without CARLA gives more control over how to define an intersection, this second approach is the one finally chosen (Lopez et al., 2018).

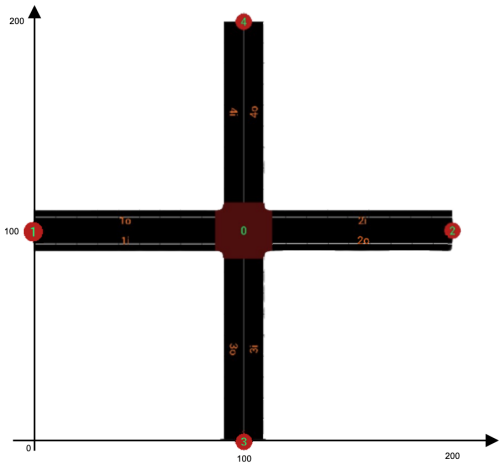
In the further process of this work, a simple crossing is used, where five nodes span out the intersection 100 meters in each direction and which are marked in the Fig 3 with red dots. Each street has two lanes, one that goes to the intersection and one that goes away from the intersection. It is a controlled intersection; there are four traffic lights that switches state in deterministic way, repeating the same sequence of red phase (78 seconds), green phase (33 seconds) and yellow phase (6 seconds).

Vehicles drive on the crossing as a flow, meaning that the vehicles join and leave the intersection continuously. The vehicles are spawned randomly with a given probability each second. To get different scenarios, different probabilities for every opportunity the cars have to pass the intersection are defined. The cars drive perfectly within the traffic, i.e. there are no collisions, nor does a car stop unpredictably in the middle of the road. For easiness, an upper threshold on the maximum number of vehicle present at once in the simulation is set to ten.

4.2 Data Collection

A snippet of the raw data collected directly out of the simulation is displayed in Appendix A.2, Fig 9. A list of the

Figure 3: Representation of simple crossing that is built with SUMO and is used for the task.



selected information out of the simulation is given below (Lopez et al., 2018):

- x, y: The position of the vehicle in the (100, 100) centred intersection in meters at timestep t.
- *speed*: The speed of the vehicle in m/s at timestep t.
- *yaw*: The orientation of the vehicle in degrees at timestep t.

During later experiments, more information was collected out of the simulation (Lopez et al., 2018):

- *traffic light status*: The status of the next upcoming traffic light at timestep t. It could take the values green, yellow, red or none, if there is no upcoming traffic light in front of the car.
- *duration until traffic light changes to the next iteration*: The time until the next upcoming traffic light changes its iteration in seconds at timestep t (see Appendix A.2, Table 1).
- *acceleration*: The acceleration of the vehicle in m/s^2 at timestep t .

Some of these features served either as the basis for the transformations described in Section 4.3 or they are used as raw input to the network.

4.3 Data Processing

In this step, the raw data is cleaned and processed into the right shape, so that a network can be trained with it. There are mandatory steps, as well as steps that can optionally be included into the pipeline.

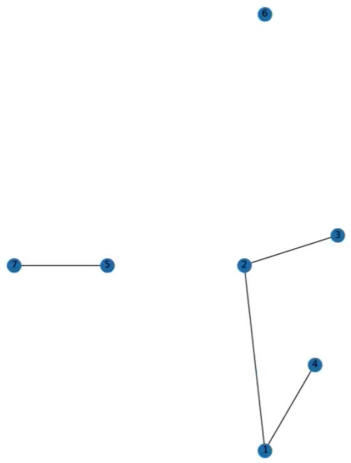
Mandatory: The mandatory steps include transforming the data into a graph structure, handling discontinuity, and adding the *intention* column information.

1. Transforming Data to Graphs:
   The data is transformed into graphs that could be used for training. At each timestep, a graph is used to represent the situation of the intersection. In each graph, a vehicle is a node, and it is represented by a set of features. The nodes are connected by edges. As an example, Appendix A.2, Fig 10 shows a situation of three vehicles and the corresponding graph structure.

   Edges can be created, maintained and weighted in many ways. Two options are explored:

   (a) First two thresholds are set: A lower one, i.e. 20 or 30, for the distance for the creation of an edge, and an upper one, i.e. 50 or 70, for the maintenance of that edge. This means that two cars won't be connected by an edge before being close enough to each other, and they will lose the connection edge the moment they would move far apart. This approach often returns graphs that are disconnected (see Fig 4), which can be more complex to handle with the GCN, and it takes more computation than the second option, which ultimately is chosen.

Figure 4: Example of disconnected graphs with edge creation and maintenance using upper and lower thresholds.



   (b) The second option consists of fully connected graphs with weighted edges (see Fig 5). In the final version, the weights for the edges depend on the position of the vehicles with respect to the intersection, so that nodes in the centre of the intersection have more weight, and on the mutual distance between the two cars, so that the edge of two nodes that are closer to each other weight more. The weight between two nodes is calculated as stated in equation 1:

$$weight = \frac{weight1}{max(dist1, dist2)} + \frac{weight2}{dist(veh1, veh2)}. \quad (1)$$

   With:

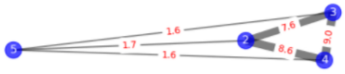
$$dist1 = dist(veh1, centre). \quad (2)$$

$$dist2 = dist(veh2, centre). \quad (3)$$

   Where:

- $dist()$ calculates the Euclidean distance between two input points
- $veh1$ stands for the coordinates of the first vehicle and $veh2$ for the coordinates of the second vehicle
- $centre$ stands for the coordinates of the centre of the crossing
- $weight1$ and $weight2$ are corrective factors, where more weight is given to the first addend

Figure 5: Example of fully connected graph with weighted edges.



2. Handling Discontinuity:
   One challenge is how to handle discontinuity in the graph. By "discontinuity" the situation in which between the time of the input and the time of the output some vehicles left the simulation are meant. This is solved by considering only "consistent" rows of data, meaning the ones in which the nodes in input and output are the same. This step reduces the amount of data to about 75 percent of the original data.
3. Adding *intention* Column:
   In addition to the columns that come directly from the simulation and that are defined in Section 4.2, the *intention* column is added:

   - *intention*: Final direction that a vehicle wants to take at timestep t.

   The *intention* is encoded in two ways, so that there are two options to represent the intention of a car. The first one is considering the right turn, left turn, u turn and straight drives. And the other is considering the cardinal directions, i.e. north, south, east and west.

Optional: In addition, experiments are conducted with some non-mandatory features. Some existing columns are transformed and used, trying to improve the network performances.

1. Normalization of Input Features:
   The standard deviations of the features differ, as can be seen in Appendix A.2 Table 2. Two different methods to normalize this difference are tested:

   (a) MinMax Normalization: This transformation scales every feature in a range of [0; 1]. The formula for the MinMax normalization is:

   $$x_i' = \frac{x_i - \min\{x_1, ..., x_D\}}{\max\{x_1, ..., x_D\} - \min\{x_1, ..., x_D\}}. \quad (4)$$

   It is a linear transformation, so the ratio of the data remains the same compared to the non-normalized data. The greatest value of the considered feature, is assigned the value 1 and the smallest value is assigned the value 0. All other values are transformed to a decimal number between 0 and 1 (Jain, Nandakumar, and Ross, 2005).

   (b) Standardization: The values of each column got transformed in a way, that they have a mean of 0 and a standard deviation of 1:

   $$x^{'} = \frac{x - \bar{x}}{s}. \quad (5)$$

   In contrast to the MinMax normalization, the data is not within a certain value range after the standardization transformation. In equation 5 $\bar{x}$ stands for the arithmetic mean and s stands for the sample standard deviation of the given dataset (Zheng and Casari, 2018).

   Applying normalization techniques could make it easier and faster for the network to train and learn interrelationships in the data (Sola and Sevilla, 1997). Note that these normalizations are done by column or by timestep.
2. Zero Centring the Coordinates:
   The coordinates for the input and the output position are zero centred, since originally the center of the SUMO crossing is located at (100, 100). The reason for this transformation is the hope, that the model would train better with a loss relatively bigger. For example, a wrong prediction at 10 instead of 0, could be seen as worse than 110 instead of 100:

   - x_{zc}, y_{zc}: The position of the vehicle in the (0, 0) centred intersection in meters at timestep t.
3. Generating Delta Output:
   For the same reason, why the coordinates are zero centered, the option is added to predict the delta position of the target instead of the absolute position of the target:

   - $\Delta x, \Delta y$: Difference in the position of the vehicle in meters between timestep t and t+2.
4. Transforming Yaw in Radians:
   The yaw is converted from degrees into radians, so that the loss of the yaw is on the same scale as the loss of the position:

   - *yaw[rad]*: The orientation of the vehicle in radians at timestep t.
5. Adding *still vehicle* Column:
   A proxy for the *traffic light status* column initially used is the column *still vehicle*:

   - *still vehicle*: Can take the value 0 if vehicle is not moving or 1 if the vehicle is moving during the timestep t and t+2.

6. Adding *duration until traffic light state changes* Column: Since different durations are defined for the red, green and yellow phase (see Appendix A.2, Table 1) and SUMO returns only the column *duration until traffic light changes to the next iteration*, transformations are performed to obtain this information:
   - *duration until traffic light state changes*: The time until the next upcoming traffic light changes its status in seconds at timestep t.

4.4 Network Training

During the whole optimization phase, more than 500 models are trained.

For efficiently training, early stopping is implemented so that the training would be blocked once the validation loss isn't improving enough after a certain number of epochs. The following shows a list of tried and tested parameters:

- Different architectures are built, this including different types of GCN layers (i.e. GCN, GraphConv, GATConv), different amount of layers (i.e. one up to ten layers), different neurons per layer (i.e. from 1 up to 512), different activation functions (i.e. leaky relu, tanh, relu) and different normalization layers (i.e. Batch Normalization, Graph Normalization).
- As optimizers Adam as well as Stochastic Gradient Descent (SGD) are tested.
- As loss functions L1 as well Mean Squared Error Loss (MSE) are used from the pytorch library. In addition, own domain specific losses are implemented, that take into consideration the physics of the problem of that task. These handmade losses, using L1 or MSE as basis, consider the current position of each vehicle and the maximum speed of vehicles in the simulation. The loss is increased 10 times if the prediction falls in an area which is not a plausible location, meaning outside the acceptance radius.
- As the learning rates the values 0.01, 0.001 and 0.0001 are used. Different learning rate schedulers, like "Cosine Annealing with Warm Restarts", "Reduce LR On Plateau" and "MultiStep LR", are adopted.
- For training the model, a number of datasets with different sizes are created, i.e. from 30 up to 150 000 input rows.
- As batch sizes the values 1, 16, 32, 64, 128, 512 and 1024 are evaluated.
- The networks are trained with and without edge weights.

4.5 Result Analysis

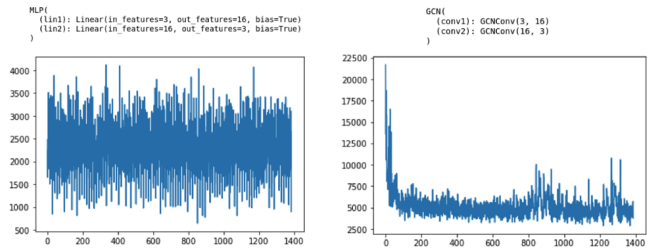
After each training, the results are collected and evaluated. First, the losses obtained from training and validation are inspected to check that the curves decrease as expected. Moreover, the intersection status of few frames, where some cars are moving during the timespan, are plotted including the target position, the prediction and the initial position, at different training epochs. This ensures that the loss curves correctly reflect the accuracy of what is actually predicted. In addition, the situations with the best losses and the with the worst losses are plotted to see how much they differ.

5 Experiments and Results

In this chapter, the most relevant experiments and results are presented.

First Check To prove that using GCN is a better option for tackling this task than a "Vanilla MLP", a quick check is performed at the very beginning of the project. For that a GCN and an equally simple MLP are trained, with the same, small dataset, low number of training epochs and low number of neurons for one hidden layer. Fig 6 shows the training loss curves of these two models.

Figure 6: Initial fast check: MLP vs GCN



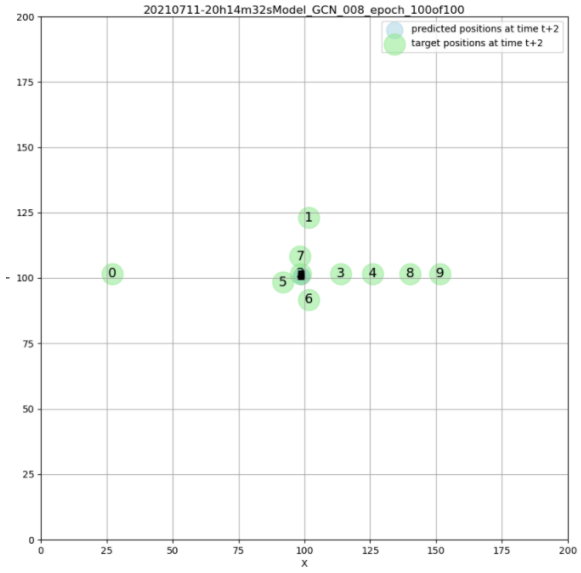
As can be seen, the loss is much lower with the MLP than with the GCN, but the trend is not decreasing, so it looks like the model is not learning anything. In contrast to the GCN, where the loss curve keeps going down continuously. Therefore, in the further proceedings and according to the task description, the focus is on optimizing the performance of the GCN. It needs to be noted that this quick check is performed with incomplete and not transformed data. A bias will be found in a later, more careful analysis.

GCN Optimization Initial GCN trainings are done with simple models, using only one hidden layer with 64 neurons, trained with few epochs. These results are not satisfactory, as can be seen in Fig 7, as all the vehicles tend to collapse into the centre, so they should be considered as initial exploration.

Attempts to improve the performances include making the models more complex, i.e. using more layers, a higher number of neurons, and adopting learning rate schedulers to break the validation losses plateau. Furthermore, additional experiments with the network parameters are performed, i.e. adopting domain specific loss, zero centring the positions, using or not using edges weights and predicting the delta values instead of the absolute values.

A closer look is given to the timesteps where the loss is the highest, trying to figure out why the networks are having a hard time to learn these cases. As mentioned in Section 4.4, different datasets are created and used to experiment with the trainings. It appears that in some cases higher errors occur in frames where the vehicles are all standing still, whereas in some other cases the worst performances are encountered in frames where all the vehicles are moving. Therefore, the distribution of stationary and moving vehicles over the datasets is analysed. Due to the initial probabilities with respect to the flow of vehicles, some datasets are

Figure 7: Initial GCN trainings: All the vehicles are predicted in the centre of the crossing.



more imbalanced than others when considering the number of rows containing all moving vehicles, all standing vehicles and both. This can be seen for example in Appendix A.2, Fig 11, where two of the most used datasets are compared. This is corrected by creating a balanced dataset from multiple data frames. However, it turns out that the dataset balancing does not influence the performance of the prediction.

The losses and the results obtained by the best performing GCN model are shown in Appendix A.3, Fig 12. It can be seen that the results have improved compared to the first trainings of using a GCN, but they are still not good enough, since the errors of the predictions are too high, especially in validation.

For this reason, the decision is taken to redo the bench marking against the MLP with the most recent datasets and improvements.

MLP Optimization As a first attempt, a very simple MLP is trained (see beginning of Chapter 5). For optimizing the MLP many more neurons, i.e. 256 instead of 16, a smaller batch size, i.e. 64 instead of 128, a learning rate scheduler and zero centred values for input and target are taken. Appendix A.3, Fig 13 shows the losses and results of the best MLP model. This model only predicts the x and the y values, but good results are also obtained when predicting additionally the yaw. Summarized, with a small amount of epochs, the simple MSE loss decreases considerably and the predictions are accurate in training, as well in validation. Even in the worst cases, the predictions are close to the target, and this without particular fine-tuning of the training parameters.

Last changes Regarding the questions that come up after the presentation, the focus is directed towards these two:

- How is it possible for the MLP to accurately predict where a vehicle should stop in a queue without receiving information from the other vehicles?
- Would a classification-head be helpful to guide the regression with the GCN?

To address the first question, the initial hypothesis is that the key information might come from the column *still vehicle*, that is used for training. This pose a problem because this column, built a posteriori, i.e. it is not available at timestep t, therefore it can not be used to control the vehicles in real time. Is it possible to use other information, available at timestep t, and still get good results with MLP?

Two attempts are made to replace the column *still vehicle*, by using features, available at timestep t:

1. *acceleration*
2. *traffic light status* and *duration until traffic light state changes*

In both cases new datasets are created, extracting for each the needed features from the SUMO simulation (see Section 4.2). Appendix A.3, Fig 14 shows the losses and examples of predictions of MLP using the *acceleration* information. It can be seen that the model that uses the *acceleration* performs better than the one using the *still vehicle* column, specifically in cases where a vehicle has to join a waiting queue. The performance of the model that uses the columns *traffic light status* and *duration until traffic light state changes* seems to be even more accurate, in the train, validation and test data, compared to the other two models (see Appendix A.3, Fig 15). However, it must be said that this model is only trained with a maximum number of five cars instead of the normally used upper bond of ten cars. Furthermore, it is trained with a significantly larger dataset, having the chance to generalize better to unseen data. All of these three simple MLP with one fully connected layer are able to predict satisfactory results, returning accurate predictions, including cases with the highest losses.

To address the second question, one more attempt is made to solve the regression task with a GCN, this time supported by a classifier that narrows the position of the prediction. For this, the area of the crossing [200m x 200m] is divided into sections, of size [20m x 20m], as can be seen in Figure 8.

This choice of classes is arbitrary, and is intended to keep a balance between eight classes, i.e. the lanes, which would be retrievable from the simulation itself and 40.000 classes, that emerge when using sections of size [1m x 1m]. The classification combined with the regression is structured as follows:

- First step: The initial positions are labelled and added as a column to the previous input matrix, in order to predict the classes of the target positions.
- Second step: This target classes are used as additional input to the regression with a GCN.

A good first approach is to check that the second step is actually improving the performances of the GCN. To do so the labels of the target positions are used as input to the GCN, as if the classifier is absolutely accurate. As can be

Figure 8: The crossing is divided in 100 classes. In the area of the red section, the target position can lie in.

seen from Appendix A.3, Fig 16, the perfect classifier seems indeed to improve the GCN performance. Comparing them directly with the results obtained with the MLP is quite difficult, since due to lack of time, the first step of the process is not completed, so there is not a clear indication on how accurate the classification can really be.

6 Conclusion and Future work

This chapter describes the conclusions and observations, and how the task can be further developed.

As main conclusion, the MLP models described above, adopting the *acceleration* column or the traffic light information, are the best performing ones. It also seems that the performance of the GCN improves when a classifier is used to identify the possible regions where the targets can be found. As more general takeaways, it can be seen that not necessarily more complex models (e.g. with more layers or neurons) are improving the performances. Moreover, it is a good practice to re-check on initial assumptions, to avoid biased results.

With more time at hand, the classification pipeline could be completed, and more thoroughly evaluated over bigger datasets, in order to have a fair performance comparison with the MLP. In the future, the best performing models could be tested online, trying to control vehicles in the SUMO simulation. In addition, another MLP model could be trained that uses both, the *acceleration* column and the traffic light information, trying to further improve the performance of the prediction. Moreover, one could train a model that generalize to a variety of crossings, meaning e.g. use more lanes or more routes the vehicles could drive. In addition, as mentioned in the related works in Chapter 2, the GCN could be used in combination with RL algorithms.

References

Almasan, P.; Suárez-Varela, J.; Badia-Sampera, A.; Rusek, K.; Barlet-Ros, P.; and Cabellos-Aparicio, A. 2020. Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case.

Bishop, C. 1995. *Neural networks for pattern recognition*. Oxford University Press, USA.

Chen, C.; Wei, H.; Xu, N.; Zheng, G.; Yang, M.; Xiong, Y.; Xu, K.; and Zhenhui. 2020. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *AAAI*.

Gardner, M., and Dorling, S. 1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment* 32(14):2627–2636.

Jain, A.; Nandakumar, K.; and Ross, A. 2005. Score normalization in multimodal biometric systems. *Pattern Recognition* 38(12):2270 – 2285.

Jiang, J.; Dun, C.; Huang, T.; and Lu, Z. 2020. Graph convolutional reinforcement learning.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *CoRR* abs/1609.02907.

Lopez, P. A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.-P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; and Wießner, E. 2018. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2020. Weisfeiler and leman go neural: Higher-order graph neural networks.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch.

Sola, J., and Sevilla, J. 1997. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science* 44(3):1464–1468.

Suárez, J. e. a. 2019. Routing in optical transport networks with deep reinforcement learning. *Journal of optical communications and networking*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph attention networks.

Wang, H.; Wang, Z.; and Cui, X. 2021. Multi-objective optimization based deep reinforcement learning for autonomous driving policy. *Journal of Physics: Conference Series* 1861(1):012097.

Wei, H.; Chen, C.; Zheng, G.; Wu, K.; Gayah, V.; Xu, K.; and Li, Z. 2019. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. KDD '19, 1290–1298. New York, NY, USA: Association for Computing Machinery.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In Chaudhuri, K., and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 6861–6871. PMLR.

Zheng, A., and Casari, A. 2018. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.".

A Appendix

A.1 Propagation Rules

In the GCN experiments, three different GCN layers are used.

GCNConv The paper Kipf and Welling (2016) proposes fast approximate spectral graph convolutions using a spectral propagation rule:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix.

It weighs neighbour, in the weighted sum, higher if they have a low-degree and lower if they have a high-degree. This may be useful when low-degree neighbours provide more useful information than high-degree neighbours.

GraphConv This is a generalization of GCNs, proposed in paper Morris et al. (2020), which can take higher-order graph structures at multiple scales into account.

$$\mathbf{x}'_i = \mathbf{\Theta}_1\mathbf{x}_i + \mathbf{\Theta}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j$$

where $e_{j,i}$ denotes the edge weight from source node j to target node i (default: 1)

GATConv Graph attention networks (GATs), proposed in Veličković et al. (2018), try to overcome the GCNConv shortcomings using the attention mechanism. The propagation rule is formally defined as:

$$\mathbf{x}'_i = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j$$

where the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{\Theta}\mathbf{x}_i \| \mathbf{\Theta}\mathbf{x}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}(i)\cup\{i\}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{\Theta}\mathbf{x}_i \| \mathbf{\Theta}\mathbf{x}_k\right]\right)\right)}$$

With these layers, it is possible to specify different weights to different nodes in a neighbourhood, without requiring any kind of costly matrix operation.

A.2 Other Figures

Figure 9: Raw data that was collected directly out of the SUMO simulation.

	time	vehID	X	Y	yaw	type	speed
0	0.0	0	92.00	194.90	180.00	DEFAULT_VEHTYPE	0.00
1	1.0	0	92.00	192.98	180.00	DEFAULT_VEHTYPE	1.92
2	2.0	0	92.00	189.34	180.00	DEFAULT_VEHTYPE	3.64
3	3.0	0	92.00	183.70	180.00	DEFAULT_VEHTYPE	5.64
4	3.0	1	92.00	194.90	180.00	DEFAULT_VEHTYPE	0.00
...	...	...	...	...	...	...	...
896	99.0	20	85.07	101.60	320.62	DEFAULT_VEHTYPE	3.29
897	99.0	21	108.00	77.55	0.00	DEFAULT_VEHTYPE	0.87
898	99.0	22	129.00	108.00	270.00	DEFAULT_VEHTYPE	13.13
899	99.0	23	108.00	46.41	0.00	DEFAULT_VEHTYPE	12.14
900	99.0	24	92.00	193.28	180.00	DEFAULT_VEHTYPE	1.62

Figure 10: Situation of a crossing transformed into a graph structure. The red car has edges to the two other cars.

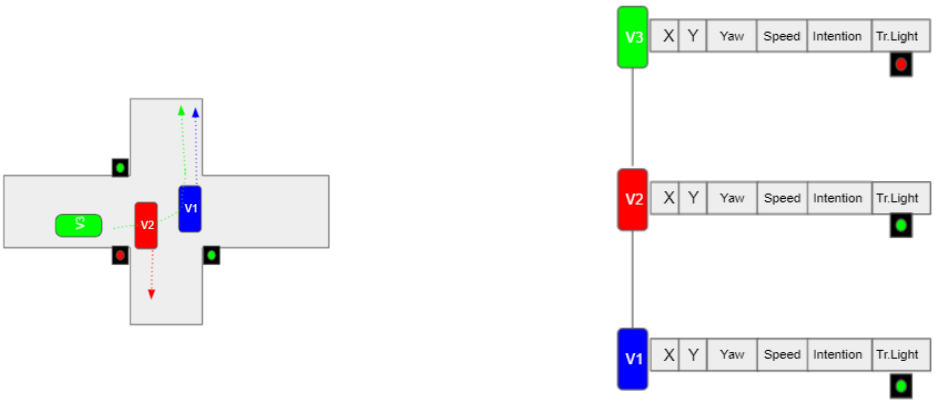


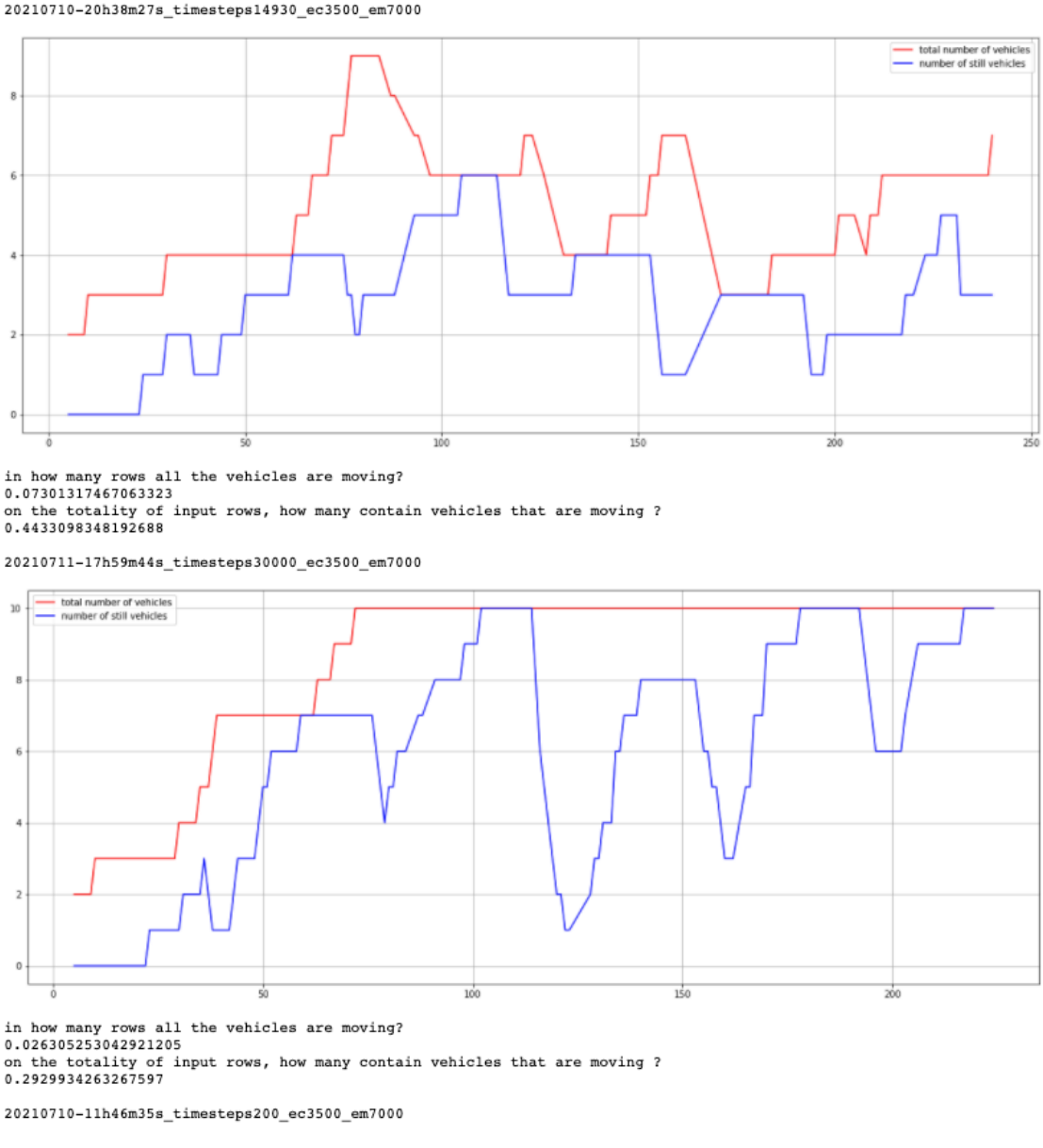
Table 1: Representation of how the traffic light duration and state is defined in the SUMO simulation.

iteration	duration	traffic light 1	traffic light 2	traffic light 3	traffic light 4
1	33	green	red	green	red
2	6	yellow	red	yellow	red
3	33	red	green	red	red
4	6	red	yellow	red	red
5	33	red	red	red	green
6	6	red	red	red	yellow

Table 2: Representative standard deviation and mean of the features of a commonly used dataset.

	x	y	speed	yaw	intention
mean	-0.94	-3.71	3.14	131.69	1.12
std	32.91	22.30	4.76	109.35	0.99

Figure 11: Comparison of distribution of vehicles among two of the most used datasets.



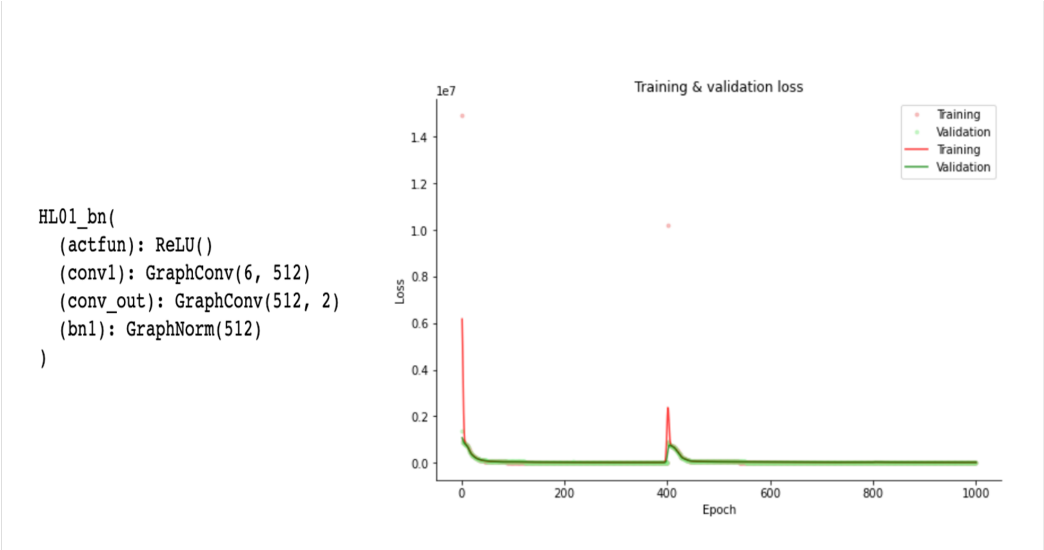
A.3 Results of Different Models

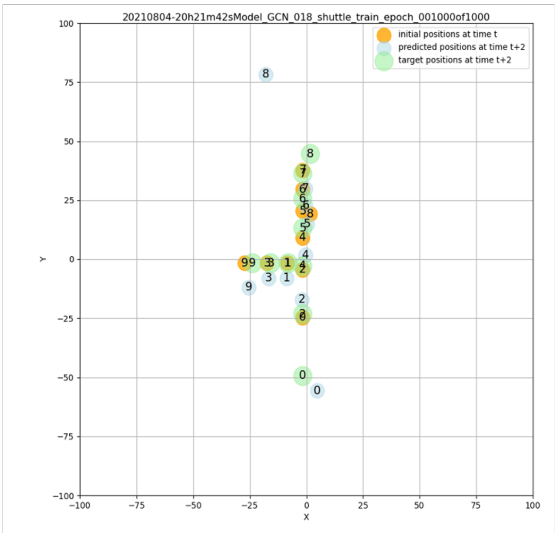
All Figures collecting results are organized as follows:

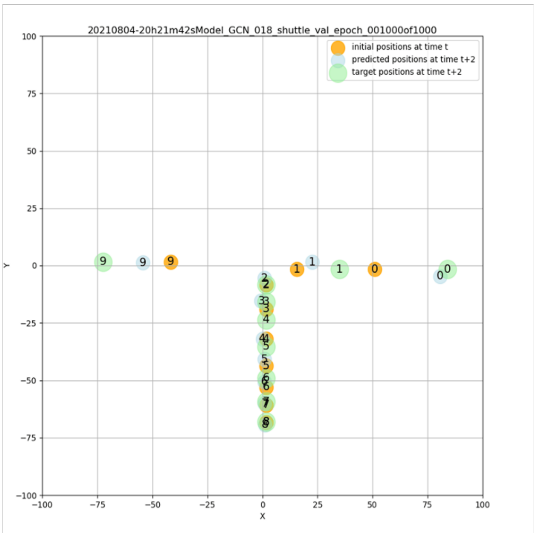
- Top left: The model.
- Top right: The loss curves of training and validation.
- Middle left: Model performance over a chosen train frame.
- Middle right: Model performance over a chosen validation frame.
- Bottom left: One of the best performing test frames.
- Bottom right: One of the worst performing test frames.

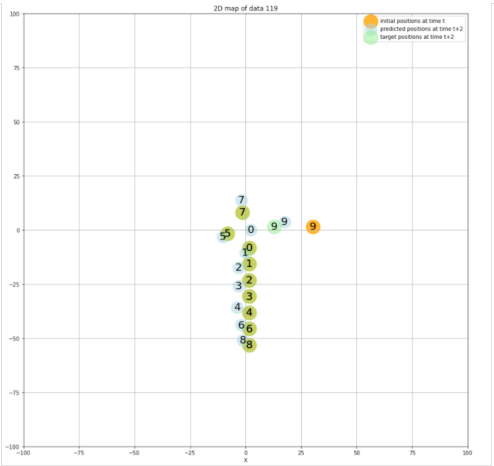
Note also that for naming convention, all the models are listed as GCN$_N NN$, $despitesomebeingactuallyMLP(eg.GCN_0 20)$.

Figure 12: GCN_18 model: Best performing GCN model.









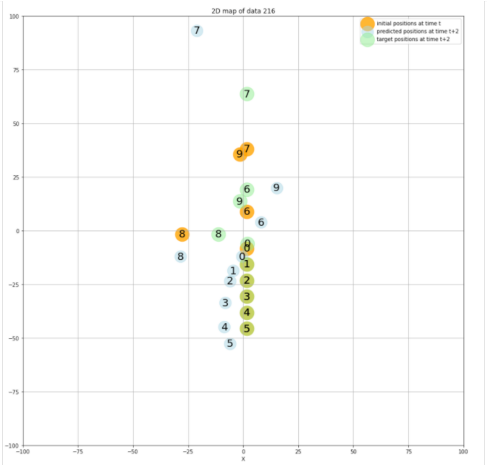
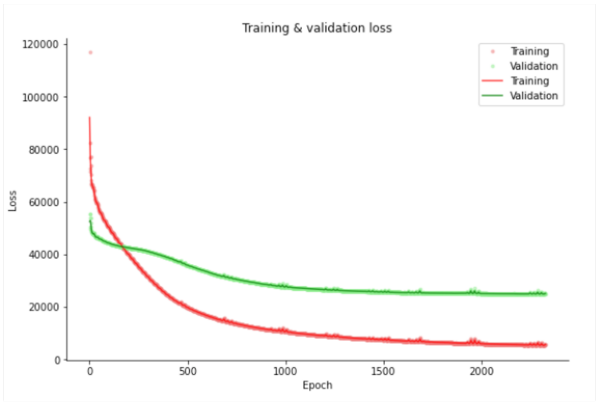
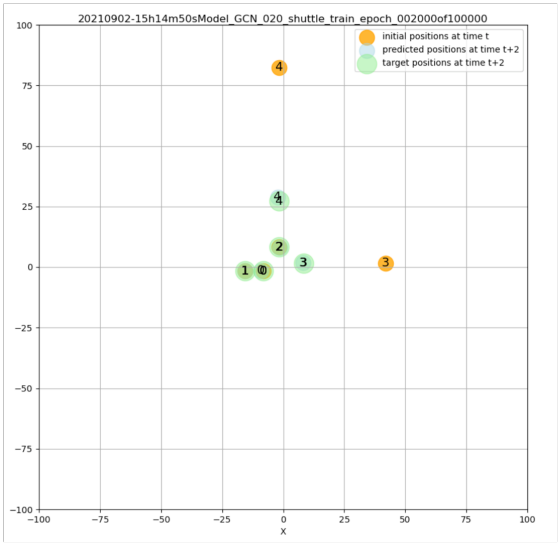
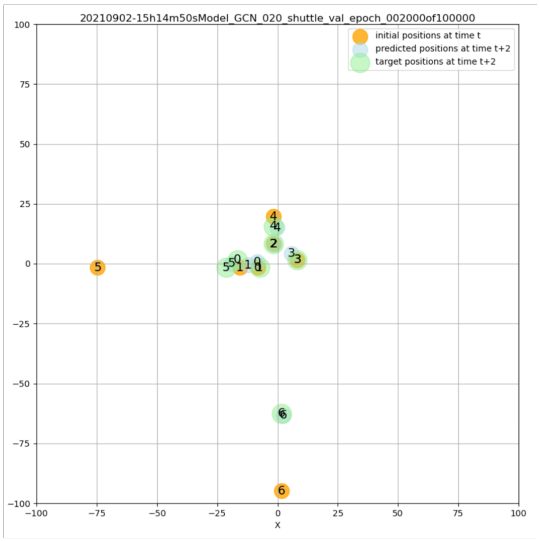


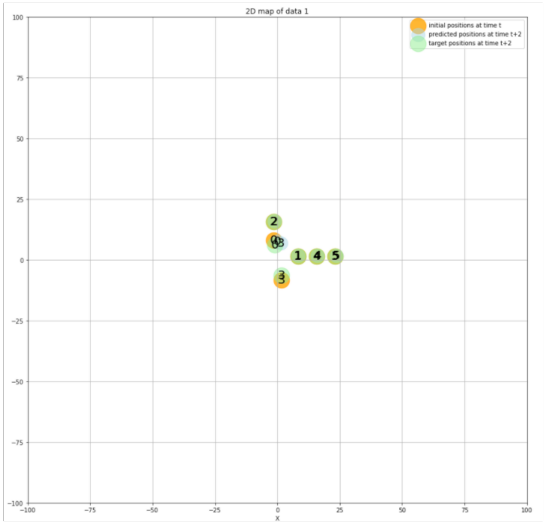
Figure 13: GCN_20 model: MLP model, using column *still vehicle*.

```
HL01_MLP(
  (actfun): ReLU()
  (lin1): Linear(in_features=7, out_features=1024, bias=True)
  (lin2): Linear(in_features=1024, out_features=3, bias=True)
)
```









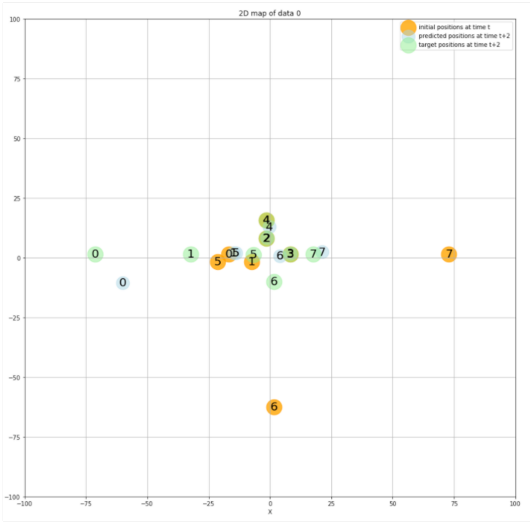
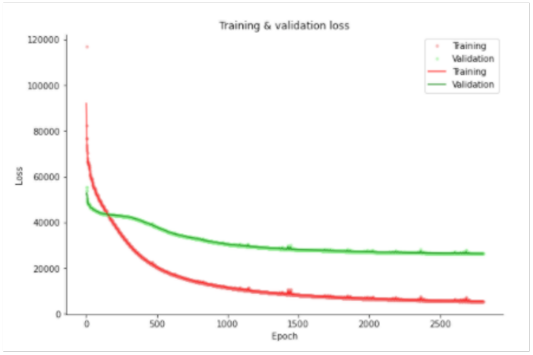
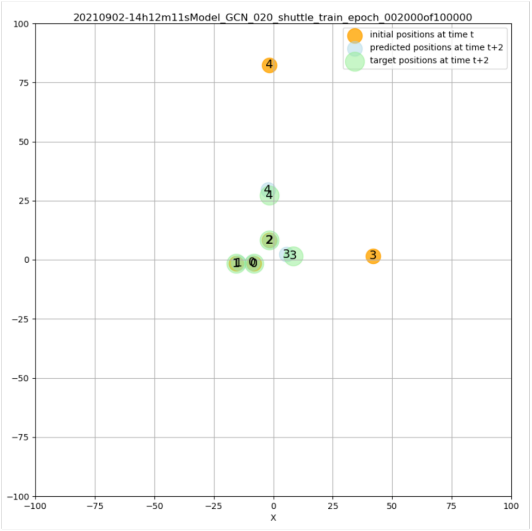
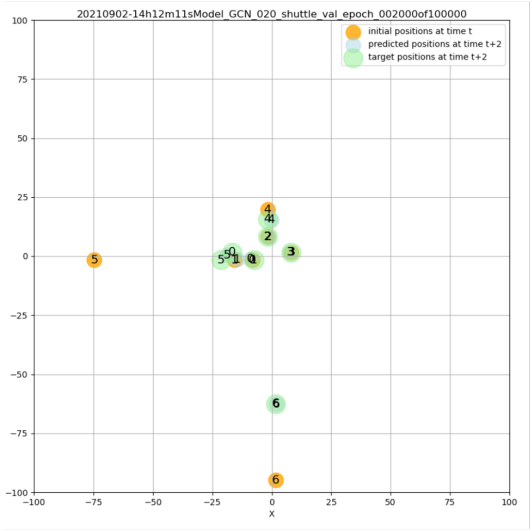


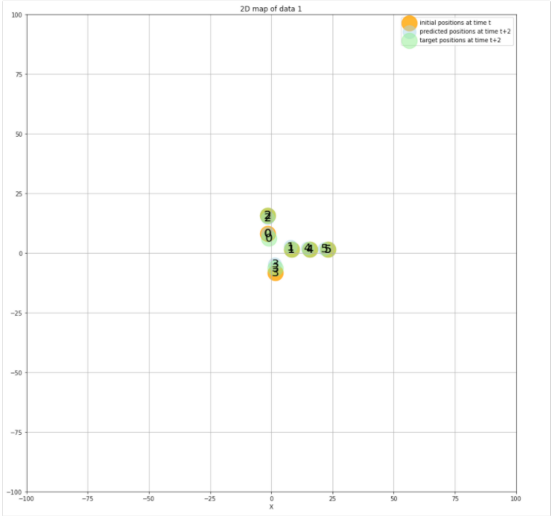
Figure 14: GCN_20 model: MLP, using column *acceleration*.

```
HL01_MLP(
  (actfun): ReLU()
  (lin1): Linear(in_features=7, out_features=1024, bias=True)
  (lin2): Linear(in_features=1024, out_features=3, bias=True)
)
```









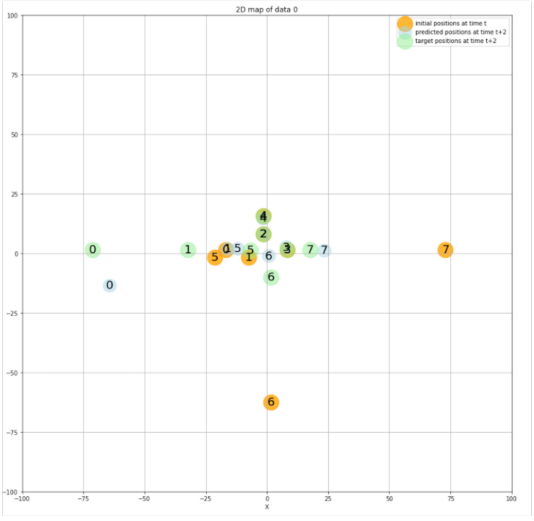
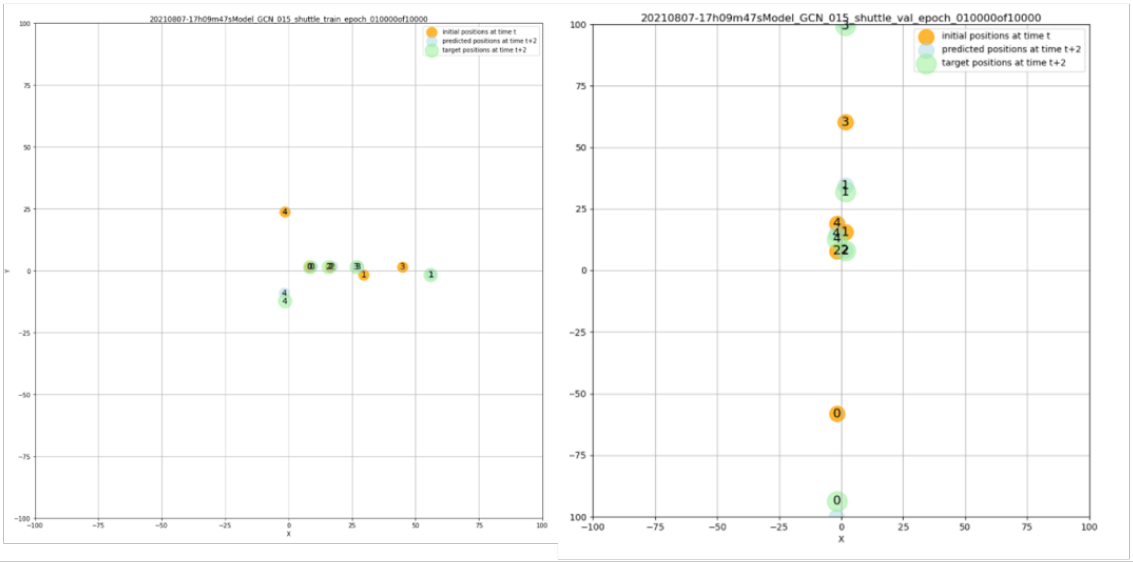


Figure 15: GCN_15 model: MLP using columns *traffic light status* and *duration until traffic light state changes*.

```
GCN_HL02_bn_relu(
  (conv1): Linear(in_features=7, out_features=64, bias=True)
  (conv2): Linear(in_features=64, out_features=128, bias=True)
  (conv3): Linear(in_features=128, out_features=2, bias=True)
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): LeakyReLU(negative_slope=0.01)
)
```



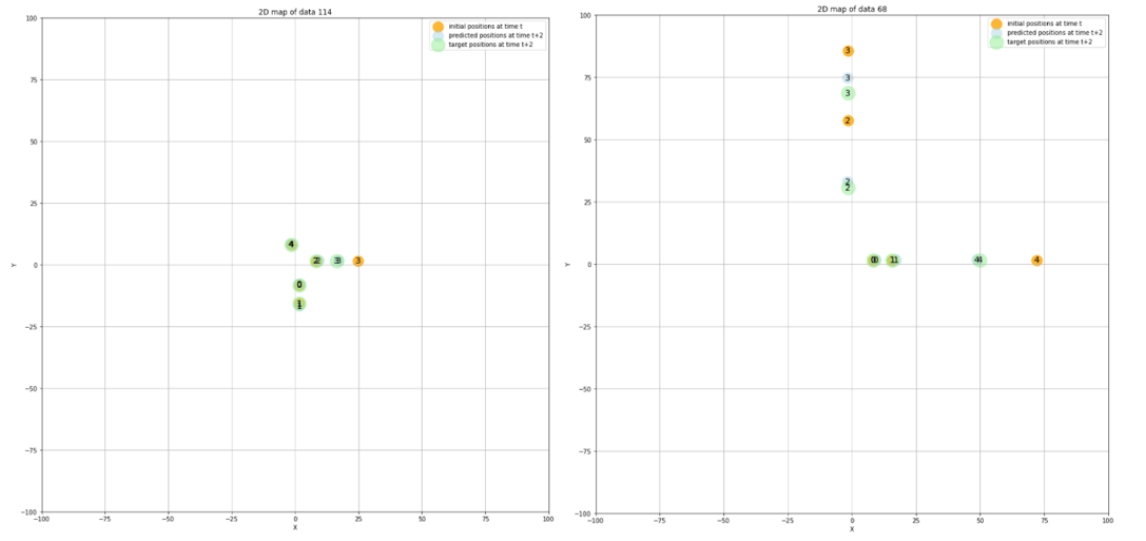


Figure 16: GCN_22 model: GCN using classification approach in combination with regression.

```
HL01_bn_regress(
  (actfun): ReLU()
  (conv1): GraphConv(7, 512)
  (conv_out): GraphConv(512, 2)
  (bn1): GraphNorm(512)
)
```