2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Dynamic Interaction-Aware Scene Understanding for Reinforcement Learning in Autonomous Driving

Maria Huegle[1], Gabriel Kalweit[1], Moritz Werling[2] and Joschka Boedecker[1,3]

Abstract— **The common pipeline in autonomous driving systems is highly modular and includes a perception component which extracts lists of surrounding objects and passes these lists to a high-level decision component. In this case, leveraging the benefits of deep reinforcement learning for high-level decision making requires special architectures to deal with multiple variable-length sequences of different object types, such as vehicles, lanes or traffic signs. At the same time, the architecture has to be able to cover interactions between traffic participants in order to find the optimal action to be taken. In this work, we propose the novel Deep Scenes architecture, that can learn complex interaction-aware scene representations based on extensions of either 1) Deep Sets or 2) Graph Convolutional Networks. We present the Graph-Q and DeepScene-Q off-policy reinforcement learning algorithms, both outperforming state-of-the-art methods in evaluations with the publicly available traffic simulator SUMO.**

I. INTRODUCTION

In autonomous driving scenarios, the number of traffic participants and lanes surrounding the agent can vary considerably over time. Common autonomous driving systems use modular pipelines, where a perception component extracts a list of surrounding objects and passes this list to other modules, including localization, mapping, motion planning and high-level decision making components. Classical rule-based decision-making systems are able to deal with variable-sized object lists, but are limited in terms of generalization to unseen situations or are unable to cover all interactions in dense traffic. Since Deep Reinforcement Learning (DRL) methods can learn decision policies from data and off-policy methods can improve from previous experience, they offer a promising alternative to rule-based systems. In the past years, DRL has shown promising results in various domains [1], [2], [3], [4]. However, classical DRL architectures like fully-connected or convolutional neural networks (CNNs) are limited in their ability to deal with variable-sized, structured inputs or to model interactions between objects.

Prior works on reinforcement learning for autonomous driving that used fully-connected network architectures and fixed sized inputs [5], [6], [7], [8], [9] are limited in the number of vehicles that can be considered. CNNs using occupancy grids [10], [11] are limited to their initial grid size. Recurrent neural networks are useful to cover temporal context, but are not able to handle a variable number of objects permutation-invariant w.r.t to the input order for a fixed time step. In [12], limitations of these architectures are shown and a more flexible architecture based on Deep Sets [13] is proposed for off-policy reinforcement learning of lane-change maneuvers, outperforming traditional approaches in evaluations with the open-source simulator SUMO.

In this paper, we propose to use Graph Networks [14] as an interaction-aware input module in reinforcement learning for autonomous driving. We employ the structure of Graphs in off-policy DRL and formalize the Graph-Q algorithm. In addition, to cope with multiple object classes of different feature representations, such as different vehicle types, traffic signs or lanes, we introduce the formalism of Deep Scenes, that can extend Deep Sets and Graph Networks to fuse multiple variable-sized input sets of different feature representations. Both of these can be used in our novel DeepScene-Q algorithm for off-policy DRL. Our main contributions are the use of Graph Convolutional Networks to model interactions between vehicles in DRL for autonomous driving and extending existing set input architectures for DRL to deal with multiple lists of different object types.

II. RELATED WORK

Graph Networks are a class of neural networks that can learn functions on graphs as input [15], [16], [17], [18], [19] and can reason about how objects in complex systems interact. They can be used in DRL to learn state representations [20], [21], [22], [17], [23], [24], [25], e.g. for inference and control of physical systems with bodies (objects) and joints (relations). In the application for autonomous driving, Graph Networks were used for supervised traffic prediction while modeling traffic participant interactions [26], where vehicles were modeled as objects and interactions between them as relations. Another type of interaction-aware network architectures, Interaction Networks, were proposed to reason about how objects in complex systems interact [18]. A vehicle behavior interaction network that captures vehicle interactions was presented in [27]. In [28], a convolutional social pooling component was proposed using a CNN to model spatial connections between vehicles for vehicle trajectory prediction.

III. METHODS

We model the task of high-level decision making for autonomous driving as a Markov Decision Process (MDP), where the agent is following a policy π in an environment

[1]Dept. of Computer Science, University of Freiburg, Germany.
{hueglem,kalweitg,jboedeck}@cs.uni-freiburg.de
[2]BMWGroup, Unterschleissheim, Germany.
Moritz.Werling@bmw.de
[3]Cluster of Excellence BrainLinks-BrainTools, Freiburg, Germany.

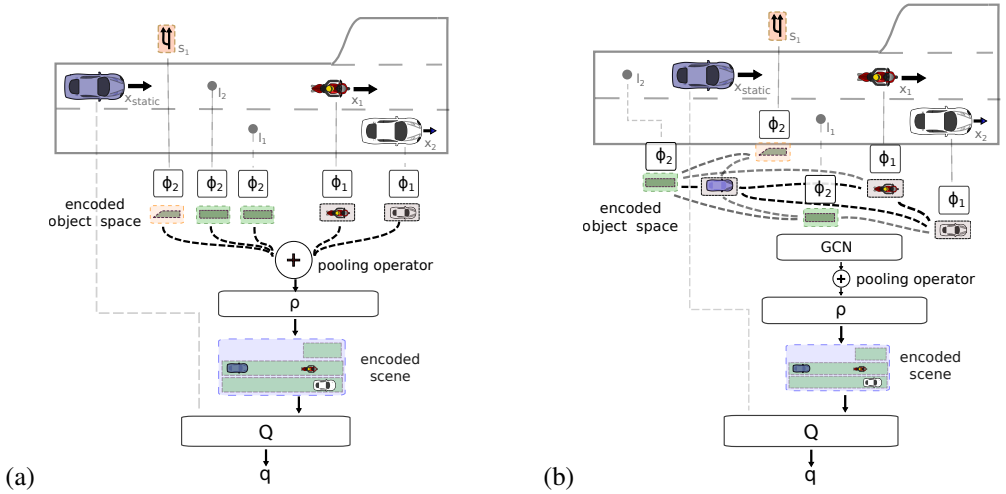

Fig. 1. Scheme of DeepScene-Q, using (a) Deep Sets and (b) Graphs. Both architectures combine multiple variable-length object lists in a scene, here traffic sign s_1, lanes l_1, l_2 and vehicles x_1, x_2. The modules ϕ. project all objects to latent vectors of the same length. For the Deep Scene-Sets, a combined representation of the scene is computed by the permutation invariant *sum* over all latent vectors and fed to the ρ network module. Concatenated with the vector x^{static}, the representation is fed to the Q network to calculate the action value output q. For the Deep Scene-Graphs, the latent output vectors are additionally fed through graph convolutional layers.

in a state s_t, applying a discrete action $a_t \sim \pi$ to reach a successor state $s_{t+1} \sim \mathcal{M}$ according to a transition model $\mathcal{M}$. In every time step t, the agent receives a reward r_t, e.g. for driving as close as possible to a desired velocity. The agent tries to maximize the expectation of the discounted long-term return $R(s_t) = \sum_{i>=t} \gamma^{i-t} r_i$, where $\gamma \in [0, 1]$ is the discount factor. In this work, we use Q-learning [29]. The Q-function $Q^\pi(s_t, a_t) = \mathbf{E}_{a_{i>t} \sim \pi}[R(s_t)|a_t]$ represents the value of following a policy π after applying action a_t. The optimal policy can be inferred from the optimal action-value function Q^* by maximization over actions. We use DQN [1] to estimate the optimal Q-function by function approximator Q, parameterized by θ^Q for state s and action a. We consider a state representation $s = (X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}, x^{\text{static}})$, consisting of K input sets $X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}$, where every set has variable length and a static input vector x^{static}. Every input set $X^{\text{dyn}_k} = [x^1, .., x^{\text{seq len}_k}]^\top$, contains of a list of feature vectors $x^j|_{1 \le j \le \text{seq len}_k}$.

In [12], this state representation with only one variable-length input set ($K = 1$) is handled by the DeepSet-Q architecture, where the Q-network consists of three network modules ϕ, ρ and Q. The features of all vehicles are projected to a latent space by the network module ϕ. The combined representation of all vehicles is then computed by $\Psi(X^{\text{dyn}}) = \rho\left(\sum_{x \in X^{\text{dyn}}} \phi(x)\right)$, which creates a fixed-length latent representation of all vehicles by summing up all latent vectors. The *sum* operator makes the architecture permutation invariant w.r.t. the order of the input [13]. Static feature representations x^{static} are fed directly to the Q-module, and the Q-values can be computed by $Q_{\mathcal{DS}} = Q(\Psi(X^{\text{dyn}})||x^{\text{static}}, a)$, where $||$ denotes a concatenation of two vectors. The resulting Q-learning algorithm is called DeepSet-Q [12].

In this work, we first propose a second, interaction-aware way of using the above state representation for only one variable input set with graphs, resulting in the Graph-Q algorithm. Then, we formalize generalizations for DeepSet-Q and Graph-Q to handle $K > 1$ input sets.

A. Graphs

In the DeepSet-Q architecture, relations between vehicles are not explicitly modeled and have to be inferred in ρ. We extend this approach by using Graph Networks, considering graphs as input. Graph Convolutional Networks (GCNs) [14] operate on graphs defined by a set of node features $X^{\text{dyn}} = [x^1, .., x^{\text{seq len}}]^\top$ and a set of edges represented by an adjacency matrix A. The propagation rule of the GCN is $H^{(l)} = \sigma(D^{\frac{1}{2}} \tilde{A} D^{\frac{1}{2}} H^{(l-1)} W^{(l-1)})$ for GCN layer $1 \le l \le L$, where we set $H^{(0)} = [\phi(x_1), ..., \phi(x_{\text{seq len}})]^\top$ using an encoder module, similar as in the Deep Sets approach. $\tilde{A} \in \mathbb{R}^{N \times N}$ is an adjacency matrix with added self-connections, $D_{i,i} = \sum_j \tilde{A}_{i,j}$, σ the activation function, $H^{(l)} \in \mathbb{R}^{N \times F}$ hidden layer activations and $W^{(l)}$ the learnable matrix of the l-th layer. The dynamic input representation can be computed from the last layer L of the GCN by computing the sum over the latent representations $H^{(L)}$ of all vehicles in sensor range similar as in the DeepSet-Q approach: $\Psi(X^{\text{dyn}}) = \rho\left(\sum_{h_x \in H^{(L)}} h_x\right)$, where ϕ is a neural network and the output vector $\phi(\cdot) \in \mathbb{R}^F$ has length F. The Q-values can be computed by $Q_{\mathcal{G}} = Q(\Psi(X^{\text{dyn}})||x^{\text{static}}, a)$. We call the corresponding Q-learning algorithm Graph-Q, see Algorithm 1.

B. Deep Scene-Sets

To generalize to $K > 1$ variable-length input sets we propose a novel architecture, Deep Scene-Sets. A combined,

Algorithm 1: Graph-Q

initialize $Q_{\mathcal{G}} = (\phi, \rho, H, Q)$ and $Q'_{\mathcal{G}} = (\phi', \rho', H', Q')$, set replay buffer $\mathcal{R}$
for *optimization step o=1,2,...* **do**
  get minibatch $(s_i, a_i, (X^{\text{dyn}}_{i+1}, x^{\text{static}}_{i+1}), r_{i+1})$ from $\mathcal{R}$
  foreach *transition* **do**
    foreach *object* x^j_{i+1} *in* X^{dyn}_{i+1} **do**
      $(\phi'_{i+1})^j = \phi'\left(x^j_{i+1}\right)$
    compute $H'^{(L)}_{i+1}$ by GCN with $H'^{(0)}_{i+1} = [(\phi'_{i+1})^1, ..., (\phi'_{i+1})^{\text{seq len}}]^\top$
    get $\Psi'_{i+1} = \rho'\left(\sum_{h_x \in H'^{(L)}_{i+1}} h_x\right)$
  $y_i = r_{i+1} + \gamma \max_a Q'(\Psi'_{i+1} || x^{\text{static}}_{i+1}, a)$
  perform a gradient step on loss: $\frac{1}{b}\sum_i \left(Q_{\mathcal{G}}(s_i, a_i) - y_i\right)^2$
  update target network by: $\theta^{Q'_{\mathcal{G}}} \leftarrow \tau\theta^{Q_{\mathcal{G}}} + (1-\tau)\theta^{Q'_{\mathcal{G}}}$

permutation invariant representation of all sets can be computed by the sum over all latent vectors of the different lists:

$$\Psi(X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}) = \rho\left(\sum_k \sum_{x \in X^{\text{dyn}_k}} \phi^k(x)\right),$$

where $1 \leq k \leq K$. The output vectors $\phi^k(\cdot) \in \mathbb{R}^F$ of the neural network modules ϕ^k have the same length F. We additionally propose to share the parameters of the last layer for the different ϕ networks. Then, $\phi^k(\cdot)$ can be seen as a projection of all input objects to the same encoded *object space*. We can combine the encoded objects of different types by the *sum* (or other permutation invariant pooling operators, such as *max*) and use the network module ρ to create an encoded *scene*, which is a fixed-sized vector. The encoded scene is concatenated to x^{static} and the Q-values can be computed by $Q_{\mathcal{D}} = Q(\Psi(X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}) || x^{\text{static}}, a)$. We call the corresponding Q-learning algorithm DeepScene-Q, shown in Algorithm 2 (Option 1) and Figure 1 (a).

C. Deep Scene-Graphs

To extend the Graph-Q algorithm to multiple variable-sized lists, we use $H^{(0)} = [\Phi^1, ..., \Phi^K]^\top$ as node features with and $\Phi^k = [\phi^k(x_1), ..., \phi^k(x_{\text{seq len}_k})]$ for $1 \leq k \leq K$, and compute the interaction-aware scene representation by:

$$\Psi(X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}) = \rho\left(\sum_{\Phi^k \in H^{(L)}} \sum_{h_x \in \Phi^k} h_x\right).$$

Similar to the Deep Scene-Sets architecture, ϕ^k are neural network modules with output vector length D and parameter sharing in the last layer. To create a fixed vector representation, we combine all node features by the sum into an encoded scene. The Q-values can be computed by $Q_{\mathcal{D}} = Q(\Psi(X^{\text{dyn}_1}, ..., X^{\text{dyn}_K}) || x^{\text{static}}, a)$. This module can replace the Deep Scene-Sets module in DeepScene-Q as shown in Algorithm 2 (Option 2) and in Figure 1 (b).

Algorithm 2: DeepScene-Q

initialize $Q_{\mathcal{D}} = (\phi^1, ..., \phi^K, \rho, [H], Q)$ and ${Q'}_{\mathcal{D}} = ({\phi^1}', ..., {\phi^K}', \rho', [H'], Q')$, set replay buffer $\mathcal{R}$
for *optimization step o=1,2,...* **do**
  get minibatch $(s_i, a_i, (X^{\text{dyn}_1}_{i+1}, ..., X^{\text{dyn}_K}_{i+1}, x^{\text{static}}_{i+1}), r_{i+1})$ from $\mathcal{R}$
  foreach *transition* **do**
    foreach *object type* $k \in (1, ..., K)$ **do**
      foreach *object* x^j_{i+1} *in* $X^{dyn_k}_{i+1}$ **do**
        $({\phi^k_{i+1}}')^j = {\phi^k}'\left(x_{i+1}{}^j\right)$
    Set (Option 1):
      get $\Psi'_{i+1} = \rho'\left(\sum_k \sum_j ({\phi^k_{i+1}}')^j\right)$
    Graph (Option 2):
      compute $H'^{(L)}_{i+1}$ by GCN with $H'^{(0)}_{i+1} = [\Phi^1, ..., \Phi^K]^\top$ and $\Phi^k = [({\phi^k_{i+1}}')^1, ..., ({\phi^k_{i+1}}')^{\text{seq len}_k}]$
      get $\Psi'_{i+1} = \rho'\left(\sum_{\Phi^k \in H^{(L)}} \sum_{h_x \in \Phi^k} h_x\right)$
    $y_i = r_{i+1} + \gamma \max_a Q'(\Psi'_{i+1} || x^{\text{static}}_{i+1}, a)$
  perform a gradient step on loss and update target network as in Algorithm 1.

D. Graph Construction

We propose two different strategies to construct bidirectional edge connections between vehicles for Graphs and Deep Scene-Graphs representations:

1) Close agent connections: Connect agent vehicle to its direct leader and follower in its own and the left and right neighboring lanes ($6 \cdot 2$ edges).
2) All close vehicles connections: Connect all vehicles to their leader and follower in their own and the left and right lanes ($K \cdot 6 \cdot 2$ edges for K surrounding vehicles).

Edge weights are computed by the inverse absolute distance between two vehicles, as shown in [26]. A fully-connected graph is avoided due to computational complexity.

E. MDP Formulation

The feature representations of the the surrounding cars and lanes are shown in section IV-A. The action space $\mathcal{A}$ consists of a discrete set of three possible actions in lateral direction: *keep lane*, *left lane-change* and *right lane-change*. Acceleration and collision avoidance are controlled by low-level controllers, that are fixed and not updated during training. Maintaining safe distance to the preceding vehicle is handled by an integrated safety module, as proposed in [11], [7]. If the chosen lane-change action is not safe, the agent keeps the lane. The reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is defined as: $r(s,a) = 1 - \frac{|v_{\text{current}}(s) - v_{\text{desired}}(s)|}{v_{\text{desired}}(s)} - p_{\text{lc}}(a)$, where v_{current} and v_{desired} are the actual and desired velocity of the agent, p_{lc} is a penalty for choosing a lane-change action and minimizing lane-changes for additional comfort.

IV. EXPERIMENTAL SETUP

We use the open-source SUMO [30] traffic simulation to learn lane-change maneuvers. All agents are trained

off-policy on datasets collected by a rule-based agent with enabled SUMO safety module integrated, performing random lane changes to the left or right whenever possible.

As dataset for training, we collected 500.000 transitions in traffic scenarios with a random number of $n \in (30, 60)$ vehicles for a *Highway* scenario and with $n \in (30, 90)$ vehicles for a *Fast Lanes* scenario. Evaluation scenarios vary in the number of vehicles $n \in (30, 35, ..., 90)$. For each fixed n, we evaluate 20 scenarios with different *a priori* randomly sampled positions and driver types for each vehicle, to smooth the high variance. In SUMO, we set the time step length to $0.5\,\mathrm{s}$. The action step length of the reinforcement learning agents is $2\,\mathrm{s}$ and the lane change duration is $2\,s$. Desired time headway τ and minimum gap are $0.5\,\mathrm{s}$ and $2\,\mathrm{m}$. All vehicles have no desire to keep right (lcKeepRight $=$ 0.0). The sensor range of the agent is $d_{\text{max}} = 80\,\mathrm{m}$. *LC2013* is used as lane-change controller for all other vehicles. To simulate traffic conditions as realistic as possible, different driver types are used, varying the parameters maxSpeed, lcCooperative, accel/decel, length, lcSpeedGain. We evaluate our approach on the following scenarios:

a) Highway: To evaluate and show the advantages of Graph-Q, we use the $1000\,\mathrm{m}$ circular highway environment shown in [12] with three continuous lanes and one object class (passenger cars).

b) Fast Lanes: To evaluate the performance of DeepScene-Q, we use a more complex scenario with a variable number of lanes, shown in Figure 2. It consists of a $1000\,\mathrm{m}$ circular highway with three continuous lanes and additional fast lanes in two $250\,\mathrm{m}$ sections. At the end of lanes, vehicles slow down and stop until they can merge into an ongoing lane. The agent receives information about additional lanes in form of traffic signs starting $200\,\mathrm{m}$ before every lane start or end. Further, different vehicle types with different behaviors are included, i.e. cars, trucks and motorcycles with different lengths and behaviors. For simplicity, we use the same feature representation for all vehicle classes.

A. Input Features

In the *Highway* scenario, we consider relative distance, velocity and relative lane index as input features, as proposed in [12]. For the *Fast Lanes* scenario, the input features used for vehicles are the same, except for an additional feature for the vehicle length. The state representation for lane j is:

- *lane start and end*: distances (km) to lane start and end
- *lane valid*: lane currently passable
- *relative lane index*: $dl_j = l_j - l_{\text{agent}} \in \mathbb{N}$, where l_j, l_{agent} are lane indices.

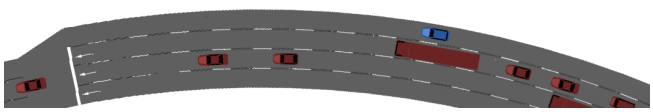

Fig. 2. *Fast Lanes* scenario in SUMO. The agent (blue) is overtaking other vehicles (red) on the fast lane and has to merge before the lane ends.

Social CNN	VBIN	GCN
Input($B \times 80 \times 5$)	Input($B \times 15$)	Input($B \times \text{seq} \times 3$)
ϕ: FC(20), FC(80) $16 \times$ Conv2D(3×1) $32 \times$ Conv2D(3×1)	ϕ: FC(20), FC(80) concat($\cdot$) ρ: FC(80), FC(20)	ϕ: FC(20), FC(80) $1\times$ GCN(80) sum($\cdot$)
concat($\cdot$, Input($B \times 3$)) FC(100)*, FC(100), Linear(3)		

Deep Scene-Sets	Deep Scene-Graphs
Input($B \times \text{seq}_0 \times 4$) and Input($B \times \text{seq}_1 \times 4$)	
ϕ_0: FC(20), FC(80),FC(80)** ϕ_1: FC(20), FC(80), FC(80)** sum($\cdot$) ρ: FC(80), FC(80)	ϕ_0: FC(20), FC(80),FC(80)** ϕ_1: FC(20), FC(80),FC(80)** $1 \times$ GCN(80) sum($\cdot$)
concat($\cdot$, Input($B \times 3$)) FC(100), FC(100), Linear(3)	

TABLE I

NETWORK ARCHITECTURES. FC($\cdot$) ARE FULLY-CONNECTED LAYERS. THE CNN USES STRIDES OF (2×1). (*) FOR VBIN FC(200). (**) PARAMETERS OF THE LAST LAYERS ARE SHARED.

Architecture	Parameter	Configuration Space
Encoders	ϕ: num layers	$1, 2, 3$
	ϕ: hidden/ output dims	$5, 20, 80, 100$
Deep Sets	ρ: num layers	$1, 2, 3$
	ρ: hidden/ output dims	$5, 20, 100$
GCN	num GCN layers	1,2,3
	hidden and output dim	20, 80
	use edge weights	True, False
SocialCNN	CONV: num layers	$2, 3$
	kernel sizes	$([7, 3, 2], [2, 1])$
	strides	$([2, 1], [2, 1])$
	filters	$8, 16, 32$
VBIN	ϕ : output dim	20, 80
	ρ : hidden dim	20, 80, 160, 200
	Q : hidden dim	100, 200
Deep Scene-Sets	ρ : output dim	20, 80
	shared parameters	True, False
Deep Scene-Graphs	use ρ network	True, False
	ρ : output dim	20, 80
	shared parameters	True, False

TABLE II

RANDOM SEARCH CONFIGURATION SPACE. FOR EVERY ARCHITECTURE, WE SAMPLED 20 CONFIGURATIONS TO FIND THE BEST SETTING.

For the agent, the normalized velocity $v_{\text{current}}/v_{\text{desired}}$ is included, where v_{current} and v_{desired} are the current and desired velocity of the agent. Passenger cars, trucks and motorcycles use the same feature representation. When the agent reaches a traffic sign indicating a starting (ending) lane, the lane features get updated until the start (end) of the lane.

B. Comparative Analysis

We use DQN to train in an offline fashion on mini-batches sampled from a fixed replay buffer $\mathcal{R}$ with transitions collected by a driver policy $\hat{\pi}$. As loss, we use $L(\theta^Q) = \frac{1}{b}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$ with targets $y_i = r_i + \gamma \max_a Q'(s_{i+1}, a|\theta^{Q'})$, where Q' is a target network, pa-

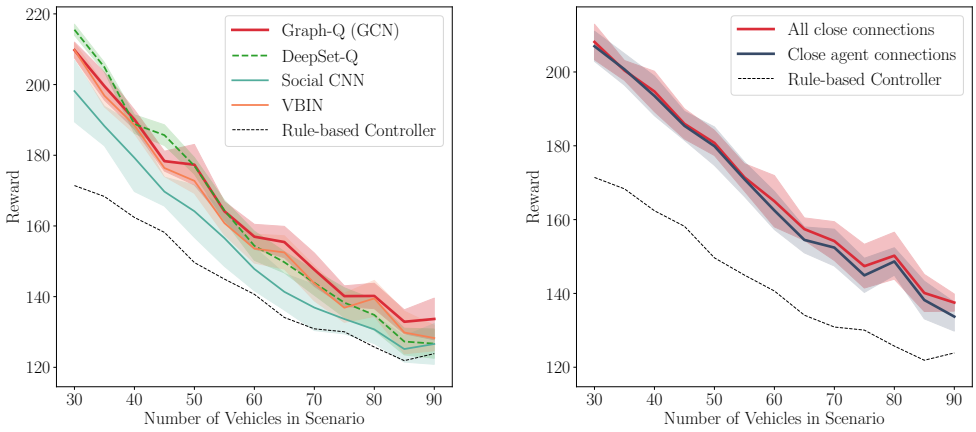

Fig. 3. Mean performance and standard deviation in the *Highway* scenario over 10 training runs for Graph-Q with all close vehicle connections, the Deep Sets [12] and two other interaction-aware Q-function input modules (left), and Graph-Q using the two proposed graph construction strategies (right). The number of vehicles indicates the traffic intensity, from light to dense traffic.

rameterized by $\theta^{Q'}$, and $(s_i, a_i, s_{i+1}, r_i)|_{0\leq i\leq b}$ is a randomly sampled minibatch from $\mathcal{R}$. For the target network, we use a soft update, i.e. $\theta^{Q'} \leftarrow \tau\theta^{Q}+(1-\tau)\theta^{Q'}$ with update step-size $\tau \in [0, 1]$. Further, we use a variant of Double-Q-learning [31] which is based on two Q-network pairs and uses the minimum of the predictions for the target calculation, similar as in [32]. Each network is trained with a batch size of 64 and optimized by Adam [33] with a learning rate of 10^{-4}. As activation function, we use Rectified Linear Units (ReLu) in all hidden layers of all architectures. The target networks are updated with a step-size of $\tau = 10^{-4}$. All network architectures, including the baselines, were optimized using Random Search with the same budget of 20 training runs. The Deep Sets architecture and hyperparameter-optimized settings for all encoder networks are used from [12]. The network architectures are shown in Table I. Graph-Q is compared to two other interaction-aware Q-learning algorithms, that use input modules originally proposed for supervised vehicle trajectory prediction. To support our architecture choices for the Deep Scene-Sets, we compare to a modification with separate ρ networks. We use the following baselines[1]:

a) Rule-Based Controller: Naive, rule-based agent controller, that uses the SUMO lane change model *LC2013*.

b) Convolutional Social Pooling (SocialCNN): In [28], a social tensor is created by learning latent vectors of all cars by an encoder network and projecting them to a grid map in order to learn spatial dependencies.

c) Vehicle Behaviour Interaction Networks (VBIN): In [27], instead of summarizing the output vectors as in the Deep Sets approach, the vectors are concatenated, which results in a limitation to a fixed number of cars. We consider the 6 vehicles surrounding the agent (leader and follower on own, left and right lane).

d) Multiple ρ-networks: Deep Scene architecture where all object types are processed separately by using K different ρ-network modules. The K resulting output vectors are concatenated as $\left[\rho^1\left(\sum_{x\in X^{\text{dyn}_1}}\phi^1(x)\right), ..., \rho^K\left(\sum_{x\in X^{\text{dyn}_K}}\phi^K(x)\right)\right]$ and fed into the Q-network module.

[1]Since we do not focus on including temporal context, we adapt recurrent layers to fully-connected layers in all baselines.

C. Implementation Details & Hyperparameter Optimization

All networks were trained for $1.25\cdot10^6$ optimization steps. The Random Search configuration space is shown in Table II. For all approaches except VBIN, we used the same ϕ and Q architectures. Due to stability issues, we adapted these parameters for VBIN. For SocialCNN, we used the optimized grid from [12] with a size of 80×5. The GCN architectures were implemented using the pytorch geometric library [34].

V. RESULTS

The results for the *Highway* scenario are shown in Figure 3. Graph-Q using the GCN input representation (with all close vehicle connections) is outperforming VBIN and Social CNN. Further, the GCN input module yields a better performance compared to Deep Sets in all scenarios besides in very light traffic with rare interactions between vehicles. While the Social CNN architecture has a high variance, VBIN shows a better and more robust performance and is also outperforming the Deep Sets architecture in high traffic scenarios. This underlines the importance of interaction-aware network modules for autonomous driving, especially in urban scenarios. However, VBIN are still limited to fixed-sized input and additional gains can be achieved by combining both variable input and interaction-aware methods as in Graph Networks. To verify that the shown performance increases are significant, we performed a T-Test exemplarily for 90 car scenarios:

- Independence of the mean performances of DeepSet-Q and Graph-Q is significant with a p-value of 0.0011.
- In the considered scenarios, there is no statistically significant difference in performance between Graph-Q and VBIN (p-value of 0.0848). However, Graph-Q is more flexible and can consider a variable number of surrounding vehicles.

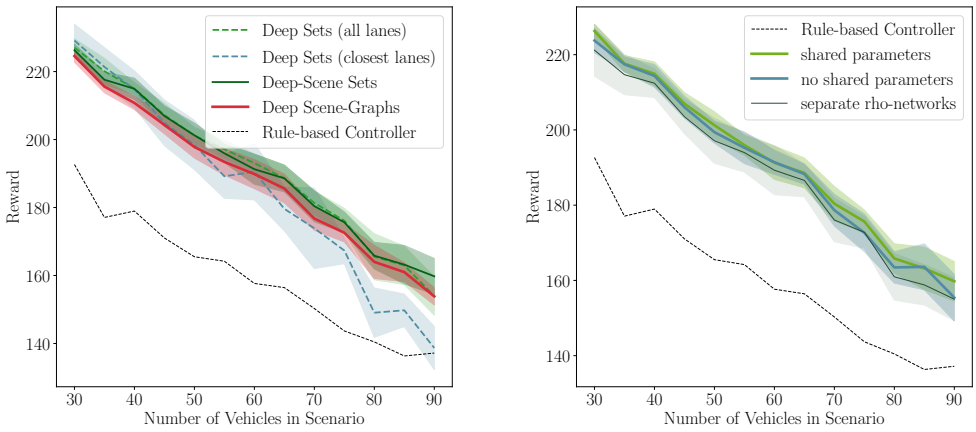

Fig. 4. Mean performance and standard deviation in the *Fast Lanes* scenario over 10 training runs for Deep Scene-Sets, Deep Scene-Graphs and the rule-based controller from SUMO (left), and different architecture choices of the Deep Scenes (right). The number of vehicles indicates the traffic intensity.

Figure 3 (right) shows the performance of the two graph construction strategies. A graph built with connections for all close vehicles outperforms a graph built with close agent connections only. However, the performance increase is only slight, which indicates that interactions with the direct neighbors of the agent are most important.

To highlight the importance of interaction-aware scene understanding, we exemplarily show two test scenarios in Figure 5. In the upper scenario, the optimal strategy is to change to the left lane because the vehicle with velocity $18\,\mathrm{m\,s^{-1}}$ will most likely overtake it's leader. In the lower scenario, the best action is to change to the right lane, because the vehicle with velocity $12\,\mathrm{m\,s^{-1}}$ will change to the left lane. While the DeepSet-Q agent acts optimally w.r.t. to its limited observation of the current scene (without considering interactions between vehicles), the Graph-Q agent anticipates the effect of relations between other vehicles and thus can leverage this information in the current decision step.

The evaluation results for *Fast Lanes* are shown in Figure 4 (left). The vehicles controlled by the rule-based controller rarely use the fast lane. In contrast, our agent learns to drive on the fast lane as much as possible (39.0% of the driving time). We assume, that the Deep Scene-Sets are outperforming Deep Scene-Graphs slightly, because the agent has to deal with less interactions than in the *Highway* scenario. Finally, we compare Deep Scene-Sets to a basic Deep Sets architecture with a fixed feature representation. Using the exact same lane features (if necessary filled with dummy values), both architectures show similar performance. However the performance collapse for the Deep Sets agent considering only its own, left and right lane shows, that the ability to deal with an arbitrary number of lanes (or other object types) can be very important in certain situations. Due to its limited lane representation, the Deep Sets (closest lanes) agent is not able to see the fast lane and thus significantly slower. Figure 4 (right) shows an ablation study, comparing the performance of the Deep-Scene Sets with and without shared parameters in the last layer of the encoder networks. Using shared parameters in the last layer leads to a slight increase in robustness and performance, and outperforms the architecture with separate ρ networks.

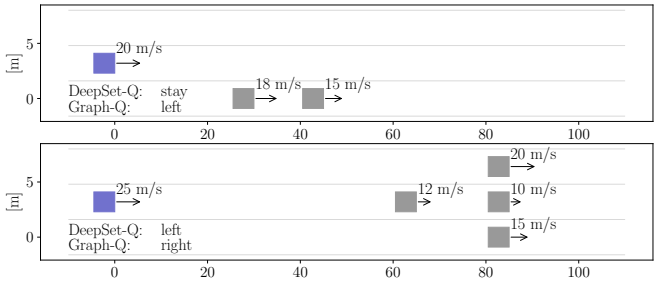

Fig. 5. Two exemplary test cases to compare the behaviors of DeepSet-Q and Graph-Q. The agent is shown in blue with the best actions shown below. Surrounding vehicles are shown in grey.

VI. CONCLUSION

In this paper, we propose Graph-Q and DeepScene-Q, interaction-aware reinforcement learning algorithms that can deal with variable input sizes and multiple object types in the problem of high-level decision making for autonomous driving. We showed, that interaction-aware neural networks, and among them especially GCNs, can boost the performance in dense traffic situations. The Deep Scene architecture overcomes the limitation of fixed-sized inputs and can deal with multiple object types by projecting them into the same encoded object space. The ability of dealing with objects of different types is necessary especially in urban environments. In the future, this approach could be extended by devising algorithms that adapt the graph structure of GCNs dynamically to adapt to the current traffic conditions. Based on our results, it would be promising to omit graph edges in light traffic, essentially falling back to the Deep Sets approach, while it is beneficial to model more interactions with increasing traffic density.

REFERENCES

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2746–2754, 2015.

[4] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17:39:1–39:40, 2016.

[5] Peter Wolf, Karl Kurzer, Tobias Wingert, Florian Kuhnt, and J. Marius Zöllner. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. *CoRR*, abs/1809.03214, 2018.

[6] Branka Mirchevska, Manuel Blum, Lawrence Louis, Joschka Boedecker, and Moritz Werling. Reinforcement learning for autonomous maneuvering in highway scenarios. *11. Workshop Fahrerassistenzsysteme und automatisiertes Fahren*.

[7] Branka Mirchevska, Christian Pek, Moritz Werling, Matthias Althoff, and Joschka Boedecker. High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2156–2162, 2018.

[8] Masoud Nosrati, Elmira Amirloo Abolfathi, Mohammed Elmahgiubi, Peyman Yadmellat, Jun Luo, Yunfei Zhang, Hengshuai Yao, Hongbo Zhang, and Anas Jamil. Towards practical hierarchical reinforcement learning for multi-lane autonomous driving. *2018 NIPS MLITS Workshop*, 2018.

[9] Meha Kaushik, Vignesh Prasad, Madhava Krishna, and Balaraman Ravindran. Overtaking maneuvers in simulated highway driving using deep reinforcement learning. pages 1885–1890, 06 2018.

[10] Mustafa Mukadam, Akansel Cosgun, and Kikuo Fujimura. Tactical decision making for lane changing with deep reinforcement learning. *NIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2017.

[11] Lex Fridman, Benedikt Jenik, and Jack Terwilliger. DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1801.02805, January 2018.

[12] Maria Huegle, Gabriel Kalweit, Branka Mirchevska, Moritz Werling, and Joschka Boedecker. Dynamic input for deep reinforcement learning in autonomous driving. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 7566–7573. IEEE, 2019.

[13] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc., 2017.

[14] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, January 2009.

[16] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[17] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *CoRR*, abs/1806.01242, 2018.

[18] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016.

[19] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[20] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *CoRR*, abs/1704.01665, 2017.

[21] Jessica B. Hamrick, Kelsey R. Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Joshua B. Tenenbaum, and Peter W. Battaglia. Relational inductive bias for physical construction in humans and machines. *CoRR*, abs/1806.01203, 2018.

[22] Jiechuan Jiang, Chen Dun, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *CoRR*, abs/1810.09202, 2018.

[23] Jiaxuan You, Bowen Liu, Rex Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *CoRR*, abs/1806.02473, 2018.

[24] Prithviraj Ammanabrolu and Mark O. Riedl. Playing text-adventure games with graph-based deep reinforcement learning. *CoRR*, abs/1812.01628, 2018.

[25] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[26] Frederik Diehl, Thomas Brunner, Michael Truong-Le, and Alois Knoll. Graph neural networks for modelling traffic participant interaction. *CoRR*, abs/1903.01254, 2019.

[27] Wenchao Ding, Jing Chen, and Shaojie Shen. Predicting vehicle behaviors over an extended horizon using behavior interaction network. *CoRR*, abs/1903.00848, 2019.

[28] Nachiket Deo and Mohan M. Trivedi. Convolutional social pooling for vehicle trajectory prediction. *CoRR*, abs/1805.06771, 2018.

[29] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[30] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker-Walz. Recent development and applications of sumo - simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 3&4, 12 2012.

[31] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[32] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1582–1591, 2018.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[34] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.