Attention-based Hierarchical Deep Reinforcement Learning for Lane Change Behaviors in Autonomous Driving

Yilun Chen[1], Chiyu Dong[2], Praveen Palanisamy[3], Priyantha Mudalige[3],
Katharina Muelling[1] and John M. Dolan[1] *†‡

Abstract

Performing safe and efficient lane changes is a crucial feature for creating fully autonomous vehicles. Recent advances have demonstrated successful lane following behavior using deep reinforcement learning, yet the interactions with other vehicles on road for lane changes are rarely considered. In this paper, we design a hierarchical Deep Reinforcement Learning (DRL) algorithm to learn lane change behaviors in dense traffic. By breaking down overall behavior to sub-policies, faster and safer lane change actions can be learned. We also apply temporal and spatial attention to the DRL architecture, which helps the vehicle focus more on surrounding vehicles and leads to smoother lane change behavior. We conduct our experiments in the TORCS simulator and the results outperform the state-of-art deep reinforcement learning algorithm in various lane change scenarios.

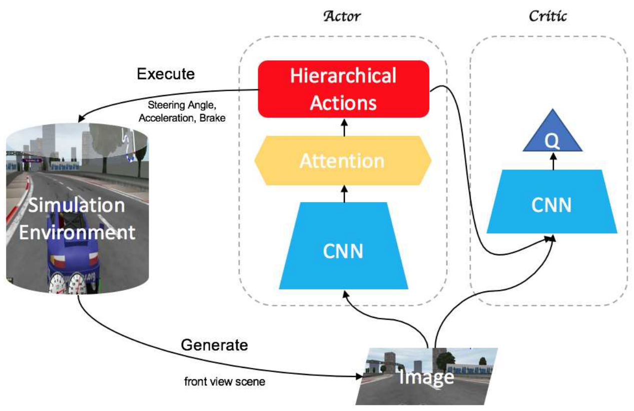

Figure 1: Illustration of the algorithm. Our algorithm is based on deep reinforcement learning with actor and critic. The actor network is for policy learning and the critic network is for policy evaluation. We propose hierarchical actions and an attention mechanism to generate lane change behavior. We use simulation environment from TORCS.

1. Introduction

Deep reinforcement learning (DRL) has recently emerged as a new way to learn driving policies. With a deep reinforcement learning algorithm, the autonomous agent can obtain driving skills by learning from trial and error without any human supervision. Previous work [24, 20] mostly focused on applying Deep Q Network (DQN) [17] or Deep Deterministic Policy Gradient (DDPG) [14] to learn to drive. These works successfully demonstrated that a car can learn to drive without leaving the road boundary. However, most of these works focus on the task of lane following while the interactions with the surrounding vehicles are often ignored. In real traffic scenarios, we need to consider more complex interactive behaviors such as lane changing for various driving purposes. Previously, rule-based lane change maneuvers [2, 16] have been developed and researched, but less effort has been made to consider lane change with deep reinforcement learning algorithms.

In this paper, we propose to use a deep reinforcement learning-based method that can learn sub-policies for lane changing behavior. The structural overview of our algorithm is shown in Figure 1. Lane change is a fundamental behavior in on-road driving for overtaking or navigation purposes. It requires high-level reasoning about surrounding vehicles' intentions and behavior to form an effective driving strategy. At the same time, it requires low-level reasoning to plan what exact path to follow under the safety requirements, generally known as the path planning problem. Each of these parts has been researched separately in previous literature [23], but they are rarely optimized together. Our method combines these two levels of reasoning in one network with a hierarchical structure, which can still be trained in an end-to-end fashion. By designing a hierarchical action space, our network can maintain a high-

*1 The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA `yilunc, katharam, jdolan@andrew.cmu.edu`
†2 Department of Eletrical and Computing Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA `chiyud@andrew.cmu.edu`
‡3 General Motor Global R&D Center, Warren, MI 48090 USA

level strategy and a low-level control command at the same time, while being differentiable end-to-end. This encourages shared computation and optimizes for the overall performance.

The concept of attention is also applied to autonomous driving. Normally, human drivers do not usually pay equal attention to all available sensory information. People select the information based on a higher level of cognition for determining the current maneuver and ignore the unrelated information. Here we incorporate the concept of attention from recent advances in image captioning tasks [26] to explicitly model this behavior. We show that during the training of deep reinforcement learning, our attention mechanism will automatically focus on feasible paths or surrounding on-road vehicles that may influence our driving behavior. This can eventually help promote the quality of the learned lane change behavior. The attention-enhanced network can make better use of perception information and lead to shorter convergence time and better performance. It also leads to better explainability of the network.

We summarize our contribution as follows. First, we propose a hierarchical deep reinforcement learning algorithm that can deal with lane change behaviors on road. Our method can be easily extended to learn multiple driving policies in one model. Second, we develop an attention mechanism that is suitable for learning driving policies with images. This helps improve compositionality of network: learn better performance with fewer examples. Finally, we report experimental results and analysis in lane change behaviors with a comparison of state-of-the-art deep reinforcement learning algorithms.

2. Related Work

Traditional lane change maneuvers use rule-based designs without learning. In DARPA challenge, the CMU Boss self-driving vehicle [2] made lane change decisions by checking empty slots. More recent methods apply sampling-based or optimization-based methods [13, 16]. All these methods include intensive hand-engineering where we have to set up lots of hand-designed parameters and spend large amounts of time tuning them. We can avoid this by learning lane change behaviors with data.

There are currently two main approaches to learning for self-driving vehicles: supervised learning and reinforcement learning.

One of the earliest supervised learning approaches was ALVINN [18], which used a neural network to directly map front-view camera images to steering angle. Recently, Nvidia [3, 6, 19] extended it with deep neural networks to demonstrate lane following with more complex real-world scenarios. [25] scaled this effort to a larger crowd-sourced dataset and proposed the FCN-LSTM architecture to derive a generic driving model. [28] extended the concept of learning steering angle to learning to control speed. [4] and [1] map images to a number of key perception indicators, which are called affordance. The affordance is later associated with actions by hand-designed rules. All these works target the task of steering the vehicle to achieve lane-keeping behavior but do not consider interactive scenarios such as lane changes.

Supervised learning requires a high amount of human-labeled driving data, which makes it expensive and hard to scale in reality. Deep reinforcement learning avoids massive human-labeled driving data and has the potential to learn to recover from unseen or unsuccessful states, which is a well-known limitation in supervised learning or imitation learning. A number of attempts used deep reinforcement learning to learn driving policies: [21] learned a safe multi-agent model for autonomous vehicles on the road and [9] learned a driving model for racing cars.

In this work, A deep reinforcement learning (DRL) with a novel hierarchical structure for lane changes is developed. Hierarchical actions for multi-scale tasks have been actively studied in DRL [12, 27]. [8] developed a parameterized action space for the agent to learn to play soccer in the RoboCup competition. We take the idea of parameterized action space and design a hierarchical action structure suitable for lane change behavior.

Attention mechanism also contributes to the interactive behavior generation. Attention models are widely used in domains such as image captioning [26] and machine translation [15]. Concurrent research introduces attention to autonomous driving. [5] investigated attention in driving from a cognitive perspective and [10]interpreted driving behavior learning with attention. We adopt the idea of temporal and spatial attention to encourage lane change behaviors.

3. Method

In this section, we first introduce the hierarchical action space designed for lane change behaviors and describe our overall approach to learning a driving model. We then dive into the attention mechanism that helps improve the learning both in efficiency and performance. Finally, we discuss the reward signal design for the DRL algorithm.

3.1. Hierarchical Action Space for Lane Change

The lane change behaviors in driving policies requires high-level decisions (whether to make a lane change) and low-level planning (how to make the lane change). Based on the parameterized action space [8], we create a hierarchical action space for autonomous driving lane changes as shown in Figure 2. We define three mutually exclusive discrete high-level actions: Left Lane Change, Lane Following and Right Lane Change. At each time step, the agent must choose one of the three high-level actions to

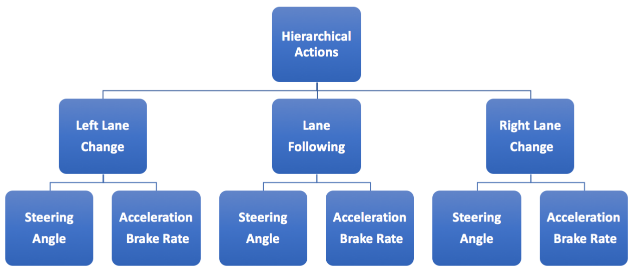

Figure 2: Illustration of the hierarchical action space for lane change behavior.

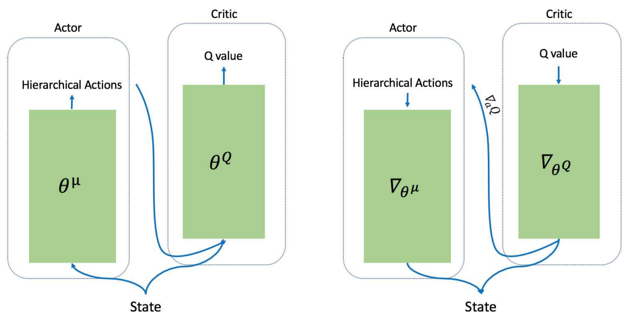

Figure 3: The Actor-Critic architecture used in deep reinforcement learning, first introduced in [14]. On the left is the data flow during inference (forward pass) and on the right is the gradient flow during training (back-propagation).

execute. Each action requires two continuous-valued parameters that must be specified. The first parameter is the steering angle with the value in the range of $[-60, 60]$ degrees. Large steering angles are intentionally prevented for safe driving. The second parameter is the acceleration brake rate (ABR) applied to the vehicle control module. This parameter is a real value in the range of $[-10, 10]$. A positive value means accelerating and a negative value means braking. Formally, the high-level discrete actions are defined as $A_d = \{a^{straight}, a^{left}, a^{right}\}$. Each discrete action $a \in A_d$ contains a set of continuous parameters $P_a = \{p_1^a, ..., p_n^a\}$. The overall hierarchical action space is defined as

$$\begin{aligned} A =&\{straight, p_{angle}^{straight}, p_{ABR}^{straight}\} \\ &\cup \{left, p_{angle}^{left}, p_{ABR}^{left}\} \\ &\cup \{right, p_{angle}^{right}, p_{ABR}^{right}\}). \end{aligned}$$

The three sets of actions represent three different types of driving behaviors. During training, the system will learn to choose from the three high-level action decisions and apply proper parameters specific to that action.

3.2. Actor-critic Based DRL Architecture

We develop our algorithm based on Deep Deterministic Policy Gradient (DDPG) [14], a deep reinforcement learning algorithm for continuous control. For better training stability, the actor-critic architecture with two neural networks is used. This architecture decouples the action evaluation and selection process into two separate deep neural networks: actor-network and critic-network, as shown in Figure 3. The actor-network μ, parameterized by θ^{μ}, takes as input state s and outputs hierarchical actions a along with its parameter p_a. The critic-network Q, parameterized by θ^{Q}, takes as input state s and hierarchical actions a along with its parameter p_a and outputs a scalar Q-Value $Q(s, a)$.

The hierarchical action a is represented as a vector of the probability of action choices p and the parameters P_a coupled to each discrete action. The discrete high-level action is chosen to be the output with maximum value among the choices of action probabilities. Then it is coupled with the corresponding parameters from the parameter outputs. Though the parameters of all actions are outputted, only the parameters of the chosen action are used. In this way, the actor-network simultaneously outputs which discrete action to execute and how to choose parameters for that action. The critic-network receives as input all the values of the output layer in the actor. We do not indicate which exact action is applied for execution or which parameters are associated with which action during training. In the back-propagation stage, the critic-network only provides gradients for the selected action and the corresponding parameters. This ensures that we update the policy only in the direction where we explore.

For training stability, we use a standalone target network [17] for the critic-network and the actor-network, which updates at a slower rate for a more stable iterative step in the Bellman equation. We also use replay memory [17] to store experiences, which can break the dependency between experiences. The exploration strategy has to deal with both discrete actions and continuous parameters. We use $\epsilon - greedy$ exploration to randomly explore among the given set of discrete actions. Then we sample uniformly for the continuous parameters. That means we choose a random discrete action $a \in A_d$ with probability ϵ and associate it with continuous parameters $\{p_1^a, p_2^a, ...\}$, each sampled uniformly over the range of its possible values.

3.3. Attention Mechanism for DRL

Human drivers can consider a series of historical observations to make driving decisions. They weight importance according to the time and location of the observations. To achieve this capability, we introduce an "Attention Mechanism" in our deep reinforcement learning algorithm, as shown in Figure 4. We first discuss how to add recurrence to the DDPG algorithm to include temporal dependencies

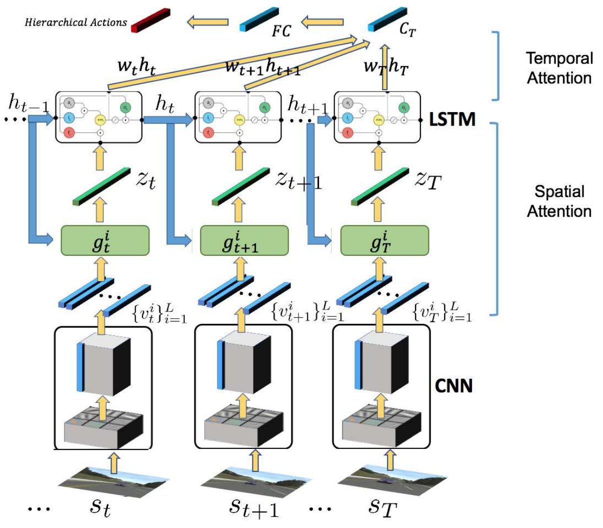

Figure 4: The architecture of the actor network in the Attention Deep Recurrent Deterministic Policy Gradient algorithm. We consider two kinds of attention: Temporal Attention and Spatial Attention. Temporal Attention learns to weight the importance of previous frames, while Spatial Attention learns the importance of different locations in the image.

in driving. Next, we introduce two streams of attention we consider: Temporal Attention and Spatial Attention. The Temporal Attention weights the past few frames with respect to importance to decide the current driving policy. The Spatial Attention detects the most important and relevant regions in the image for driving. The attention module is added to the actor network for better action selection. The critic network is set to be a fully convolutional network for action evaluation. Note that the attention architecture we propose can also apply to other neural-network-based sequential prediction problems, not just to learning for driving behaviors of self-driving cars. It can be used as an auxiliary task to enhance the performance of the original algorithm since the new information needed is limited in our method.

3.3.1 Recurrence in DDPG

The driving task often features incomplete and noisy perception information due to the partial observability from sensors. Given only one frame of input, the autonomous driver cannot collect sufficient environment information to generate the right driving behavior. For example, one ignored piece of information is the velocities of surrounding vehicles, which are crucial when making lane changes. This makes driving a Partially-Observable Markov Decision Process (POMDP).

To deal with partial observability, recurrence is introduced to better estimate the underlying environment state. As adding recurrence to DQN improved performance in Atari games [7], we introduce recurrence into DDPG for the driving task. The idea is to add an additional recurrent neural network module (RNN) to the output of the convolutional neural network in the actor network in DDPG. The RNN module processes the temporal information in the network instead of just having stacked historical observations as input. Also, a longer sequence of history information can be incorporated and considered due to the connectivity through time via RNN, which can help generate more complex driving strategies. In practice, we choose LSTM as the basic RNN structure, with the implementation based on [29]. We refer to this recurrent version of the DDPG algorithm as Deep Recurrent Deterministic Policy Gradient (DRDPG).

As compared to the original DDPG, DRDPG offers several advantages, including the ability to handle longer input sequences, exploration of temporal dependencies and better performance in case of partially observable experiences.

3.3.2 Temporal Attention

We wish to go further by deciding which frames matter most in past observations. We add temporal attention over the output of the LSTM layer in the DRDPG model, as shown in Figure 4. The temporal attention mechanism learns scalar weights for LSTM outputs at different time steps. The weight of each LSTM output w_i is defined as an inner product of the feature vector v_i and LSTM hidden vector h_i, followed by a softmax function to normalize the sum of weights to 1. By this definition, each learned weight is dependent on the previous time step's information and current state information.

$$w_{T+1-i} = Softmax(v_{T+1-i} \cdot h_{T+1-i}) \quad i = 1, ..., T$$

Then we compute the combined context vector C_T. The context vector C_T is a weighted sum of LSTM outputs through T time steps.

$$C_T = \sum_{i=1}^{T} (w_{T+1-i} h_{T+1-i})$$

The derived context vector C_T is passed by a fully connected layer FC before calculating the hierarchical actions of the actor network. The learned weights $\{w_{T+1-i}\}_{i=1}^{T}$ here can be interpreted as the importance of the LSTM output at a given frame. Therefore, the optimizing process can be seen as learning to choose which observations are relatively more important for learning the correct actions.

Temporal attention works in the sense that it explicitly considers the past T frames' LSTM output features for computing the action output, while this information is only passed implicitly by plain LSTM. By increasing the value of T, the model can consider a longer sequence of history frames and thus can make a better action choice.

3.3.3 Spatial Attention

Human drivers pay special attention to certain objects when performing specific tasks. For example, drivers care more about the direction of the road when traversing curvy roads and pay more attention to surrounding vehicles when making lane changes. This intuition can be enforced in the algorithm by adding importance to different locations in the image, which we call spatial attention.

With spatial attention, the network learns weights for different areas in an image. The context feature used by LSTM is a weighted sum combination of spatial features multiplied by the learned weights. According to how the combination is modeled, previous spatial attention models are divided into hard attention and soft attention [26]. Inspired by [22], we use a soft version of attention, which means learning a deterministic weighted context in the system.

The spatial attention, as shown in Figure 4, occurs after convolution layers and before recurrent layers. At time step t, suppose the convolutional layers produce a set of d feature maps with size $m \times n$. These feature maps can also be seen as a set of region vectors with length d : $\{v_t^i\}_{i=1}^L, v_t^i \in \Re^D, L = m \times n$. Each region vector corresponds to the features extracted by the CNN in a different image region. In the soft attention mechanism, we assume the context vector z_t is represented by a weighted sum of all-region vectors $\{v_t^i\}_{i=1}^L$.

$$z_t = \sum_{i=1}^{L} g_t^i \cdot v_t^i$$

The weights in this sum are chosen in proportion to the importance of this vector (i.e. the extracted feature in this image region), which is learned by the attention network g. The attention network g_t^i has region vector v_t^i and hidden state h_{t-1} produced by the LSTM layer as input and outputs the corresponding importance weight for the region vector v_t^i. The attention network g_t^i here is represented as a fully connected layer followed by a softmax function:

$$g_t^i = \text{Softmax}(w_v \cdot v_t^i + w_h \cdot h_{t-1}) \quad i = 1, ..., T$$

The context vector z_t is fed into the LSTM layer. The output of the LSTM layer is concatenated with action vector A_T and then used to compute the actions.

The attention network can be interpreted as a mask over the CNN feature maps, where it reweights the region features to get the most informative features for computing the actions. Thus, the spatial attention acquires the ability to select and focus on the more important regions when selecting the action. This helps to reduce the total number of parameters in the network for more efficient training and testing.

3.4. Reward Signal Design

Tabel 1 shows the five components that constitute the reward function. r_1, r_2, r_3 encourage the car to stay in the lane; r_4, r_5 encourage the car to run efficiently and make a proper lane change. Specifically, r_5 encourages the car to overtake if the front vehicle is within a distance of 100 meters. Here x means the distance to the front vehicle in the same lane. If no vehicle is found, then x has a default value of 100 meters.

Rewards Term	Reward Functions
Road Alignment	$r_1 = \cos\theta - sin\theta$
Lane Centering	$r_2 = -\lvert d \rvert$
Out-of-Bound Penalty	$r_3 = -\mathbb{1}\{OutofBoundary\}$
Prefer Speed Limit	$r_4 = \begin{cases} v & v \leq 35\,m/s \\ 70 - v & v > 35\,m/s \end{cases}$
Encourage Overtake	$r_5 = -\max(0, 100 - x)$

Table 1: Terms in the reward function. Where θ is the yaw angle w.r.t. the road direction; d is the offset to the lane center; v is the speed; x is the distance to the leading vehicle.

The overall reward function is a linear combination of terms in Table 1 with assigned weights w: $R = \sum_{i=1}^{5} w_i r_i$. Here we first normalize the rewards to the range $(0, 1)$ and then search on the different weighting coefficients to find the best combination that generates a good result.

4. Experiments

The algorithm is evaluated in the open source car simulation environment TORCS. To ensure the model is not limited to a particular road configuration or car type, five tracks and ten types of cars are used to generate our evaluation sets. Each track has two to three lanes with various brightness levels of the sky. The selected tracks have different surrounding environments and route shapes so that the training agent can confront all circumstances of driving, which ensures the complexity of the driving scenarios. For example, in the Corkscrew scenario, the road has more curves and fewer straight sections, and there is a sharp high-speed corner. To successfully traverse the selected tracks, the car has to learn various skills like U-turn, hill climbing, overtaking and throttling before a large-angle turn. All these challenges are posed to the training agent with no human supervision.

We add traffic cars in different locations on every track in each trial of the training to ensure the density and complexity of the traffic. The traffic cars' behaviors are controlled by the internal AI of the TORCS environment, which is designed to mimic human behaviors when driving. We add

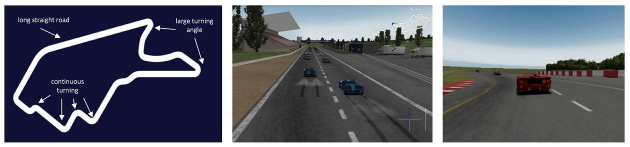

Figure 5: One example of the five tracks used for training. From left to right: the map of the example track *Street-1*, image top view when starting a new episode, a screenshot of the front view camera during training.

diversity to the traffic cars' behavior by changing the internal parameters like average speed, acceleration limit and chances of lane changes in the car simulation AI. This will change the car's driving style, e.g., being more aggressive when turning. In this way, we would like to mimic the real traffic patterns on the road. We set the speed limit of traffic cars to 30 m/s and set the speed limit of our ego vehicle to 35 m/s. We do this to encourage the overtaking and lane changing behavior of our vehicle. An example of the environment used is shown in Figure 5.

We use images of the car front view as perception sensor input. The raw images are collected at 5 HZ frequency to avoid feeding near-duplicate images into the model. The images are then down-sampled to 320×240 from the original video frames. Other measurements used for reward design like car direction, speed direction, distance to lane boundaries, etc. are collected along with the images. For each model tested, we train the agent on all five tracks in TORCS for a fixed 5-hour period. This allows the agent to collect around 100,000 frames for each scenario. To avoid overfitting in one track and add diversity in training, we randomized the sequence of tracks encountered in training. For each model, the training starts an episode by automatically selecting one of the five tracks. Every time the car goes outside of the road boundary or experiences a collision that causes failure, the episode will end and the program will restart to generate a new episode. In sum, the agent updates its network while testing it in the environment for 15 hours before we evaluate its performance.

We use CNN networks for both the actor and critic networks. The feature extraction from images is done by a standard AlexNet [11] with the first five layers. We train the LSTM layers sequentially on the video sets from simulation with an unrolling size of 8 frames during training. Specifically, the LSTM we use has 64 hidden units. For optimization, we train our model using stochastic gradient descent (SGD) with an initial learning rate of 10^{-3}, momentum of 0.99 and a batch size of 10. The learning rate is decayed by half whenever training loss plateaus. We run our training in parallel on 4 Nvidia Titan X GPUs.

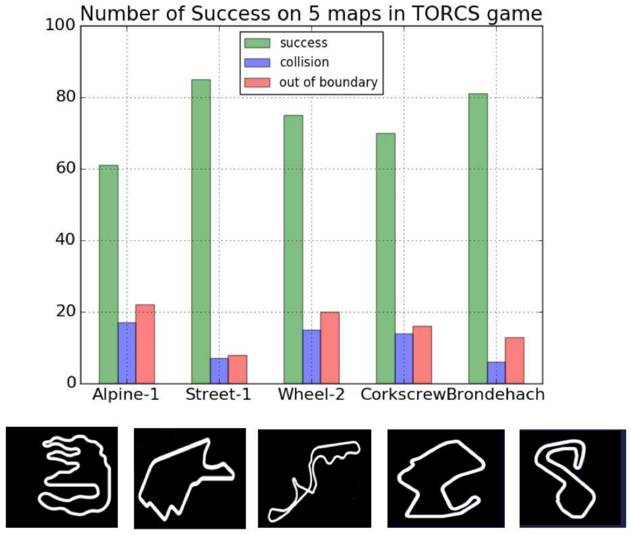

Figure 6: Final DRDPG model with hierarchical actions, Spatial and Temporal attention tested on different trials in TORCS game. We obtain the result of each map by running 100 episodes.

5. Evaluation

We first evaluate the success rate of our final model with both temporal and spatial attention on all five tracks in TORCS. We calculate the success rate by testing the agent's ability to drive along the entire track and see if it can successfully finish a loop. As shown in Figure 6, the model shows overall good performance in all five trials with a success rate between 60 and 80 percent to complete a runway circle on the map. The failure cases mainly come from collision or out-of-bounds cases, about half of each. One example of out-of-boundary comes from the case in which a high-speed car fails to turn at a sharp curve since the agent does not learn to decelerate ahead of time. Another case occurs when a front vehicle blocks the view of a turn, and the agent doesn't have enough time to react to a turn that suddenly appears. For the collision issue, most collisions happen when trying to overtake a vehicle or being forced to do a lane change. Some of these cases are difficult: occlusions can arise when entering a corner or the front vehicle suddenly slows down due to the road geometry.

We further investigate the performance of our method compared to the original DDPG baseline, as shown in Figure 7. All models are evaluated based on average values over 100 episodes after 15 hours of training. We first compare the version which has hierarchical action space (the "hier" configuration in Figure 7) with the original DDPG algorithm. We observe an obvious boost of 2 m/s in average speed and 3 more lane change behaviors after we apply our hierarchical action space. The results show that the original DDPG algorithm tends to drive more conservatively and stays behind other vehicles more often without trying

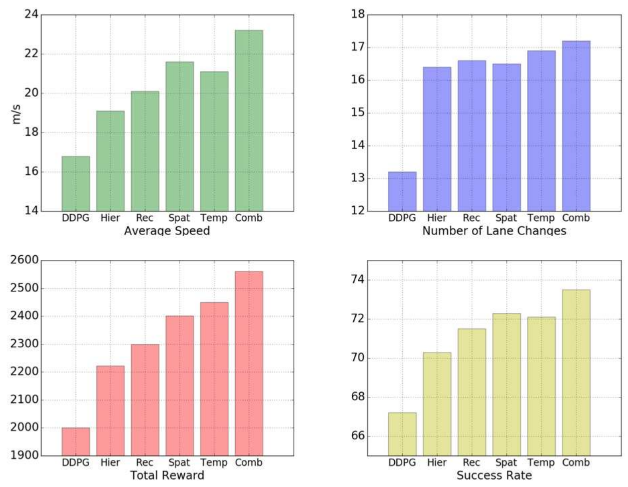

Figure 7: A comparison of the performance improvements with respect to the baseline (DDPG), obtained by introducing the hierarchical actions (Hier), recurrence (Rec), spatial attention (Spat) and temporal attention (Temp) from left to right. The final combined model (Comb) applies all of above. The performance evaluation is based on average speed, the number of lane changes, total reward during an episode and percentage of successful episodes.

to overtake. In comparison, our model with hierarchical actions is designed to learn separate policies for left lane change and right lane change, so it tends to act more aggressively and maintain higher speed to make lane changes whenever possible. Although the method encourages overtake behavior, it still guarantees safety at the same time, as the success rate of finishing a loop increases by 3 percent. This is due to the fact that more overtake behaviors are able to avoid getting too close to or colliding with the leading vehicle. Our method achieves higher rewards and fewer collisions with other vehicles, thus resulting in a higher success rate for a complete episode compared to the original DDPG algorithm.

We next show the improvements of adding recurrence (”rec”), temporal attention (”temp”) and spatial attention (”spat”) to the DDPG algorithm with hierarchical actions in Figure 7. We find that adding recurrence can actually speed up the training curve and achieve higher rewards and performance as evaluated by the success rate. It also results in higher average speed due to smoother and more stable driving behavior. For the attention mechanism, we can observe that both temporal and spatial attention improve the metrics compared to the recurrence model. This can be attributed to the better utilization of the images as input and finally results in safer and more stable lane change behavior. With the final combined model (”comb”), we see an increase in the average speed from 16.7 to 23.2 m/s and the number of lane changes increase from 13.2 to 17.2 while also improving the success rate from 67.2% to 73.5% compared to the original DDPG model.

6. Visualization

Figure 8 and 9 give two examples to illustrate the effect of attention mechanism in turning and overtaking scenarios. In the turning scenario (Figure 8), the agent has learned to change lane in order to perform a more efficient turn at a sharp right curve. At the top-left corner in each figure, different weights w_i in temporal attention are assigned to the input images. Notice that although the weights generally increase through time, there are exceptions where the agent thinks a certain situation that needs special attention. For example, in frame 5, the car is just crossing the lane marker on the road, which could be more dangerous and need greater attention. In this case, a single last frame helps less than a sequence of frames. By looking back into the past at the weighted sum of features of several frames before, the agent can determine the best trajectory for turning at the corner.

In the overtaking scenario depicted in Figure 9, the regions of lane end and front vehicle are highlighted by the Spatial Attention. The attention model gives large weight to the region where the neighboring vehicle appears, which helps the agent figure out when and how to launch the proper lane change behavior. As the result, the agent has learned to approach the front vehicle, slow down to keep distance, perform a lane change and then speed up to overtake the front vehicle. By learning a mask over the input image (i.e., the CNN features correspond to the input image), the agent can extract the relevant context to the task and learn the proper behavior more efficiently.

7. Conclusion

We introduce an attention-based hierarchical deep reinforcement learning algorithm for learning lane change behaviors in dense traffic with an end-to-end trainable architecture. The proposed hierarchical action space for learning driving behaviors can generate sub-policies for lane change behaviors in addition to the lane following behavior. This model simplifies the work of deliberately designing a sophisticated lane change maneuver and introduces a data-oriented approach that can learn the lane change policy through trial and error. We also investigate how an attention mechanism can help in the task of driving with deep reinforcement learning. The two streams of temporal attention and spatial attention are proven in experiments to boost the performance in deep reinforcement learning. The attention model also helps to explain what is learned in the driving model.

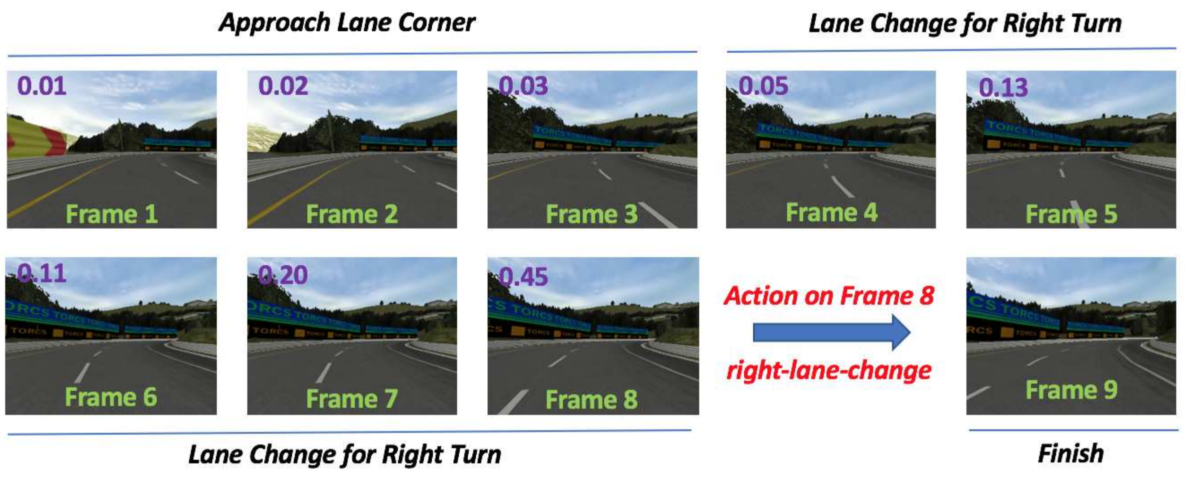

Figure 8: A turning scenario to illustrate Temporal Attention. In this scenario, the agent has learned to do a right lane change for a more efficient right turn. The number at the top left of each image is the weight assigned to that image frame for temporal attention (higher weight indicates more importance).

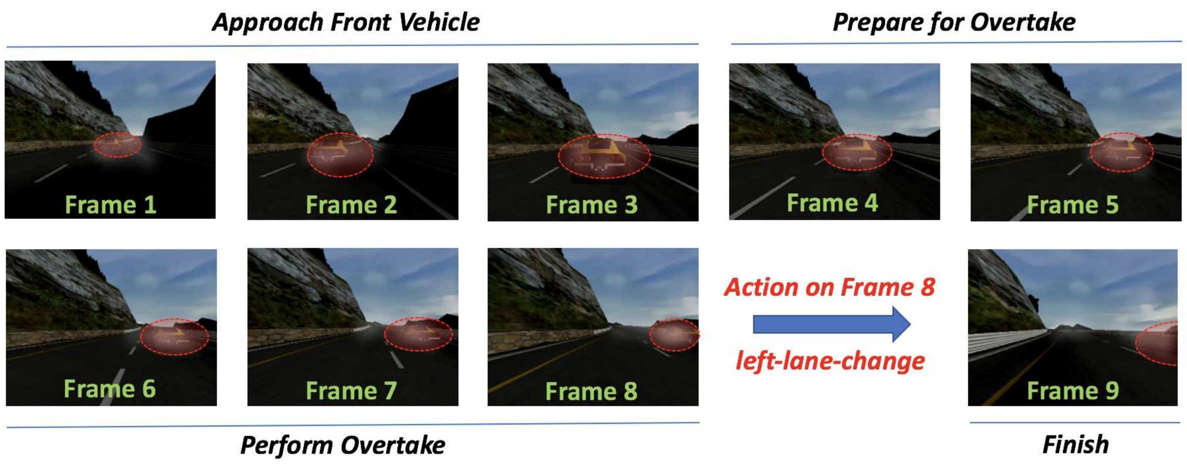

Figure 9: An overtaking scenario to illustrate Spatial Attention. A mask over the input image is learned by the Spatial Attention. Brighter colors indicate higher weights assigned to that region. The weights are smoothed with a Gaussian kernel for visualization.

References

[1] Mohammed Al-Qizwini, Iman Barjasteh, Hothaifa Al-Qassab, and Hayder Radha. Deep learning algorithm for autonomous driving using googlenet. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 89–96. IEEE, 2017. 2

[2] Christopher R Baker and John M Dolan. Traffic interaction in the urban challenge: Putting boss on its best behavior. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1752–1758. IEEE, 2008. 1, 2

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 2

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 2

[5] Shitao Chen, Songyi Zhang, Jinghao Shang, Badong Chen, and Nanning Zheng. Brain-inspired cognitive model with attention for self-driving cars. *IEEE Transactions on Cognitive and Developmental Systems*, 2017. 2

[6] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1856–1860. IEEE, 2017. 2

[7] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015. 4

[8] Matthew Hausknecht and Peter Stone. Deep reinforce-

ment learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015. 2

[9] Maximilian Jaritz, Raoul de Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. *arXiv preprint arXiv:1807.02371*, 2018. 2

[10] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Int. Conf. Comput. Vis.(ICCV)*, pages 2961–2969, 2017. 2

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 6

[12] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016. 2

[13] Sang-Hyun Lee and Seung-Woo Seo. A learning-based framework for handling dilemmas in urban automated driving. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1436–1442. IEEE, 2017. 2

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1, 3

[15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015. 2

[16] Stefanie Manzinger, Marion Leibold, and Matthias Althoff. Driving strategy selection for cooperative vehicles using maneuver templates. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 647–654. IEEE, 2017. 1, 2

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 1, 3

[18] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 2

[19] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *American Control Conference (ACC), 2017*, pages 4914–4919. IEEE, 2017. 2

[20] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016. 1

[21] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016. 2

[22] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*, 2015. 5

[23] Junqing Wei, Jarrod M Snider, Tianyu Gu, John M Dolan, and Bakhtiar Litkouhi. A behavioral planning framework for autonomous driving. *Intelligent Vehicles Symposium (IV), 2014 IEEE*, 2014. 1

[24] Peter Wolf, Christian Hubschneider, Michael Weber, André Bauer, Jonathan Härtl, Fabian Dürr, and J Marius Zöllner. Learning how to drive in a real world simulation with deep q-networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 244–250. IEEE, 2017. 1

[25] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint*, 2017. 2

[26] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015. 2, 5

[27] Zhaoyang Yang, Kathryn Merrick, Lianwen Jin, and Hussein A Abbass. Hierarchical deep reinforcement learning for continuous action control. *IEEE Transactions on Neural Networks and Learning Systems*, 2018. 2

[28] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. *arXiv preprint arXiv:1801.06734*, 2018. 2

[29] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. 4