2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Deep Hierarchical Reinforcement Learning for Autonomous Driving with Distinct Behaviors

Jianyu Chen*, Zining Wang* and Masayoshi Tomizuka

Abstract— **Deep reinforcement learning has achieved great progress recently in domains such as learning to play Atari games from raw pixel input. The model-free characteristics of reinforcement learning free us from hand-encoding complex policies. However, for real world tasks such as autonomous driving, there are some complex sequential decision making processes that contain distinct behaviors. Due to the delayed rewards and the averaged gradient, it is pretty difficult for a flat deep reinforcement learning algorithm to learn a good policy.**

In this paper, we design a hierarchical neural network policy and propose a hierarchical policy gradient method to train the network with the semi markov decision process (SMDP) temporal abstraction formulation. We apply this method to a traffic light passing scenario in autonomous driving, where the vehicle has two distinct behaviors (e.g., pass and stop) and its primitive actions (e.g., acceleration) should follow the corresponding behavior. We show via simulation that our method is able to select correct decision and acts appropriately when the traffic light turns yellow. On the contrary, the flat reinforcement learning algorithm is not able to achieve a good performance and exhibits a large variance. Furthermore, the trained neural network modules are reusable in the future to cover more scenarios.

I. Introduction

Recent progress in deep learning is quite inspiring. Several breakthroughs happened in multiple domains, such as computer vision [1] and natural language processing [2]. When combined with reinforcement learning [3], deep reinforcement learning showed its power on tackling complex decision making and planning problems. Exciting successes are achieved in game playing [4], [5], robotics [6], [7] and self-driving [8]. The wonderful characteristics of deep neural network such as its expressive representation are well exploited, making it capable to capture extremely complex policies for difficult tasks. Its benefit is huge, e.g, it allows robots to learn policies automatically through experience. Furthermore, it may potentially avoid developing tedious hand-encoded components for perception and control.

Despite the exciting achievements, there are still many tasks that deep reinforcement learning is not able to solve well. One main reason is the delayed rewards. Unlike supervised learning, reinforcement learning receives only very weak feedbacks. In the reinforcement learning process, the agent is only told how good it is acting (rewards) instead of what it should do (state-action mapping). This problem becomes more serious when there exists distinct behaviors whose corresponding actions are quite different from each other. For example, in most urban driving scenarios such as intersection or roundabout, there are basically two distinct behaviors, pass or yield. In this kind of tasks, we can only get a significant reward after the behavior is finished (e.g., finished passing the intersection). Furthermore, since the deep neural network tends to be a smooth mapping, it is very hard to learn a policy that can act distinct behaviors. Generally speaking, The flat deep reinforcement learning is often not able to deal with such tasks.

There are some applications of deep reinforcement learning on autonomous driving [8]–[10]. However, they are only applied in simple tasks such as lane keeping, without complex urban scenarios and distinct behaviors. Imitation learning based methods [11], [12] can learn policies to handle more complex tasks. However, they require a fairly good guidance (either by human or by an optimal controller), which might not be accessible all the time.

Hierarchical reinforcement learning (HRL) [13] introduces hierarchical structures into the policy. The key point of hierarchical reinforcement learning is to add temporal abstraction and intrinsic motivation. There are several great works in this field, such as MAXQ [14], option-critic [15], FeUdal network [16], modulated locomotor [17] and MLSH [18]. The success of these works on tackling difficult tasks proves that the method of temporal abstraction can significantly improve the performance of deep reinforcement learning. Using hierarchical policy can also benefit transfer learning, because the modularized policies can be easily reused in new tasks. Moreover, the decoupled policy modules enable us to get more interpretation of how the policy works.

In this paper, we take advantage of the compositional characteristics of hierarchical reinforcement learning to separate these different behaviors into different policies. Then we train an additional master policy that can choose which behavior to execute with temporal abstraction. A policy gradient based hierarchical reinforcement learning algorithm is proposed to implement the method. The method is evaluated in a traffic light passing scenario simulation. The learned policy is analyzed with explicit interpretation, and the policy submodules can be reused in other scenarios.

The remainder of the paper is organized as follows. Section II describes the structure of the deep hierarchical reinforcement learning, how the hierarchical policy is executed and how it is trained with policy gradient. Section III explains the details of implementing the method such as data collection and normalization. Section IV shows a case study

*indicates equal contribution.

J. Chen, Z. Wang and M. Tomizuka are with the Department of Mechanical Engineering, University of California, Berkeley, CA 94720 USA (e-mail: jianyuchen@berkeley.edu, wangzining@berkeley.edu, tomizuka@berkeley.edu).

978-1-5386-4451-5/18/$31.00 ©2018 IEEE

applying the proposed method and analyzes its performance. Finally, Section V concludes the work.

II. Deep Hierarchical Reinforcement Learning with Policy Gradient

A. Temporal Abstraction

Temporal abstraction is the core of hierarchical reinforcement learning, which allows us to explore the world more efficiently. In markov decision process (MDP), we have state abstraction which is often known as function approximation. Temporal abstraction is the abstraction of action, which together with state abstraction significantly improves generalization and scales up the reinforcement learning algorithms.

The concept of temporal abstraction has been developed early. Before hierarchical reinforcement learning, it was exploited in hybrid system research [19]. There are several temporal abstraction methods, such as the option framework [20] and the manager-worker framework [21]. In this paper our problem formulation is similar to the method of MAXQ [14], which is based on the option framework and decomposes the value function of an MDP into a combination of value functions of smaller MDPs.

The semi markov decision process (SMDP) [22] is used in this paper to construct a mathematical formulation of temporal abstraction. SMDP is an abstract form of markov decision process (MDP). The key idea of SMDP is that in contrary to MDP whose action can only be performed in a single time step, the action in the SMDP framework can persist over a variable period of time. As shown in Fig.1, the state trajectory of MDP is made of small, discrete-time transitions, whereas that of SMDP is made of larger, continuous-time transitions.

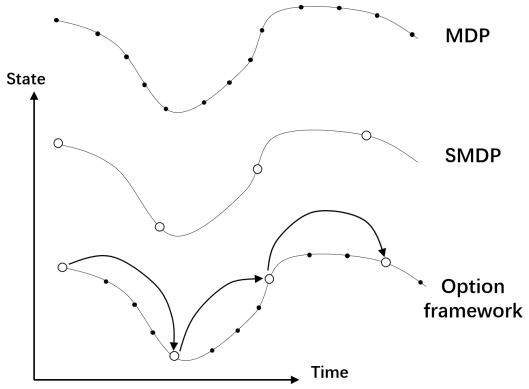

Fig. 1: The state trajectory of MDP and SMDP

An SMDP problem setup has the same elements with MDP $\langle S, A, P, R, P_0 \rangle$, but some elements have different meanings:

- S: The set of states of the environment.
- A: The set of actions. Here each module in the hierarchical policy has different set of actions. For the primitive modules, the action set contains the actions that can be applied directly to the environment. For a higher-level module, the action is an option, which defines an index to specify which lower module to select.
- P: The transition model. When an action $a \in A$ is performed, the environment makes a probabilistic transition from the current state s and time step k to a resulting state s' and time step $k + N$ according to the probability distribution $P(s', N | s, a)$.
- R: When an action a is performed, the agent will receive a reward r whose expected value is $R(s', N | s, a)$. And the reward functions are different for different modules.
- P_0: The initial state distribution.

B. Hierarchical policy execution

The policy design in this paper is hierarchical and networked. Fig.2 shows the structure of an example hierarchical policy network. There are five neural network modules connected as a net, where each lower level module can be selected and called by multiple higher level modules. Only the lowest level modules, a.k.a primitive controllers, have direct access to the environment. Output produced by higher level modules is considered as an option and command to its lower level modules.

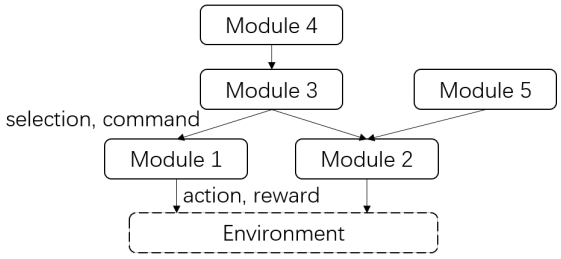

Fig. 2: The hierarchical policy structure

When executing this kind of hierarchical policy network, a mechanism for transition termination is needed. For a primitive controller, it is just the same as the original MDP. The transition is terminated immediately after applying the action to the environment, and a corresponding resulting state and reward will be given. However, for a higher level module, a terminating state needs to be specified, e.g., it will terminate at state s with probability $\beta(s)$. Moreover, there should be a pointer that specifies which module is being executed now. When it terminates, the pointer should return to point to its higher-level module. The pseudo-code of this process is shown in Algorithm 1.

C. Hierarchical policy gradient

For the training process, instead of using Q-learning to train the hierarchical policy, we use policy gradient. There are several benefits of using the policy gradient method. First, when applying deep neural networks, policy gradient has more rigorous theoretical foundation than deep Q-learning. Although the tabular Q-learning, as is used in MAXQ, is proved to converge to the global optimum, the deep Q-learning is not supposed to converge. Second, a lot of scenarios need to be formulated as a partially observable markov decision process (POMDP) problem instead of MDP.

Result: Execute one step of the root module and save the (s, a, r) tuples of trajectories for each module individually
s_t is the state of time t;
$M_0, M_1, \cdots, M_N$ are all the modules in the hierarchical policy. Each module stores its lower module, e.g, M_i.lowermodule $= M_j$;
K_t is a stack at time t, which stores modules in the order of execution;
Let $t = 0$; K_t = empty stack;
Append the root module M_0 to K_t;
repeat
  repeat
    Get action from K_t[-1];
    Save the current (s, a, r) tuple to K_t[-1];
    Append $K_t\,[-1]$.lowermodule to K_t;
  until $K_t\,[-1]$.*lowermodule!=None*;
  Get action from K_t[-1];
  Get the next state and reward;
  Save the current (s, a, r) tuple to K_t[-1];
  for i *in range(len(K_t)-1)* **do**
    Add intrinsic reward to $K_t\,[i]$;
  end
  for i *in range(len(K_t))* **do**
    if $K_t\,[i]$ *terminates* **then**
      K_t.pop()
    end
  end
until K_t *is empty*;

Algorithm 1: The Hierarchical Policy Execution Process

For example, in a multi-agent system, the internal states of other agents are not directly accessible. For these problems, the Q-learning method which depends on the markovian assumption does not work well. In contrast, the policy gradient method does not depend on the markovian assumption.

During the training phase, at each time the policy gradient for one specific module needs to be calculated based on the intrinsic rewards. The whole policy needs to be trained from the lowest level modules up to higher levels. The training of a higher level module can succeed only when its lower modules already achieve relatively good performance.

III. Implementation Details

This section introduces the learning process designed for the traffic light passing problem. Section III-B introduces a slightly modified policy gradient method to deal with the large range of initial conditions. Section III-A introduces the simulation setup and input-output configuration for hierarchical reinforcement learning.

A. Efficient Data Collection

The training data is collected by a simulator of the vehicle dynamics. The states of the system consist of $\{v, s, t_{red}\}$ where v is the velocity of the vehicle and s is the distance from the vehicle to the crossing line. t_{red} is the time for the traffic light to turn to red. The parameters of the reward function consist of $ind_{pass}, v_f, t_{pass}$ where ind_{pass} is the indicator from upper controller determining whether to pass the light or stop before the line. For different ind_{pass} values there are different reward functions. v_f is the reference velocity for the vehicle to track. t_{pass} is the time limit to pass the light generated by the upper controller. The tuple $(v, s, ind_{pass}, t_{pass}, v_f)$ is fed to the primitive controller as input and the policy network outputs acceleration command a bounded by a sigmoid function. When $ind_{pass} = 0$ which means that the vehicle should stop, the t_{pass} is set to the time for the vehicle to stop. The cost function for the primitive controller is designed as

$$r = w_1(v - v_f)^2 + w_2 a^2 + w_3(0 - v)_+ + w_4 r_{\text{terminate}} \quad (1)$$

which includes tracking penalty, acceleration penalty, negative velocity penalty and termination penalty $r_{\text{terminate}}$ with weights w_1, w_2, w_3 and w_4. The termination penalty is only applied when the primitive controller meets the termination criterion.

$$r_{\text{terminate}} = \begin{cases} (0 - s)_+, & \text{if } ind_{pass} = 1 \\ (s - 0)_+ + (\delta - s)_+, & \text{if } ind_{pass} = 0 \\ 0, & \text{if not last time step} \end{cases} \quad (2)$$

For the upper controller, the reward function formulations is as given by (1), but the termination term (2) is different:

$$r_{\text{terminate}} = \begin{cases} \frac{v}{s}, & if\ s \geq 0 \\ 0, & if\ s < 0 \end{cases} \quad (3)$$

which means that it has a termination penalty for high speed and failure of crossing when the light turns red. In this case, the vehicle is likely to have a red light violation. The upper controller reward function is also the overall reward function for the non-hierarchical policy network.

It is found in simulation that the speed is the bottleneck for the efficiency of training because the policy networks are usually simple and small. The simulation is modified to allow parallel batch experiments. Note due to the termination criterion, the total steps of each iteration may vary. For a batch of experiments, only the non-terminated states are updated and stored. The simulation step is determined by the longest experiment so the number of experiments updated is changing in one batch. The training is observed to be about 20 times faster with an experiment batch size of 20 compared to training with 20 experiments one by one. Training one policy network takes 10 to 30 minutes with efficient data collection.

B. Task Dependent Policy Gradient

1) Advantage Normalization: Normalizing the rewards to advantages, which has zero mean and unit variance, is very common in policy gradient. It is natural to think that for a batch of trajectories collected, those with different reward functions should be normalized separately. Otherwise the reward function with higher mean would overwhelm the one with lower mean and ruin the training of the lower one.

Normalizing the advantage with respect to different rewards is found to be insufficient when initial states vary in

a large range. In the traffic light passing scenario considered here, there are various kinds distributions of rewards with respect to the initial states. Therefore it is desirable to normalize advantage with respect to both initial states v_0, s_0, t_0 and reward functions indexed by ind_{pass}.

2) Hyper-parameter Tuning for Gradient update: Here the reward weights are imbalanced where w_3 and w_3 are much larger as they are supposed to approximate hard constraints. It is found using the Adam optimizer setting results in a lot of oscillation during training. The parameter for Adam is set to $\beta_1 = 0.8, \beta_2 = 0.9, \epsilon = 1$.

IV. Experiments

A. Experiment Setup

We apply our method to a traffic light passing scenario for autonomous driving. As shown in Fig.3, there is a pedestrian crosswalk along with a traffic light in the middle of the track. The vehicle will be initialized with a random distance s_0 to the crosswalk with a random initial velocity v_0. The traffic light is initialized as "green", but it will turn to "yellow" in a random time t_0. And the "yellow" light will last for 3 seconds. We believe this is a quite challenging scenario for most new drivers, as they often don't know what to do when the traffic light suddenly turns to "yellow" and thus often results in suboptimal performance.

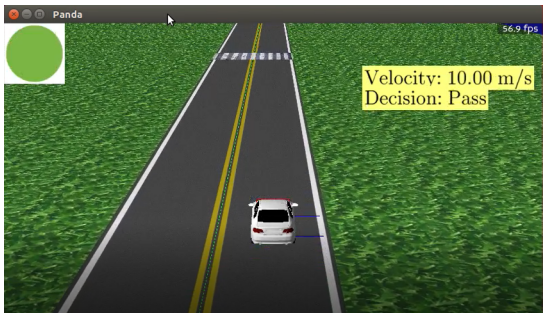

Fig. 3: The traffic light scenario for autonomous driving

For a flat deep reinforcement policy, we just need one single policy that maps the current state directly to the actions applied to the environment. Here it is a mapping from the state to the acceleration and deceleration for each time step.

For our hierarchical policy, we designed a structure shown in Fig.4. The root module "Traffic light" has three lower level modules "Green", "Yellow" and "Red", and their state transition processes depend on the traffic light time. When the light is green and red, the policy is pretty trivial so we will only concentrate on the yellow case. Here the "Yellow" module will decide either to stop or to pass. And the lower level "Stop" module will calculate how much to decelerate to stop before the crosswalk, while the "Pass" module will calculate how much to accelerate to pass before the traffic light turns red.

The hierarchical structure is tested on a traffic light passing problem in simulation. Primitive controllers are trained individually for their own tasks first. Then the upper controller is assembled together with lower level ones and

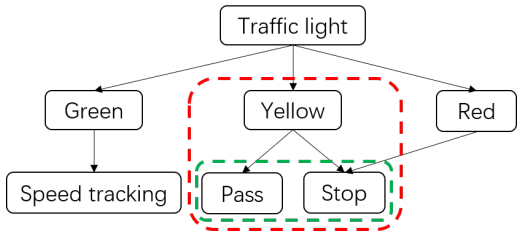

Fig. 4: The hierarchical policy of the traffic light scenario

trained. This section shows their individual performance and combined performance. The effectiveness of using hierarchy is demonstrated by comparing the result with a flat structured baseline.

B. Evaluation Metric

In addition to the average return, another metric is calculated to evaluate the absolute performance of the controller, called the violation. Due to the randomness and wide range of initial conditions, there are scenarios when the vehicle is neither able to stop nor able to pass the traffic light. For example, the initial velocity is high (like $15m/s$) and distance is small (like 10m) but the light is going to be red in a very short period (like 0.1s). The vehicle cannot avoid violating the traffic rule, resulting in a very low reward that makes the average return dominated by the large penalty.

The violation has two terms: absolute violation and relative violation. The absolute violation is the termination penalty $r_{\text{terminate}}$. The relative violation is the gap between the violation of the current network and the violation produced by the optimal controller (like pushing the braking pedal to the bottom all the time to minimize the violation). The relative violation shows the gap between the policy network and the optimal controller when violation is unavoidable.

C. Hyper-parameter Details

1) Simulation: The initial condition uniformly chooses initial velocity $v_0 \in [8, 12]m/s$, initial distance $s_0 \in [5, 60]m$ and time to red $t_{red} \in \{3s, 10s\}$. The reference speed is fixed to $10m/s$. The system is discretized with a sampling rate of 50Hz. Weights for the primitive controller is set to be $w_1 = 2, w_2 = 1, w_3 = 50$ and $w_4 = 20000$. Violations get larger weights w_3 and w_4 so as to approximate the hard constraint.

2) Primitive Controller: The the primitive controller has 6 fully connected (FC) layers with 32 hidden units in each layer, regardless of its task. The acceleration $a \in [-5, 3]m/s^2$ is bounded by a sigmoid function added to the last layer of the network. Each update iteration generates 10 different initial conditions. For each initial condition a number of 20 paths is simulated. So there are 200 paths in total for each iteration. The advantages are calculated as stated in III-B. The discount for calculating the Q-value is set to be 0.999, which coincides with the discount rate of the upper controller which decays to 0.9 every 2 seconds. The learning rate is 0.005. The total training iteration is 2000.

D. Primitive Controller Evaluation

First it is shown that the primitive controller works well individually for each task. Fig.5 shows the average return of each primitive controller. They converge to relatively good performance. Fig.6 shows the absolute and relative violations by each primitive controller, and the violations are around zero.

Fig. 5: Average returns of the primitive controllers.

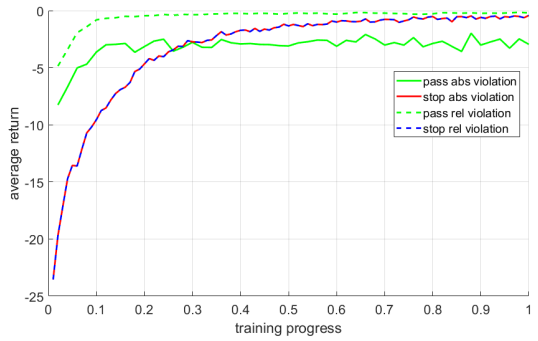

Fig. 6: Violations of the primitive controllers.

E. Upper Controller Evaluation

The upper controller can be evaluated only if some primitive controllers are given. Here the trained primitive controllers in Section IV-D are used following the hierarchical structure. Another non-hierarchical policy network is trained simultaneously as the baseline. Note here for the overall controller the training iteration is different from the general setting in Section IV-C.2. For the hierarchical structure, the fine-tuning only takes 150 iterations. For the non-hierarchical structure, the training takes 5000 iterations and the performance is still significantly lower than the hierarchical structure shown in Figure 7.

Besides the average return plot, another 3D plot is shown in Figure 8 which is the point cloud of state-action pairs during the test. In the test case 500 initial states are sampled randomly and 4 paths are simulated for each initial state. Two dimensions of the observations, namely velocity v and distance s, and the acceleration a from action are chosen to form state-action pairs for plot. There are totally about 500K points in the point cloud representing all state-action pairs in all trajectories.

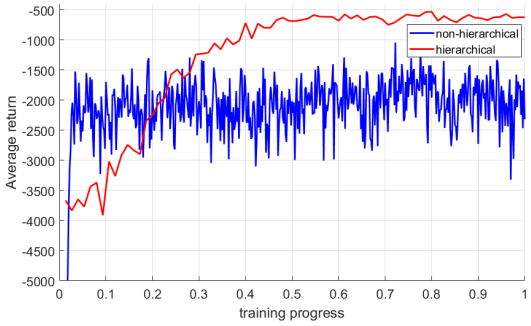

Fig. 7: Average returns of the overall controllers.

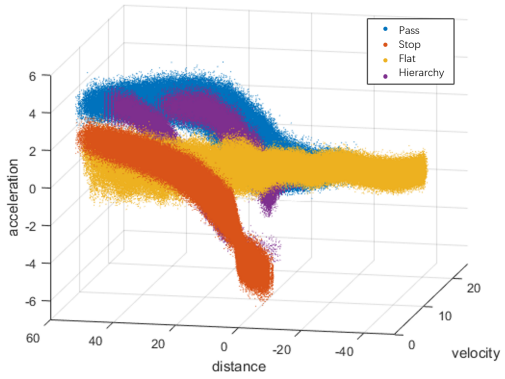

Fig. 8: State-action point cloud of controllers. The red part represents the primitive controller for “Stop”, the blue part represents the primitive controller for “Pass”. The yellow part represents the non-hierarchical policy network and the purple part represents the hierarchical policy network.

In Fig.8, the red part represents the primitive controller for stop and the blue part represents the primitive controller for pass. For the hierarchical controller colored by purple, the point cloud clearly splits into two parts with a significant gap. However, for the non-hierarchical controller colored by yellow, the point cloud is in the middle where it has a long “tail” resulting in passing the line for a long distance. This means the non-hierarchical controller has a lot of violations when the traffic light turns red quickly. It is optimized to an aggressive local optimum where it always tries to pass the yellow light.

F. Simulation Results

We simulate and visualize the policy with a self-built python simulator for autonomous driving, which is built upon the panda3d API. In the visualization window as show in Fig.3, the left upside shows the state of the traffic light, and on the right side we display the current vehicle velocity and decision. A time series snapshot is shown in Fig.9.

In the figure, the top four pictures show how the vehicle decides to stop. At the beginning, the light is green and the vehicle maintains a constant speed; then the traffic light turns to yellow and there is still a long distance to the crosswalk, and the higher level policy decides to stop;

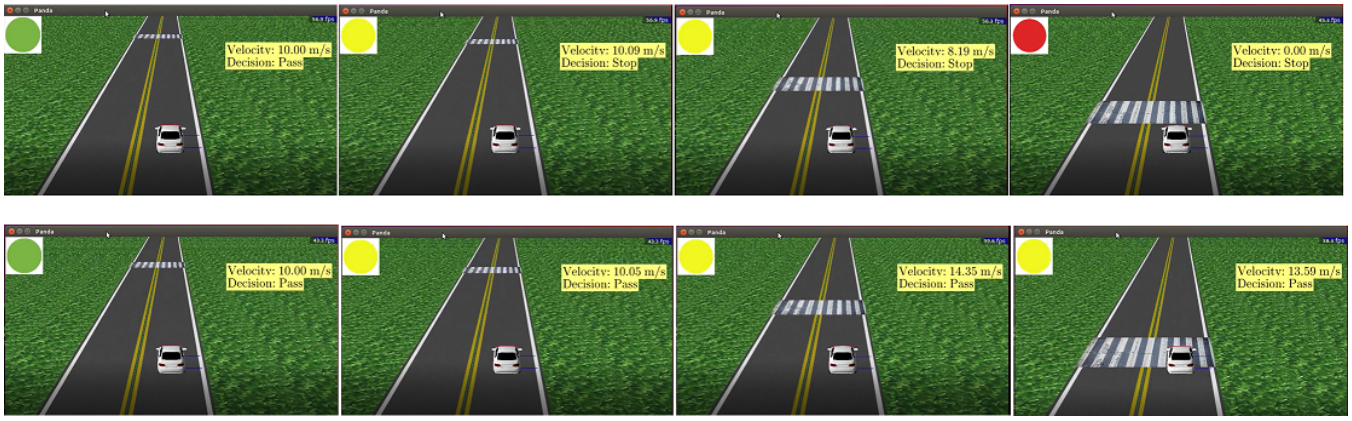

Fig. 9: The simulation results visualization

Receiving this command, the lower level policy generates appropriate deceleration to slow down the vehicle; Finally, the vehicle stops right in front of the crosswalk.

The bottom four pictures show how the vehicle decides to pass. At the beginning, the light is green and the vehicle maintains a constant speed; then the traffic light turns to yellow and the distance seems possible for the vehicle to pass, and the higher level policy decides to pass; Receiving this command, the lower level policy generates appropriate acceleration to speed up the vehicle; Finally, the vehicle passes the crosswalk safely before the traffic light turns to red.

V. Conclusion

In the traffic light passing problem, the flat deep reinforcement learning is not able to solve the whole task which has distinct behaviors. The hierarchical reinforcement learning structure designed based on the SMDP model breaks down the whole task into several simpler and self-consistent modules. Each module is proved to have good performance on its subtask and the assembled hierarchical policy is able to address the traffic light passing problem. We showed the hierarchical structure can handle the delayed reward and distinct behaviors problem.

Figure 8 shows two distinct optimal behaviors side-to-side. When trained with one non-hierarchical controller using policy gradient, the network does not capture the two-manifold structure and converges to a local optimum which is a single manifold. The proposed method splits the policy into two manifold of state-action pairs.

References

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[2] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. Journal of Machine Learning Research, 12(Aug), 2493-2537.

[3] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[5] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

[6] Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39), 1-40.

[7] Levine, S., & Koltun, V. (2013). Guided policy search. In Proceedings of the 30th International Conference on Machine Learning (ICML-13) (pp. 1-9).

[8] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[9] Sallab, A. E., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. Electronic Imaging, 2017(19), 70-76.

[10] Santana, E., & Hotz, G. (2016). Learning a driving simulator. arXiv preprint arXiv:1608.01230.

[11] Zhan, W., Li, J., Hu, Y., & Tomizuka, M. (2017, October). Safe and feasible motion generation for autonomous driving via constrained policy net. In Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE (pp. 4588-4593). IEEE.

[12] Sun, L., Peng, C., Zhan, W., & Tomizuka, M. (2017). A Fast Integrated Planning and Control Framework for Autonomous Driving via Imitation Learning. arXiv preprint arXiv:1707.02515.

[13] Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems, 13(4), 341-379.

[14] Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res.(JAIR), 13, 227-303.

[15] Bacon, P. L., Harb, J., & Precup, D. (2017). The Option-Critic Architecture. In AAAI (pp. 1726-1734).

[16] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161.

[17] Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., & Silver, D. (2016). Learning and Transfer of Modulated Locomotor Controllers. arXiv preprint arXiv:1610.05182.

[18] Frans, K., Ho, J., Chen, X., Abbeel, P., & Schulman, J. (2017). Meta learning shared hierarchies. arXiv preprint arXiv:1710.09767.

[19] Alur, R., Courcoubetis, C., Henzinger, T. A., & Ho, P. H. (1993). Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Hybrid systems (pp. 209-229). Springer, Berlin, Heidelberg.

[20] Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2), 181-211.

[21] Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. In Advances in neural information processing systems (pp. 271-278).

[22] Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems. In Advances in neural information processing systems (pp. 393-400).