Autonomous Lane Changing using Deep Reinforcement Learning with Graph Neural Networks

Arvind S. Menon*, Lars C.P.M. Quaedvlieg*, Somesh Mehra*
Machine Learning (CS-433), École Polytechnique Fédérale de Lausanne — Fall 2022

***Abstract*—This paper introduces an approach to autonomous lane changing using deep reinforcement learning. By employing graph neural networks and deep Q-learning, general size-agnostic representations of the problem are learned. The agent is trained in a simulated environment to control the lane changing of a truck with a trailer. The final agent is able to outperform a model-predictive controller in terms of the number of crashes.**

***Keywords*-Autonomous lane changing, deep Q-learning, graph neural networks, model-predictive controller**

I. Introduction

There are several methods that have been proposed for automated lane changing in autonomous vehicles. One approach is to use a model-based controller, such as a Model Predictive Controller (MPC) [1] or a Linear Quadratic Regulator (LQR) [2], which takes into account the dynamics of the vehicle and the road environment. These controllers typically rely on a mathematical model of the vehicle and the surrounding traffic, and they optimize a predetermined performance metric, such as fuel efficiency or relative velocity to the speed limit, over a finite horizon.

However, model-based controllers can be sensitive to model uncertainty and may not be able to adapt to changing traffic conditions or unexpected behaviors of other road users. As a result, machine learning techniques have been explored as an alternative approach for automated lane changing. These methods aim to learn a control policy directly from data, without the need for explicit modeling of the vehicle or the environment.

One type of machine learning approach that has been used for lane changing is supervised learning, which involves training a model to predict the appropriate control action given a set of input features [3]. Supervised learning can be effective when a large amount of labeled data is available, but this availability is currently lacking.

More naturally, another type of machine learning approach that has been applied to lane changing is reinforcement learning (RL), which is a framework designed for stochastic sequential decision-making processes [4]. This approach has been extensively investigated [5], [6], [7], [8], [9]. The network architecture type for the model is key to solving the problem, as it determines how the RL agent processes and represents the information from the environment and how it selects actions based on this representation. Recent research employs convolutional neural networks (CNNs) as the model architecture [5], [6].

In this paper, we propose a graph neural network (GNN) architecture combined with RL and a model predictive controller to solve the problem of autonomous lane changing. There are several reasons why GNNs might be more suitable than CNNs for this task. GNNs are better equipped to learn the complex and dynamic relationships between vehicles, rather than only considering local features or patterns. They can also very naturally handle variable-sized inputs, which is more suitable for the uncertain nature of the traffic environment where scenes are constantly changing.

II. Background

A. Preliminaries of Reinforcement Learning

Reinforcement learning (RL) is the problem faced by an agent that must learn behavior through trial-and-error interactions within an environment [4]. Fully observable reinforcement learning environments can be formulated as a Markov Decision Process (MDP). Formally, this is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$ [10], where $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and $\gamma \in \mathbb{R}$. Here, $\mathcal{S}$ and $\mathcal{A}$ are the state and actions space of the agent, $\mathcal{R}$ is the reward function, $\mathbb{P}$ is the transition probability function and γ is the discount factor.

An RL agent is situated in a particular state $s \in \mathcal{S}$, where it has to take an action $a \in \mathcal{A}$. Action selection is done using the policy $\pi(a_t|s_t) \in [0, 1]$, which denotes the probability of taking action a_t given that the agent is in state s_t.

After taking action a in state s, the agent observes reward $\mathcal{R}_s^a$ and ends up in state s' with probability $\mathbb{P}_{s,s'}^a$. This form of sequential decision-making is repeated until the agent reaches a terminal state, after which the episode ends. There also exist non-episodic environments where there are no terminal states.

The return $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R_i$ is a discounted cumulative sum of rewards R_i. γ ensures that rewards observed later count exponentially less in the computation of the return from time step t. State value $v_\pi(s_t) = \mathbb{E}_\pi[G_t|S = s_t]$ is the expected return when in state s_t under policy π. $Q_\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|S = s_t, A = a_t]$ is the expected return G_t when in state s_t and taking action a_t under policy π.

One of the most promising algorithms in RL is the Q-learning algorithm [11], which uses equation (1) to smoothly

update $Q(S, A)$ to the target value y with learning rate α.

$$\begin{aligned} y_t &= r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \\ Q(s_t, a_t) &= (1-\alpha)Q(s_t, a_t) + \alpha y_t \end{aligned} \tag{1}$$

B. Graph Neural Networks

GNN is a blanket term for any neural network that deals with graph data. There exist various types of architectures for different tasks. Some popular architectures include:

- Graph Convolutional Networks (GCN): GCNs apply the concept of convolution to graph data. For each node, we get the following forward pass ($l^{th} layer$):

$$h_i^{(l)} = \sigma(\sum_{i \in N_j} c_{ij} W h_j^{(l-1)})$$

where h_i represents the node-embedding of the i^{th} node, σ is the activation function, W is the weight matrix, and $c_{ij} = \frac{1}{\sqrt{N_i N_j}}$ where N_i, N_j represent the size of the node neighborhoods.

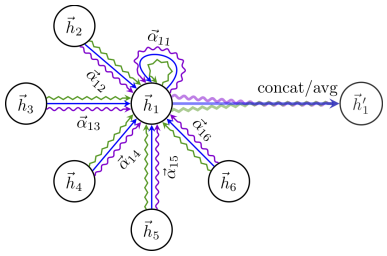

Figure 1: A Graph Attention Network (GAT) Layer

- Graph Attention Networks: GATs are similar to GCNs, however instead of a fixed weight c_{ij} for each neighbour, it uses α_{ij} as follows:

$$h_i^{(l)} = \sigma(\sum_{i \in N_j} \alpha_{ij} W h_j^{(l-1)})$$

where α_{ij} represents individual weights for neighboring nodes which are learned using the self-attention mechanism. The attention mechanism computes normalized weight coefficients for each pair of nodes i and j based on their feature vector representation,

$$\alpha_{ij} = \mathbb{F}(h_i, h_j)$$

The aim of the attention mechanism is to learn how much weight ("attention") should be given to each neighbour node.

C. Deep Reinforcement Learning

$$\begin{aligned} y &= r + \gamma \max_{a'} Q(s', a'; \theta_i^-) \\ L_i(\theta_i) &= \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[(y - Q(s, a; \theta_i)^2 \right] \end{aligned} \tag{2}$$

Since it is often intractable to store all possible Q-values, and in order to generalize better between similar states, Q-learning was extended to function approximation with deep neural networks. This algorithm, referred to as deep Q-learning [12], uses gradient-based optimization with $\nabla_{\theta_i} L_i(\theta_i)$ from equation (2) to train the model. Here, θ_i refers to the weights of the neural network architecture. Due to the instability of RL with function approximation, deep Q-learning proposes two additional changes to increase stability:

1) Since learning on a trajectory $(s_0, a_0, r_0, s_1, ..., s_T)$ violates the assumption of i.i.d. sampling, the paper proposes creating an empirical distribution from observed tuples (s_t, a_t, r_t, s_{t+1}) called $U(D)$ (the experience replay buffer), which is sampled to update the model.
2) The target y is frozen for a fixed number c of iterations using historical weights of θ called θ^-. After every c iterations, θ^- is updated to the most recent θ. This reduces variance in the loss function since it does not change to the most recent θ at every iteration.

This paper uses deep Q-learning to learn the weights of an architecture that uses graph neural networks to build an effective representation.

III. Methodology

A. Environment Representation

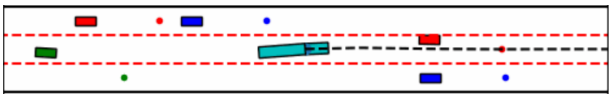

Figure 2: Example of a state of the highway driving scenario. The truck is shown in cyan. The dots represent vehicle targets, and the black dashed line in front of the truck is the current trajectory that the truck follows.

The environment is represented by the Simulation of Urban Mobility (SUMO) traffic simulator [13], [6]. A one-way highway with three lanes was simulated, where the controlled vehicle consisted of a truck-trailer combination, as depicted in figure 2. More specific configurations of the environment are discussed in section IV.

In this paper, we formulate this simulation as a fully-observable environment. As discussed in section II-A, it is formally defined as an MDP; a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$.

- State space ($\mathcal{S}$)
  Every state in the state space is defined as a graph $G = (\mathcal{V}, \mathcal{E}, F)$. Here, $\mathcal{V} = \{v_0, ...v_n\}$ is the set of vehicles present in that state. v_0 is the ego vehicle (in this case the truck) and is present in every state. $\mathcal{E}$ is the set of edges between vertices in the graph. An undirected edge is present between two vehicles if they are within a certain predefined distance from each other ($\Delta x < 10$m, $\Delta y < 30$m in our case). Self-connections are also included. It is important to note that subgraphs

disconnected from v_0 are not considered, since those do not contain useful information for the truck. Finally, F is the feature matrix, where each row F_i corresponds to the features of vehicle v_i: $\left[\Delta x_i, \Delta y_i, \frac{v_i - v_{max}}{v_{max}}, \theta_i\right]$. These are the x- and y-distances relative to the ego vehicle, the normalized difference with the maximum velocity, and steering angle respectively. An example of such a graph is displayed in figure 3.

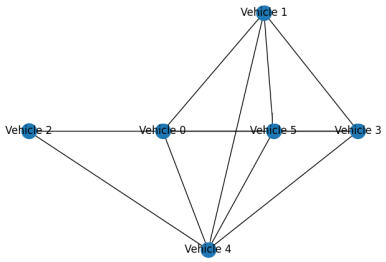

Figure 3: Example of the representation of one state

- Action space ($\mathcal{A}$)
  In the simplest case, the agent is able to perform three actions: $\mathcal{A} = \{a_L, a_N, a_R\}$, where these actions represent left lane change, no lane change, and right lane change respectively. If the agent is in the left or rightmost lanes, the invalid actions are simply masked. These actions are then executed by a trajectory planner, which is discussed in more detail in section III-B.
- Reward function ($\mathcal{R}$)
  We want to encourage the ego vehicle to make lane changes for two primary reasons: 1) for efficiency in getting to its destination (i.e. maintaining a speed close to the speed limit), and 2) for safety to avoid collisions. As such, we assign the following reward structure:

$$r = \begin{cases} r_v = \frac{v}{v_{max}} & \text{normal drive} \\ r_c = crash\_penalty & \text{crash occured} \end{cases}$$

  Here, v is the speed of the truck and v_{max} is the speed limit (60 km/h or 16.66 m/s in our case). We tested two values for the crash penalty, $r_c = \{0, -10\}$, to study how this affects the agent's learning.
- Transition probability distribution ($\mathbb{P}$)
  The transition model is defined by the previously mentioned simulation software. Since the simulation takes very small time steps (every 0.2s), the agent is defined to only take an action every k steps. Hence, the transition happens over an interval of k time units. Within that interval, the trajectory planner represents the action taken and is controlled in the simulation.

B. Agent Architecture

Figure 4a) shows the original controller workflow before introducing the RL agent. In this setup, the MPC is responsible for modeling the current state, calculating the cost of each possible decision, and then executing the optimal action. As shown in 4b), in our setup, whilst the actual trajectory planning is still handled by the MPC, the RL agent is responsible for deciding when to perform a lane change. This has the additional benefit of saving time and computation by the MPC controller since it only needs to calculate the cost of one decision as opposed to three at each iteration.

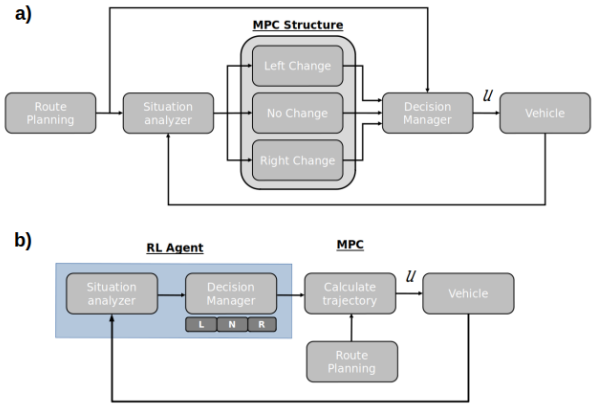

Figure 4: Controller workflow a) before and b) after introducing the RL agent.

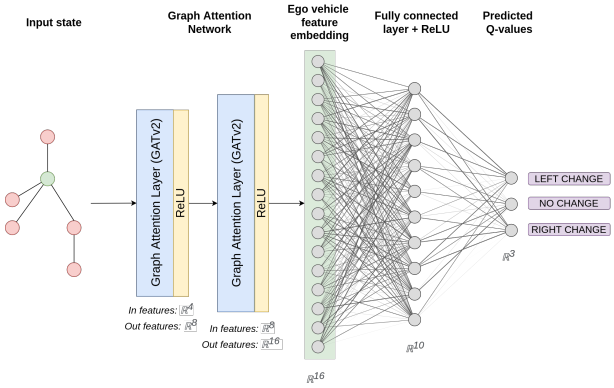

Figure 5: Architecture of the GNN

For the RL agent, we use a GNN with the architecture shown in figure 5. The network takes as input the current state, s, in its graph representation, and first applies two Graph Attention Network layers. Specifically, we use GATv2 layers, which are a more dynamic and better-performing variant of the original GAT layer [14]. This allows neighboring vehicles to aggregate information from their neighbors, and then propagate this information to the ego vehicle. After this, we extract the feature embedding of the ego vehicle and pass this through a fully connected neural network to ultimately predict the Q values for each action.

Rather than always performing the action with the highest Q-value, we also employed an ϵ-greedy exploration strategy

Crash penalty (r_c)	# episodes	# crashes	# derails	avg distance (m)	avg speed (m/s)	avg decision time (s)
No RL agent (baseline)	30	1	2	420.9	15.0	0.084
0	30	2	4	377.5	14.4	0.030
-10	30	0	4	364.5	13.7	0.012

Table I: Comparison of different crash penalties vs using only the controller as a baseline. Each episode was run for 30s.

during training to avoid getting stuck in a poor local optimum, as shown below.

$$a_t = \begin{cases} max\ Q_t(a) & \text{with probability } 1-\epsilon \\ random\ action & \text{with probability } \epsilon \end{cases}$$

Additionally, after each learning step, we decay the value of ϵ by subtracting a fixed decrement (ϵ_{dec}) until a minimum value (ϵ_{min}).

IV. Simulations and Results

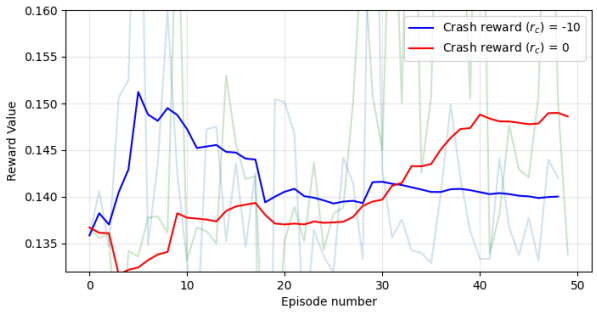

Figure 6: Plot showing of average reward per episode (with smoothing = 0.99) for crash rewards r_c = {0, -10}

To train the networks, we simulated 50 episodes of 50s each, invoking the RL agent for lane-changing decisions at every 1s interval ($k = 5$) and rewarding the decisions with the scheme described in section III-A.

The hyperparameters used for the training were: discount factor $\gamma = 0.9$, target copy delay $c = 5$, epsilon $\epsilon = 0.1$, $\epsilon_{dec} = 1e^{-3}$, $\epsilon_{min} = 0.01$, a learning rate of 0.01, memory buffer of 100,000 transitions, and batch size of 32.

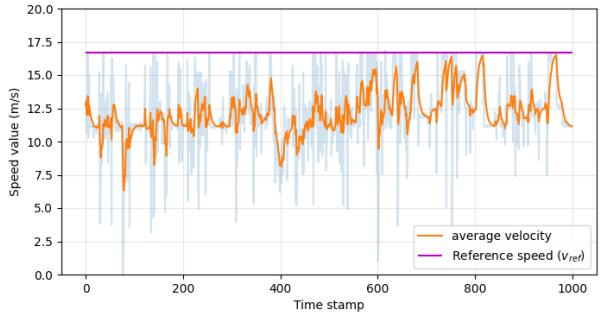

Figure 7: Plot of the truck speed (m/s) vs speed limit (16.66 m/s) at each time step during training (with smoothing=0.4)

The results for the two crash penalties ($r_c = \{0, -10\}$) we tested are shown in figure 6 and table I. In figure 6 we observe lower reward values for $r_c = -10$, which may be due to lower velocity and the expensive crash penalty. This is indicative of a more conservative approach to driving and lane changes, whereas for $r_c = 0$ we see higher reward values indicating that the vehicle is able to maintain speeds closer to the speed limit, as shown in figure 7.

This safety vs efficiency tradeoff is also present in table I, where there are no crashes for $r_c = -10$, suggesting that it is possible to train a more safety-oriented agent by heavily penalizing decisions that result in a crash.

Although the MPC makes almost-optimal decisions in the relatively simple scenarios being simulated, the RL agents still achieved comparable performance in terms of average distance and speed achieved in each episode, whilst also offering a significant speed up in average decision time.

V. Discussion and Conclusions

As outlined in [15], a major problem with deep Q-learning is that models which use a traditional experience replay mechanism tend to learn bias from imbalanced data.

We observed a strong bias towards certain actions (for example, the $r_c = 0$ model predicted 55% left change, 45% no change, and 0% right change decisions during our simulation). This was likely because the proportion of each decision in the replay buffer is highly sensitive to the network parameter initialization. One way to counteract this would be to use a more aggressive exploration policy initially (using a higher ϵ), to increase the diversity of decisions in the buffer. Another way is to use different sampling methods like stratified sampling by decision, or oversampling underrepresented decisions.

Alternatively, we could employ a curriculum learning approach by providing the agent with a more controlled learning environment. By selecting the tasks and the order in which they are presented to the agent, it is possible to create a learning path that helps the agent progress more efficiently and avoid getting stuck in local minima [16]. However, one downside is that it may limit the agent's flexibility by only selectively choosing training situations.

Nevertheless, in figure 6 we can see some evidence of improvement in the $r_c = 0$ model from episode 30 onward, with the reward function on average trending upwards. Thus, further training the model for more episodes could likely improve performance, since the RL agent would encounter new situations to learn from, and have more time to converge to an optimum.

Overall, although there are many areas for improvement, we have demonstrated the potential usage of deep RL with GNNs for automating lane-changing decisions.

REFERENCES

[1] J. Suh, H. Chae, and K. Yi, "Stochastic model-predictive control for lane change decision of automated driving vehicles," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4771–4782, 2018.

[2] H. Chae, Y. Jeong, S. Kim, H. Lee, J. Park, and K. Yi, "Design and vehicle implementation of autonomous lane change algorithm based on probabilistic prediction," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2845–2852.

[3] V. Mahajan, C. Katrakazas, and C. Antoniou, "Prediction of lane-changing maneuvers with automatic labeling and deep learning," *Transportation research record*, vol. 2674, no. 7, pp. 336–347, 2020.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[5] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2148–2155.

[6] ——, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1563–1569.

[7] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2156–2162.

[8] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[9] P. Wang, C.-Y. Chan, and A. de La Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1379–1384.

[10] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[14] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2021. [Online]. Available: https://arxiv.org/abs/2105.14491

[15] W. Yuan, Y. Li, H. Zhuang, C. Wang, and M. Yang, "Prioritized experience replay-based deep q learning: Multiple-reward architecture for highway driving decision making," *IEEE Robotics & Automation Magazine*, vol. 28, no. 4, pp. 21–31, 2021.

[16] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *arXiv preprint arXiv:2003.04960*, 2020.

* All authors contributed equally to this work