Visual Cart-Pole with Deep Reinforcement Learning

Krittin Janjaochay, Rom Parnichkun, Sitiporn Saelim
Computer Science and Information Management
Asian Institute of Technology
Thailand
{st121417, st121411, st121532}@ait.asia

***Abstract*—Several recent deep reinforcement learning agents are tested and compared on the cart-pole problem with only visual observations as the input. Results show that policy gradient based methods outperform DQN based methods in this problem domain.**

***Index Terms*—deep reinforcement learning, control**

I. Introduction

The cart-pole problem is a classical control problem in which forces are applied to the base of a cart to balance a freely rotating pole connected to it. The unstable nature of this problem poses an interesting problem to solve for both classical controllers as well as reinforcement learning based controllers.

Recent advances in deep reinforcement learning have shown that it is possible to create policies that master games through raw visual inputs. In this project, we explore the viability of these recent deep reinforcement learning agents in the cart-pole problem. However, the input to the reinforcement learning agent is the raw visual input rather than the environment states.

II. Literature Review

A. Double DQN

The Deep Q Network is a multi-layered neural network by given state s outputs a vector of action values XXX Q(s,.;theta), where theta are the parameters of the network. For n-dimensional state space and an action space containing m actions, the neural network is a function of Rn to Rm.

The problem of Deep Q learning is overestimation of action value, Q as a neural network because most of the time agents select the non-optimal action in any given state due to maximum Q-value. When such a problem occurs, there is noise from the estimated Q-value that causes a large amount of positive biases in updating Q-value. As a result, the learning process of the model will be complicated which causes the difficulty to learn. The reason is the best Q(s,a) which is estimated by using current Q(s,a). As a result, it is very noisy. Also,The difference between the best and current Q(s,a) is messy which is caused by different noises. However, overestimation is not the problem if all noise of Q-values are uniformly distributed. On the other hand, Double Deep Q-Learning employs two different action-value networks namely Q and Q' as their estimators. These noises can be seen as uniform distribution. Thus, this algorithm can solve the problem.

Identify applicable funding agency here. If none, delete this.

Double Q-Learning network uses two different neural networks, Deep Q Network(DQN) and Target Network. It can be noted that there is no learning rate alpha when updating Q-values as it will be used in updating the parameters of optimization of Deep Q Network. (edit) Selecting the best action with maximum Q-value in the given next state while a target network estimates Q-value with action above. Updating the Q-value of the Deep Q network is based on the estimated Q-value from the target network, and the target network is based on the parameter of the Deep Q network per several iterations.

B. Dueling DQN

The difference in Dueling DQN is in the structure of the model which makes model learning faster. Instead of a single sequence of fully connected layers, two sequences of fully connected layers are replaced. This two sequences represent state value V and advantage A functions which use for estimate Q value as 1. State value function estimated the value of a given state. Advantage function shows how advantageous selecting action is relative to the other at the given state. Separate tasks to value and advantage function make model converge faster.

$$Q^{\pi}(s,a) = V^{\pi}(s) + (A^{\pi}(s,a) - \frac{1}{|A^{\pi}|}\sum_{a} A^{\pi}(s,a')) \quad (1)$$

$$V^{\pi}(s) = E_{a\sim\pi(s)}[Q^{\pi}(s,a)] \quad (2)$$

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \quad (3)$$

Subtract A with mean force it to have zero advantage at the chosen action. While we lose the original semantics of V and A, it increases stability of the optimization. The relative rank of Avalues remain the same. The separate sequences create advantages to learn the stage-value and advantage function efficiently. In update state, Vvalue and all of Avalues get updated while a single-sequence architecture only updates the value of one chosen

C. Policy Gradient

Rather than approximating the Q-function of the environment/problem, policy gradient based methods directly optimizes and approximates a stochastic policy. The parameterization of the stochastic policy makes it possible to find the gradient which updates the policy towards higher performance. Both sum of discounted rewards and average reward over time can be used as the performance metric to update the policy upon. The following formula from [1] is the gradient which is used to update the policy

$$\frac{\partial P}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} Q^\pi(s,a) \tag{4}$$

Where P is the performance, θ is the policy parameters, d^π is the stationary state visitation probability following a policy π, s is the state, a is the action, and Q^π is the policy state-action value of the policy. Note that in policy gradient methods, Q^π is often approximated using the actual returns at the end of the episode. Therefore, training may be slower than DQN and its counterparts which use a TD method to update the Q function. Furthermore, Q^π is often replaced with the advantage function $Q(s,a) - V(s)$ where V is policy state value, as it reduces the gradient's variance introduced through the value function.

D. Actor Critic

The actor critic method is an improvement from the policy gradient method in which the state-action value is approximated with a parameterized function using DQN based methods. The policy network acts as the actor whereas the Q network acts as the critique, hence actor critic. By using a separate Q network as the critique, the policy can be updated in a TD based fashion, and the state-action value will also have less variance, as they will be based on the Q network rather than the actual returns of each episode, which the value for any state changes per episode.

III. Methodology and Results

A. Cart-Pole Environment

The environment used to benchmark the reinforcement learning algorithms are based on Open AI Gym cart-pole-v0. A +1 reward is given for each timestep the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. In this project, the goal is to keep the pole standing upright for 1000 steps. Although there is an API to get the actual states of the cart-pole system, only raw visual images were fed into the reinforcement learning algorithm.

B. Double DQN

Grayscale input The raw visuals images are converted to gray scale, and resize to 150x40 and 90x40. A sequence of 2 images are then stacked on top of each other to be fed into the neural network. The network architecture consisted of 3 convolutional layers with 5x5 filters and a stride of 2 having hidden sizes of 32, 64, 64 respectively. The output from the convolutional layers are then fed into 1 fully-connected layer. The ReLU activation function is used for all the activations aside from the output.

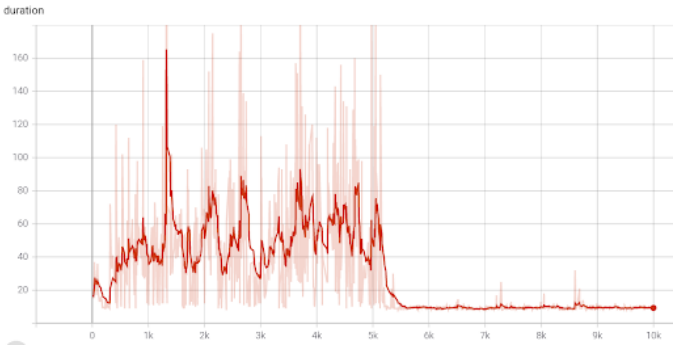

Fig. 1. Steps vs Episodes with Gray Scale full-width Input (Double DQN)

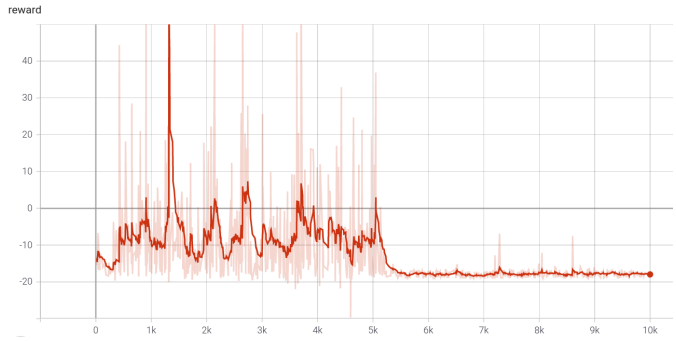

Fig. 2. Rewards vs Episodes with Gray Scale full-width Input (Double DQN)

Color input The raw visuals images are converted to RGB scale, and resize to 150x40 and 90x40. A sequence of 2 images are then stacked on top of each other to be fed into the neural network. The network architecture consisted of 3 convolutional layers with 5x5 filters and a stride of 2 having hidden sizes of 32, 64, 64 respectively. The output from the convolutional layers are then fed into 1 fully-connected layer. The ReLU activation function is used for all the activations aside from the output. gray

The Adam optimizer with a learning rate 1e-6 is used as

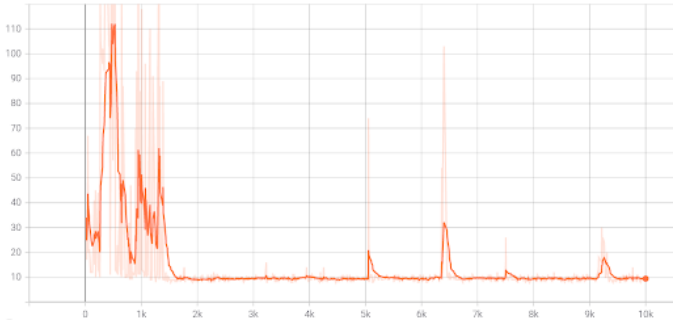

Fig. 3. Steps vs Episodes with Gray Scale Non with full-width Input (Double DQN)

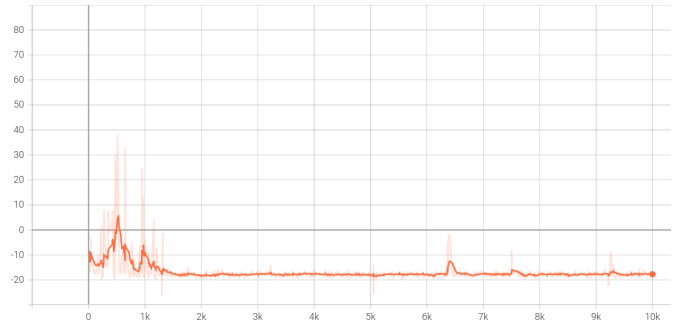

Fig. 4. Rewards vs Episodes with Gray Scale Non full-width Input (Double DQN)

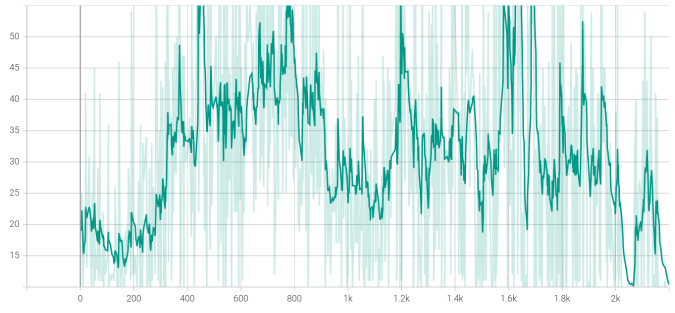

Fig. 5. Steps vs Episodes with RGB Scale full-width Input (Double DQN)

the optimizer to train the networks.To initialize two networks, current network and target network are employed. Then, actions are selected by given initial state.For each iteration, the model runs each environment step by taking action and storing state, action, reward, and next state in replay buffer. Then, each update steps samples state, action, reward, and next state. Mean squared error are computed by expected Q-value employed target network and policy network.

C. Double Dueling DQN

Implement Dueling DQN is done in a neural network architecture. From the basic DQN structure, we replace a single fully-connected layer with two separate streams. Both begin with the same size fully-connected layer and ReLU activation function. For the value stream, It has one output. The advantage stream output with the same amount of the

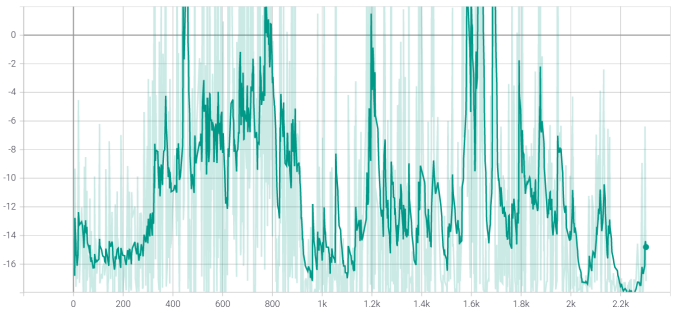

Fig. 6. Rewards vs Episodes with RGB Scale full-width Input (Double DQN)

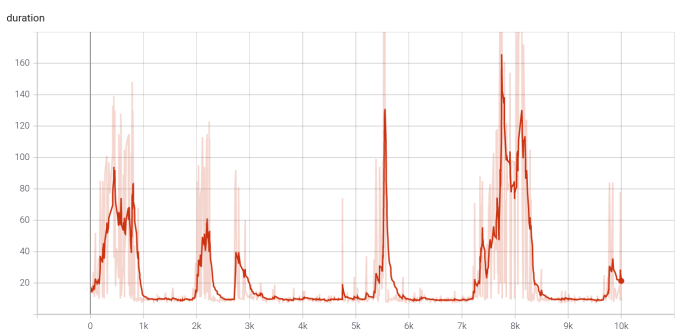

Fig. 7. Steps vs Episodes with RGB Scale Non full-width Input (Double DQN)

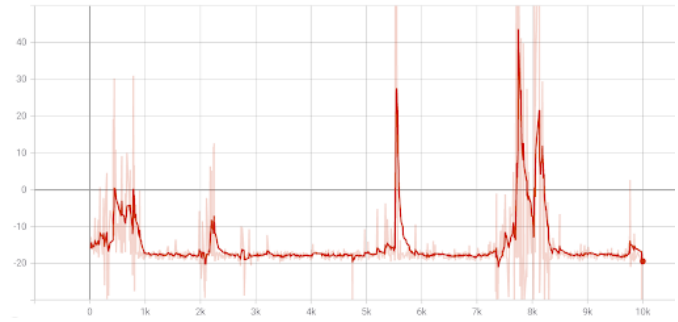

Fig. 8. Rewards vs Episodes with RGB Scale Non full-width Input (Double DQN)

actions. Separate stream will encourage each function because for many states, it is unnecessary to estimate the value of each action choice. Q value which is the output of the model will be calculated as an equation mentioned above.

The screens are converted to grayscale, and resize 150x40 pixels. A sequence of 2 screen images are ordered stack by queue. We explore two types of image, crop and full-width. The crop screen will keep the cart at the center of the square image while full-width will use the original resize image. We decided to use Adam optimizer with 1e-7 learning rate.

For the crop version, the best model can reach 1000 steps for some epoch. However, the model is very unstable. Please note that this model initial by load weight from the previous training at the period that it performed the best.

After some discussion, we decided to use the same setup to every experiment. Therefore, In the full-width version, we did not load the previous weight. There is one time that a model can perform 1000 steps, but the model still unstable and drop performance after their experience is full 9 10.

D. Policy Gradient

The actual returns are calculated after each episode and the mean of the returns are subtracted from the actual returns to be fed into the calculation of the policy gradient. Therefore, with $\gamma = 0.99$ approximately the first half of the episode have positive advantage values, and the second half of the episode have negative advantage values. Since, the actual returns can

Fig. 9. Training Episodes vs Steps (Double Dueling DQN)

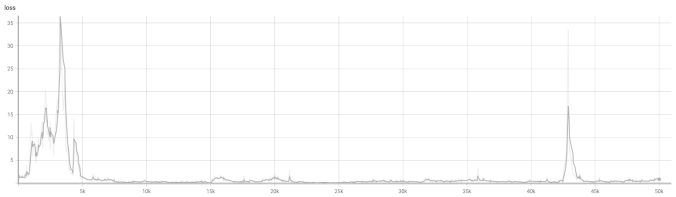

Fig. 10. Training Episodes vs Loss (Double Dueling DQN)

only be calculated after the episode, the policy network was updated after the end of every episode.

The raw visual images are converted to grayscale, and resized to 150x40 pixels. A sequence of 2 images are then stacked on top of each other to be fed into the neural network. The network architecture consisted of 3 convolutional layers with 5x5 filters and a stride of 2 having hidden sizes of 32, 64, 64 respectively. The output from the convolutional layers are then fed into 2 fully-connected layers. The ReLU activation function is used for all the activations aside from the output which uses a Softmax activation.

The Adam optimizer with a learning rate of 1e-5 is used as the optimizer to train the network. The average entropy of the policy output is calculated after each episode. Depending on the learning rate chosen, the vanilla policy gradient method is able to get up to approximately 200-300 steps. However, as the network continues training, the policy often converges suboptimally and lowers the entropy towards becoming deterministic. This behavior often results in the network becoming unstable and unable to control the cart-pole system in the long run.

To promote exploration and stabilize training, entropy loss is introduced to the learning objective. The entropy loss increases the policy entropy, preventing the policy from becoming a deterministic policy. The results show that with entropy loss and approximately 25 hours of training on a single RTX 2080 GPU, the policy network is able to train up to 1000 steps. The long training time can be attributed to the fact that the

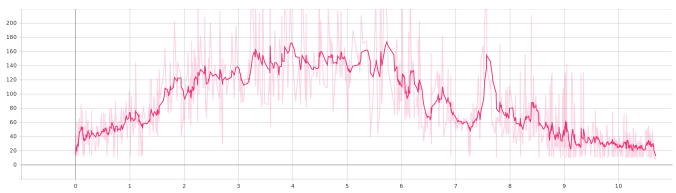

Fig. 11. Training Time (hours) vs Steps (Vanilla Policy Gradient)

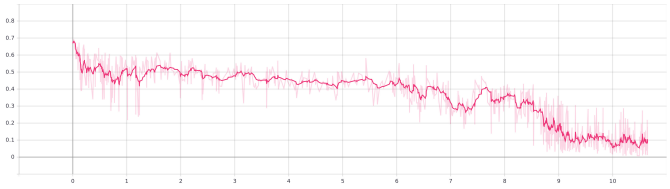

Fig. 12. Training Time (hours) vs Entropy (Vanilla Policy Gradient)

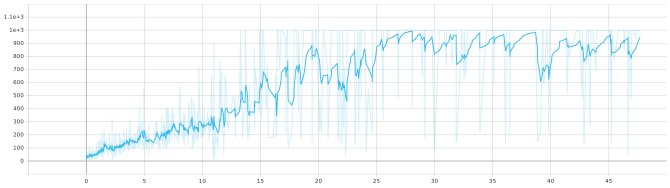

Fig. 13. Training Time (hours) vs Steps (Policy Gradient with Entropy loss)

advantage function of any particular state can greatly vary with each episode which causes the policy gradient to have high variance.

E. Actor Critic

Unlike the previously mentioned policy gradient method, actor critic methods utilize a Q-value function which can produce accurate state-action values irrespective of the current episode. Both off-policy and on-policy methods for training the Q-value function are explored.

2 networks are trained in actor critic methods. The policy network is trained in the same way as the policy gradient method but since the episode doesn't have to end in order to calculate the state-action value, TD methods can be utilized for training. The Q network on the other hand can be trained in a similar fashion to both DQN(off-policy) and SARSA(on-policy) methods. Without the replay buffer from DQN, the network was able to perform up to approximately 450 steps after training for 10 hours. In contrast, by utilizing the replay buffer, the network was able to train up to 1000 steps in merely 2 hours of training. However, initialization proved to be a big factor as training with the same parameters did not always yield similar results. By introducing entropy loss, the network was able to consistently get up to 1000 steps after approximately 2 hours.

IV. Conclusion

In conclusion, we found that training neural network agents to balance a cart-pole using raw visual inputs is still challeng-

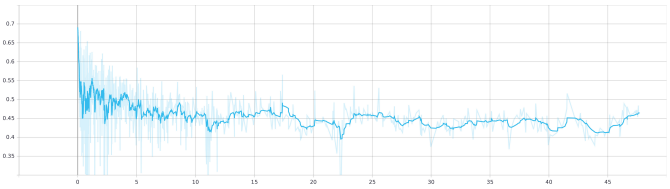

Fig. 14. Training Time (hours) vs Entropy (Policy Gradient with Entropy Loss)

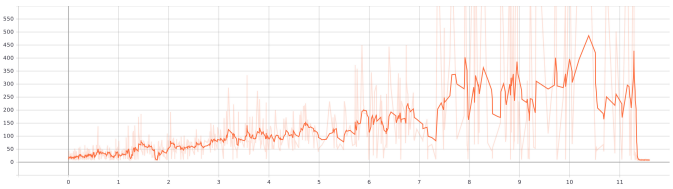

Fig. 15. Training Time (hours) vs Steps (Actor Critic)

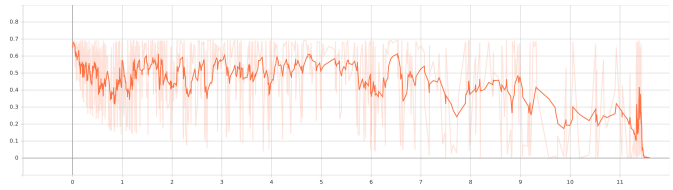

Fig. 16. Training Time (hours) vs Entropy (Actor Critic)

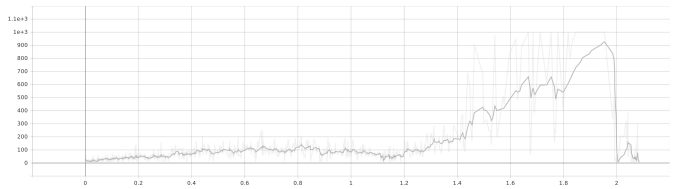

Fig. 17. Training Time (hours) vs Steps (Actor Critic with Replay Memory)

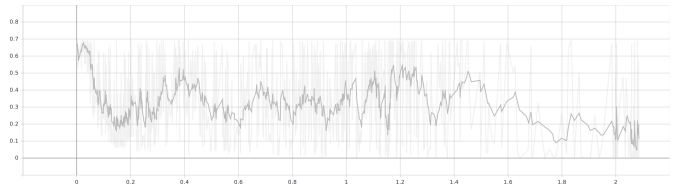

Fig. 18. Training Time (hours) vs Entropy (Actor Critic with Replay Memory)

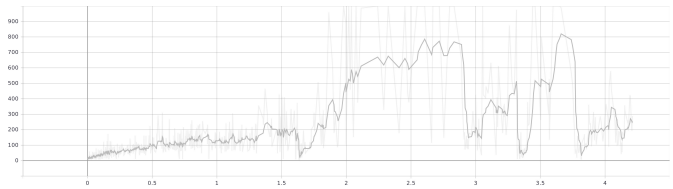

Fig. 19. Training Time (hours) vs Steps (Actor Critic with Replay Memory and Entropy Loss)

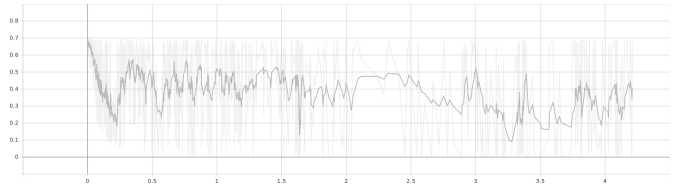

Fig. 20. Training Time (hours) vs Entropy (Actor Critic with Replay Memory and Entropy Loss)

ing as it requires many tricks to get it to work. The training was also extremely unstable and sensitive to hyperparameters. From our experiments, we found that DQN based methods struggled to consistently get up to 1000 steps, this instability may be attributed to the fact that DQN methods have a maximization bias. Furthermore, methods based on policy gradient has the advantage of being able to utilize entropy loss to set the exploration vs exploitation dynamically.

REFERENCES

[1] Sutton, Richard & Mcallester, David & Singh, Satinder & Mansour, Yishay. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. Adv. Neural Inf. Process. Syst. 12.

[2] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016, June). Dueling network architectures for deep reinforcement learning. In International conference on machine learning (pp. 1995-2003). PMLR.

[3] Hado van Hasselt and Arthur Guez and David Silver.(2015). Deep Reinforcement Learning with Double Q-learning.1509.06461. cs.LG.