Robot Learning HW 3

Cole Resurreccion, Kanghyun Lee

***Abstract*—In this paper, we use Deep Q-Learning to train a reinforcement learning agent to run on CARLA. The agent takes a 320x240 RGB image as an input. This paper looks at effect of changing parameters and different initialization of the Q-Learning model. It also experiments with different types of reward functions of a CARLA environment.**

I. Base Implementation

A. Deep Q-Network

The Deep Q-Network (DQN) takes 320x240 RGB frames as input and generates Q-values for output for a defined action space. The network uses a convolution neural network with five convolutional layers and a ReLU activation layer to process front-view image input. These layers progressively increase the number of filters from 24 to 64 and reduce the spatial dimensions of the input while capturing meaningful patterns essential for decision making in reinforcement learning. Then, the output of the convolution layers is flattened and passed through 3 fully connected layers, which progressively reduce the dimensionality (from 11264 to 50) before producing Q-values for each possible action[2]. The network outputs one of 4 possible actions: steering, steering right, throttle, or brake. This design balances computational efficiency and learning capacity, ensuring that the network can effectively approximate Q-values for action selection in the CARLA environment.

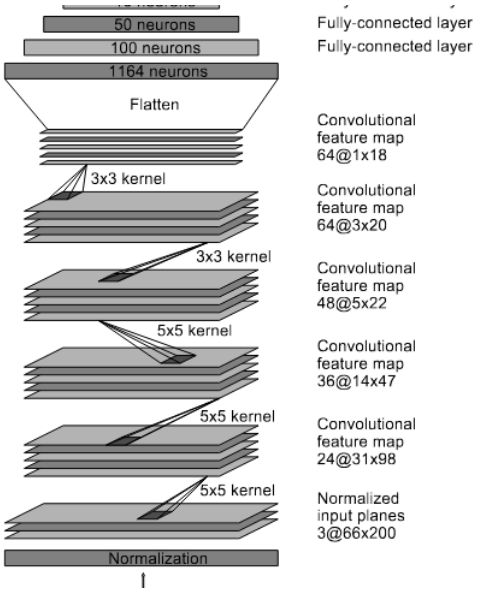

Fig. 1. Visualization of the model using a smaller input size.

B. Deep Q-Learning

Fixed targets are utilized through a separate policy network and a target network, where the weights of the target network are periodically updated with the weights of the policy network. As the target value is derived from the current Q-value estimate that depends on the same network being optimized, this can make Q-values during the learning process destabilized. Therefore, this approach reduces training instability, as the target network provides more stable targets during updates. Using consecutive frames causes inefficient learning, as the input data are too similar. And this can cause sticking to local minimum. Therefore, sampling training data from a replay memory helps break the temporal correlation between samples, which is critical to ensuring that the neural network learns more effectively. By randomly sampling experiences from a diverse pool, replay memory promotes better generalization and reduces the risk of over-fitting to specific sequences of events in the environment.

C. Action selection

It is important to balance exploitation and exploration. Exploitation allows the agent to use its current knowledge of Q-values to select the best actions with the highest immediate reward. However, this can lead the agent to stuck in a local optimum, as the agent might not have explored the environment enough and will make the most optimal decision from a tiny part of the exploration. If we prioritize exploration then the agent will only try new actions that has not been chosen before. The agent may spend time on non-important or non-optimal actions. With epsilon-greedy, we can set the value of epsilon, which will be the probability that agent will select a random action. A random action is known as exploration and the greedy action is known as exploitation.

D. Training

Observations from the training process show that the agent struggled to achieve positive rewards at the beginning, as it explores the state-action space. However, as the training progresses and the ϵ-greedy exploration schedule reduces the exploration rate ϵ, the agent increasingly exploits its learned policy, leading to a consistent improvement in cumulative rewards. The relationship between the exploration schedule and cumulative rewards show the importance of balancing exploration and exploitation; early exploration allows the agent to discover effective strategies, while later exploitation refines its performance. The generated loss curve shows a gradual decrease with occasional spikes, reflecting the agent's learning process and the non-stationary of the environment. This behavior shows difference from a typical supervised learning loss curve, which typically decreases smoothly, as reinforcement learning involves dynamic target values based on future rewards. Overall, the training process demonstrates

the agent's ability to learn from its interactions with the environment, with the loss and reward curves illustrating its progression from random exploration to effective decision-making.

E. Evaluation

The evaluation of the Deep Q-Learning agent, as observed from the provided reward and loss curves, reveals key insights into its performance and learning behavior. The reward curves show a gradual improvement over time, reflecting the agent's increasing ability to make optimal decisions as it transitions from exploration to exploitation during training. Early episodes demonstrate low and volatile rewards, indicative of the agent exploring the environment, while later episodes exhibit higher and more consistent rewards as the agent converges on an effective policy.

The loss curves, on the other hand, display a decreasing trend with occasional spikes. This behavior is expected in reinforcement learning due to the dynamic and non-stationary nature of Q-value updates. The initial high losses represent the agent's attempts to learn meaningful Q-value approximations, and the subsequent reduction indicates stabilization as the policy improves. The spikes correspond to significant updates caused by unexpected transitions or the agent encountering novel states.

Overall, the evaluation highlights the agent's success in learning to perform well in common scenarios, such as maintaining stability on straight paths and moderate turns. However, the agent struggles in complex scenarios, such as sharp turns or recovering from off-road situations, where rewards drop significantly. These observations suggest potential areas for improvement, such as enhancing the diversity of training scenarios and refining the reward function to encourage better handling of challenging situations.

II. Further Investigations and Extension

A. Discount factor

The discount factor γ determines the weight of future rewards relative to immediate rewards. With high discount factor, it leads the agent to prioritize the future more than immediate rewards and with low discount factor, it leads the agent to prioritize the immediate rewards more. And with the discount factor of $\gamma = 1$, it might look attractive as it would treat all future rewards equally important as immediate rewards. However, this can create some issues. First the Q-value would have harder time to converge, as it weights equally even on a very far-future occasion's reward. Also, as it treats all future rewards equally, it will have an infinite cumulative reward, so there would be continuous tasks without a clear termination. And finally, it will easily ignore the importance of immediate rewards, as immediate rewards have their values and it matters as much as future and far future rewards as well. In this investigation, varying γ from its default value of 0.99 reveals its influence on the agent's performance. With a higher discount factor (e.g.,γ = 0.999), the agent prioritizes long-term rewards, leading to more deliberate and strategic actions. However, this may result in slower convergence as the agent attempts to optimize for far-reaching consequences. Conversely, a lower discount factor (e.g.,$\gamma = 0.9$) causes the agent to prioritize immediate rewards, often leading to faster convergence but potentially suboptimal long-term behavior, as observed in the reward curves showing less stable and lower cumulative rewards over time.

B. Action repeat parameter:

The Action Repeat Parameter, set to 4 by default, determines the number of frames for which a selected action is repeated during training and evaluation. Adjusting this parameter has a significant impact on both the agent's training progress and evaluation performance.

From the reward curves, increasing the action repeat parameter generally results in smoother, more consistent increases in cumulative rewards. This reflects faster convergence during training because the agent effectively learns over extended time steps, reducing the noise associated with frequent decision-making. Conversely, reducing the action repeat parameter leads to more fluctuating reward curves, as the agent updates its actions more frequently. This results in greater responsiveness to environmental changes but less stability in learning.

The loss curves further highlight these dynamics. For higher action repeat values, the loss stabilizes earlier, as the agent performs fewer updates per episode. This smoothens the learning process and reduces overfitting to short-term dynamics. On the other hand, with lower action repeat values, the loss curves exhibit higher variability. This is due to the increased frequency of updates, which introduces more noise into the learning process and slows down convergence.

From the provided reward and loss plots, it is evident that higher action repeat values can help the agent achieve a more stable and efficient training process. However, these values may limit the agent's ability to adapt to rapidly changing situations, as actions are executed for longer durations without re-evaluation. Lower action repeat values enhance the agent's responsiveness but may lead to instability in both rewards and losses.

Why might it be helpful to repeat each action several times? Repeating actions several times reduces the computational burden of selecting actions at every time step. This is particularly beneficial in environments like CARLA, where rendering and processing frame-by-frame observations are resource-intensive. Additionally, repeating actions helps smooth the agent's behavior, preventing jerky or erratic movements that can arise from noisy Q-value estimations.

By balancing stability and adaptability, the action repeat parameter plays a critical role in shaping the agent's learning efficiency and driving style.

C. Action space

Expanding the action space by adding a null action ([0, 0, 0]) or more fine-grained steering, braking, or acceleration actions introduces notable changes in the agent's driving style

and evaluation score. Adding the null action will allow the agent to maintain its current state, as sometimes, the best decision can be keeping what it has been doing, such as keep driving straight. Therefore, it can improve stability and avoid unnecessary adjustments. Fine-grained actions provide greater control over the agent's movement, potentially leading to smoother driving and more precise maneuvering. However, a larger action space means the agent needs to evaluate more options and process more options to assign Q-values, which makes it more complicated for the agent, therefore this can be more complicated for the agent. This also can create more noise and if the added action is not important, then this will make the agent to process and compute more for such a little benefit.

Why are Deep Q-Networks limited to a discrete set of actions, and what solutions exist to overcome this limitation? Deep Q-Networks (DQNs) work over discrete action spaces. Therefore, Deep Q-Networks were born to to output Q-values for a fixed, discrete action space. This makes them unsuitable for handling continuous action spaces directly. To overcome this limitation, there are solutions such as Deep Deterministic Policy Gradient (DDPG). Deep Deterministic Policy Gradient is a model-free algorithm for continuous action spaces, so this can be combined with Deep Q Networks and handle the continuous action spaces. And Discretization of Continuous Actions can be used, it discretizing the continuous action space into bins, so it can effectively convert continuous continuous action space into a discrete action space problem.

D. Double Q-learning

Overestimation in Standard DQN In standard Deep Q-Learning (DQN), the Q-values are estimated using the Bellman equation. The maximization step during the target computation, $\max_a Q(s', a)$ can lead to overestimation because the same network is used to select and evaluate actions. This results in a bias toward overestimating the value of certain actions, especially when the Q-values are noisy or inaccurate during training. Over time, this overestimation can destabilize the training process and degrade the agent's performance.

How Double Q-Learning Addresses Overestimation Double Q-Learning resolves this issue by decoupling the action selection and evaluation processes. Instead of using the same network to select and evaluate the action, the policy network selects the action, and the target network evaluates it. Specifically, the target for a given state "s" is computed as: $Q_{\text{target}}(s, a) = r + \gamma Q_{\text{target}}(s', \arg\max_a Q_{\text{policy}}(s', a))$ This formulation reduces the bias because the action evaluation depends on the more stable target network, leading to more accurate Q-value estimates. As a result, Double Q-Learning improves training stability and leads to better convergence in most reinforcement learning scenarios.

E. Initialize from Behavior Cloning

Integrating behavior cloning with reinforcement learning allows for a more efficient and robust training process. Behavior cloning pre-trains the network using supervised learning on a dataset of expert demonstrations, providing the agent with a solid initial policy. This pre-training significantly reduces the initial exploration phase in reinforcement learning, as the agent begins with some understanding of how to navigate the environment effectively. During fine-tuning with reinforcement learning, the agent refines its policy by exploring and optimizing actions to achieve higher rewards. The training curves typically show a rapid initial increase in rewards due to the pre-trained weights, followed by slower, more consistent improvements as the agent explores further. Loss curves often stabilize faster, reflecting the initial proficiency of the pre-trained network. This approach leverages the structured learning from expert data while retaining reinforcement learning's adaptability, leading to better early performance and faster convergence. However, care must be taken as the pre-trained policy may inherit biases or limitations from the expert demonstrations, requiring reinforcement learning to overcome these inefficiencies.

Fig. 2. Reinforcement agent running in CARLA

F. Best solution

Our final solution initializes policy with imitation learning weights. In our experience, the main affect on our model was changing the reward function. Since CARLA does not have a built in reward function, we went through a lot of versions to find the best one. Many iterations caused the car to not move in order to maximize reward. To tune the network, we edited how much reward or penalty was caused by a state. For example, we set a collision state to give a -100 reward. Currently, our model does not stop for red lights or for other cars. One way to fix this would to be had additional data to the model input, like velocity, so the model knows how fast it is moving.

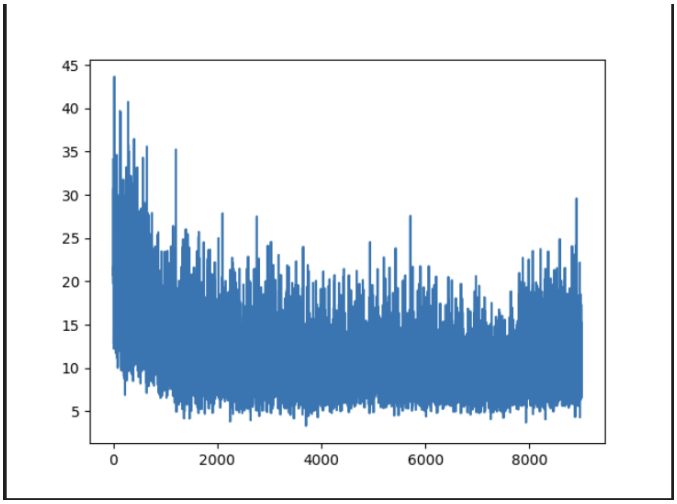

Fig. 3. Loss of final training model

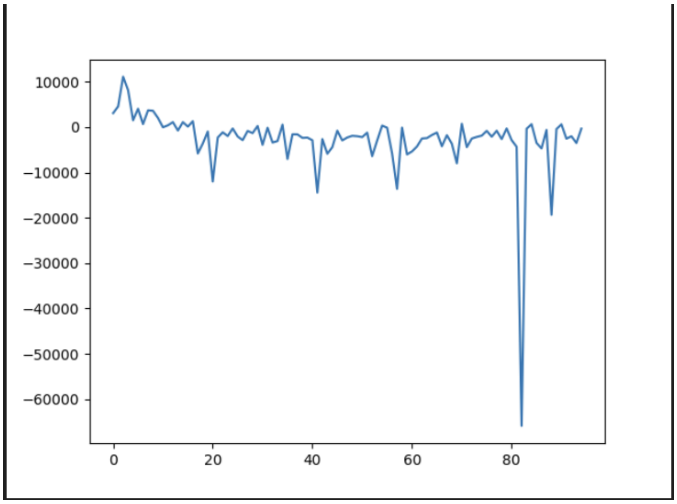

Fig. 4. Reward of final training model

REFERENCES

[1] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning." Available: https://arxiv.org/pdf/1509.06461

[2] M. Bojarski et al., "End to End Learning for Self-Driving Cars," 2016. Available: https://arxiv.org/pdf/1604.07316