No Greed No Gain: Solving OpenAI Gym Games with Q-Learning and Variants*

Priyam Mazumdar (priyamm2)
Statistics
priyamm2@illinos.edu

Gowthami Venkateswaran (gv4)
Agricultural and Consumer Economics
gv4@illinois.edu

Yang Yue (yangyue3)
Statistics
yangyue3@illinois.edu

***Abstract*—In this report, we first provide a brief review of the Q-Learning algorithm and its variants implemented in the project. Training results from the OpenAI Gym's *CartPole-v1* (Classic Control), *LunarLander-v2* (Box2D), and *Pong-v0* (Atari) environments, as well as some technical details are presented and discussed. We observe similar levels of performance from our implementations of Deep Q-Learning and Double Deep Q-Learning algorithms in the first two environments using the built-in numerical state values. On the other hand, meaningful learning is absent when we switch to using static single-frame images as inputs due to the lack of temporal information. By leveraging various frame processing techniques, we successfully train a Double Deep Q-Learning agent and a Dueling Deep Q-Learning agent in the image-based Atari environment *Pong-v0* with the second model offering significantly better performance than the first one.**

***Index Terms*—Reinforcement Learning, Q-Learning, Neural Networks**

I. Introduction

Neural networks can often be thought of as an iterative function approximator and can express complex relationships that cannot be easily derived in closed form. These networks have traditionally been used with data-related tasks, where users have some input and are expecting a specific output and the goal is to gain some predictive power when given unseen data. Our group want to explore Reinforcement Learning algorithms where some agents can interact with an environment and learn how to thrive within it.

Reinforcement Learning begins with Q-Learning and the Bellman Equation, and a lot of the assumptions are deeply embedded in the Markov Chains. Q-Learning is known as model-free learning, where an agent observes the state of its environment, performs an action, gets rewarded for that action, and then observes the new state it is in following that action. All calculations happen on a Q-table where we store the expected returns for every state-action pair. As the table gets updated over time, the agent will learn the optimal action to take in a given state. The Q-value finding process can be represented by a Bellman equation, which describes the value of an action by its immediate payoff and the expected value of the remaining decision problem. A discount factor γ is imposed on future rewards to capture how patient the agent is. Given enough iterations, the values in the Q-table will converge, which allows the agent to pin down the Q-value maximizing action to take in each state.

*This is the final project report of the LeepDearning group. We thank Professor James Balamuta for offering valuable suggestions. Our code can be accessed through the team GitHub repository.

Unfortunately, there are limitations that come with the Q-table. Most notably, the curse of dimensionality. Q-table becomes infeasible when we have large discrete state spaces or continuous state spaces. One possible solution is to bucketize the observations into categories, which comes with the caveat of having sub-optimal performance. A better solution is to leverage Neural Networks to approximate complex value functions.

For the games using numerical values to describe the states, we use a three-layer dense neural network with *ReLU* activation for hidden layers (256 neurons) and Linear activation for the output layer. For games using images as input, our network contains three convolutional layers and two fully connected layers. The specifications are exactly the same as the one used in the Mnih et al. (2015) [1] paper. For all our networks, we adopt the Huber loss as the loss function and the Root Mean Squared Propagation (RMSProp) as the optimizer.

II. Literature Review

Q-Learning as a simple way for agents to learn how to behave optimally in a Markovian environment was first introduced by Watkins (1992) [2]. Although it achieves some level of success, its application is limited to low-dimensional state space environments. In an effort the expand the applicability of the Q-Learning algorithm, Mnih et al. (2015) [1] introduced a Deep Q-Network (DQN) in their seminal paper to accommodate for high-dimensional or continuous state spaces. However, van Hasselt, Guez, and Silver (2016) [3] showed that the network suffers from similar overestimation issues prevalent in the classic Q-Learning algorithm. The quality of the resulting policy could be negatively affected when the overestimations aren't uniform across all states. To alleviate the issue, they proposed to use the Double Deep Q-Network (DDQN) where the action selection and the action value are obtained from two networks. The frontier was further pushed by Wang et al. (2016) [4] where the Dueling architecture was introduced. The new model achieves better policy evaluation when facing many similar-valued actions and outperforms previous models using Atari 2600 games as benchmarks. As an exercise inspired by the literature, we implement and benchmark the performance of the variations of the Q-Learning algorithms in the OpenAI Gym environments and investigate how changes of the input format (numerical to image) affect the observed performance.

III. Methods

Fundamentally, reinforcement learning is achieved by exploiting the Markov Decision Process where an agent repeatedly interacts with the environment and receives rewards based on the actions taken. The Q-Learning framework is a value-based learning algorithm that estimates the discounted lifetime value of each action and chooses the rewards-maximizing action in all states. We can use a Bellman equation to characterize the optimization process

$$\begin{aligned} Q_{\text{new}}(s_t, a_t) &= Q_{\text{old}}(s_t, a_t) \\ &+ \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q_{\text{old}}(s_t, a_t)) \end{aligned}$$

where α is the learning rate, r_t is the reward received from the environment, γ is the discount rate, s is the state, and a is the agent's action.

In addition to the model specifications covered in the previous sections, we want to go into more details regarding the Dueling Deep Q-Network architecture, which introduces value and advantage functions to the model. Note that the previous models are built to estimate the Q-value for all states under the assumption that every state is equally important, which could be problematic in certain cases. Consider the following example:

We use reinforcement learning algorithms to enable an autonomous vehicle to derive from a house in the suburb to an office downtown. The vehicle needs to drive through a neighborhood with narrow roads before getting on the highway. As the vehicle gets closer to the city, the traffic also gets worse. It also needs to deal with traffic lights and pedestrians once it arrives at the city.

It is reasonable to claim that the scenarios the vehicle needs to deal with on its way to the office are not equally chaotic. If a driver is present, she may choose to turn on auto pilot in the suburb but manually navigate through the city traffic. Dueling Deep Q-Networks work in a similar way: Q-value is calculated as the sum of the value function and the advantage function that measures how advantageous an action is compared to the others (with a scalar parameter that weighs the importance of a state). The value function takes the output of the fully connected layer and produces a single value. The advantage function uses the same fully connected layer output and the action space dimension as inputs. Note that as discussed in Wang et al. (2016) [4], adding the value function and the advantage function together

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

doesn't work due to unidentifiability. We can instead use

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha))$$

so that our network calculates the advantage-mean advantage value for every action plus the value of the state.

One more thing we want to touch on before discussing the results is the construction of replay buffer. We construct a function to store the history of agent-environment interactions (current observation, action, reward, done, and new observation) as memory for random sampling. We set the maximum length of such memory to be $200,000$ with deque mechanism incorporated.

IV. Results and Analyses

We adopt the simulation of games within the OpenAI Gym environment to train agents using different Reinforcement Learning algorithms.

A. *CartPole-v1*

We start with the simple *CartPole-v1* environment. The cart is an agent that takes discrete actions. The pole stands upright and the goal of the game is to prevent the pole from falling over by strategically moving the cart to the left or right. There are two possible actions: to move right or left and four observations that involves the numerical values for the position of the cart, the velocity of the cart, the angle of the pole, and the rotation rate of the pole. For every step taken, a reward of +1 is given. The game is considered solved when the average return is greater than 195 score points over 100 consecutive trials. First, we apply a Q-Learning model to the environment. Fig. 1 reports the score per episode for $10,000$ episodes. We see that the moving average scores of the most recent 100 episodes fail to cross the specified threshold.

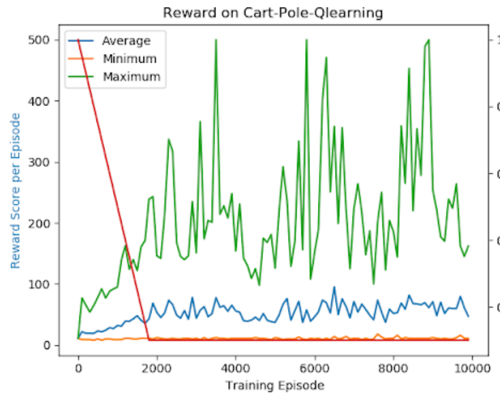

Fig. 1. Q-Learning scores for *CartPole-v1*

Since the observation space is continuous, we implement a model that employs Deep Q-Learning. Fig. 2 shows the scores for the agent over a range of 500 episodes. We see that after 100 episodes, the average score stabilizes at around 175 points. It is important to note that since the algorithm allows for occasional random action, it is within our expectation that we observe significant minimum score fluctuations. Compared with the basic Q-Learning algorithm, it is clear that the Deep Q-Learning algorithm yields better training outcome.

Next, we attempt to run the same model using static one-frame screenshots of the game as a substitute for states. Fig. 3 reports the scores obtained using images and a Convolutional Neural Network. We can see that the agent fails to achieve learning over 500 training episodes. The score points on average are below 40 points which is much lower than the score obtained in our earlier models. This could be due to the

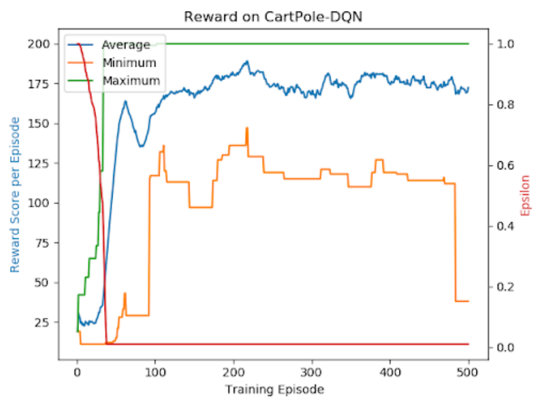

Fig. 2. Deep Q-Learning scores for *CartPole-v1*

fact that temporal information is not captured in static images used in this model.

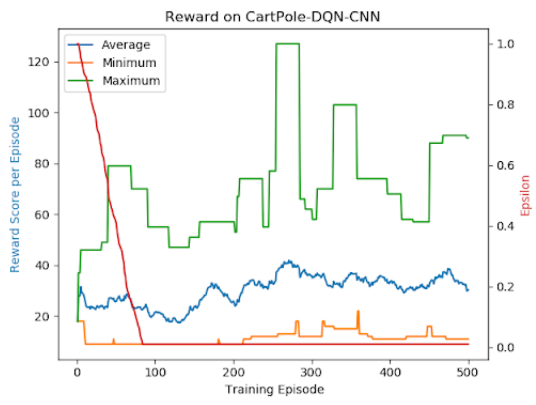

Fig. 3. Image-based Deep Q-Learning scores for *CartPole-v1*

Finally, we implement a Double Deep Q-Learning algorithm to the environment using static images as inputs. Fig. 4 reports the scores for each episode. We cannot conclude that the model performs better than our preceding model.

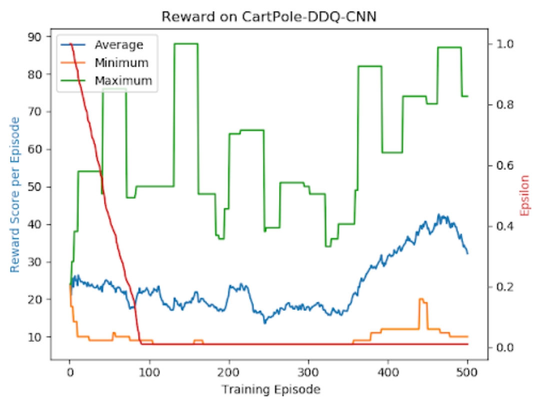

Fig. 4. Image-based Double Deep Q-Learning scores for *CartPole-v1*

B. *LunarLander-v2*

In addition to classic control problems like *CartPole-v1*, we also apply multiple Q-Learning models to the Box2D *LunarLander-v2* environment. As a brief introduction, the agent's goal is to land inside the landing zone without crashing (soft landing) by taking as few actions as possible (fuel-efficient). Although the action space is discrete, the state space is continuous as it describes the lander's coordinates, velocities, etc. Therefore, Deep Q-Learning is a good starting point within the Q-Learning algorithm family. In addition, we also implemented the Double Deep Q-Learning algorithm to the *LunarLander-v2* environment. Both implementations use the built-in numerical state values as inputs. Our objective is to check if there exists noticeable difference in the performance of the two models. Note that based on the action-reward menu, a total score of 200 per game is typically considered as the success threshold.We will adopt this criterion when comparing the performances of the models.

Training results over 2,000 episodes are shown in Fig. 5 and . 6. With the same hyperparameters, the Deep Q and the Double Deep Q algorithm yield similar performances with the latter being slightly better if we use the number of training episodes needed to cross the 200 average score threshold as the measure. Note that we can reduce the number of training episodes required for both models if we accelerate the epsilon decay, that is, reduce the amount of time the agents spend performing exploration. Also, without additional exercises like exploring the performances under different sets of hyperparameters, we cannot conclude that Double Deep Q is the superior algorithm for *LunarLander-v2* when compared with Deep Q. The two graphs above simply confirm that our agents are indeed learning. Notice that there exists a curious dip in scores for the Deep Q model in Fig. 5. We hypothesize that it is due to consecutive back luck in either the action-selection policy or replay memory sampling.

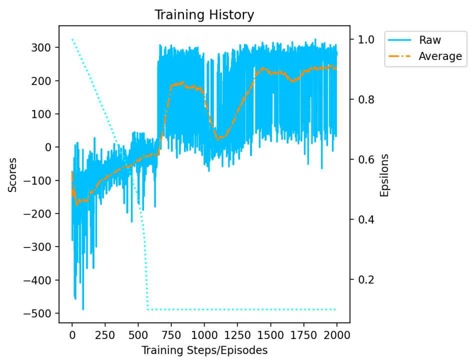

Fig. 5. Deep Q-Learning scores for *LunarLander-v2*

As shown in Fig. 7, our current image-based model is unsuccessful. Using the static one-frame screenshots as states, the agent fails to achieve meaningful learning over $10,000$ episodes.

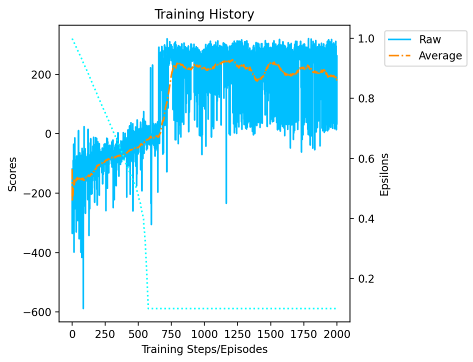

Fig. 6. Double Deep Q-Learning scores for *LunarLander-v2*

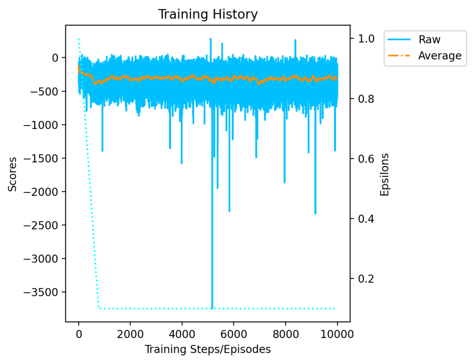

Fig. 7. Image-based Double Deep Q-Learning scores for *LunarLander-v2*

C. *Pong-v0*

In most video games, human players aren't asked to use the numerical approximations of the environment to determine the optimal actions. Instead, we interact with a game based on its visual outputs and attempt to extrapolate all the parameters being calculated internally for the game engine to function. We, as experienced players, can quickly create a map of what action to take based on the observation. We can replicate this process by mapping complex images to optimal actions using Convolutional Neural Networks. Convolutions are ideal because we can gain latent structures of the game while retaining the spatial locations of objects.

Open AI Gym gives us access to Atari 2006 Games, a set of retro games that are relatively simple for a human to master. Hard-coding a solution for an agent to follow has the downside of being too game-specific, if such an approach is feasible in the first place. Instead, we will use Convolutional Neural Networks to feed batches of images from the game environment and perform the score maximizing actions. To implement this idea and obtain decent performance, we need to tackle several issues that come with the Atari game environments.

Firstly, when the agent performs an action, the game engine will randomly (uniform distribution) repeat the action two to four times. We can avoid this by using a *NoFrameSkip* parameter so that each action is paired with one frame of the game. Secondly, depending on how quickly the agent is moving, we could have the difference between two consecutive frames being almost negligible. We set it so that every action taken in the game is repeated n times (4 in this project) to have enough differentiation between the images. Thirdly, some of the Atari games are half-rendered (e.g., if we have 4 trees in an image, 2 trees are rendered in the first frame while the other 2 are rendered in the second frame). While this doesn't affect human players, it can confuse the network as we are taking frame by frame data. We can fix this by taking the maximum of two consecutive frames and use the result as a single-frame input for the model. Since consecutive frames are nearly identical, this won't cause any problems such as ghosting. Lastly, we need to overcome the issue of static images not fully capturing the agent's state information (e.g., directions of movement). To fix this, we implement the frame-stacking technique. Instead of feeding a single image into our network, we pass in a $\phi \times N \times N$ image where N is our image dimension and ϕ is the number of frames we stack. The combination of images will allow the network to determine the current trajectory of a projectile.

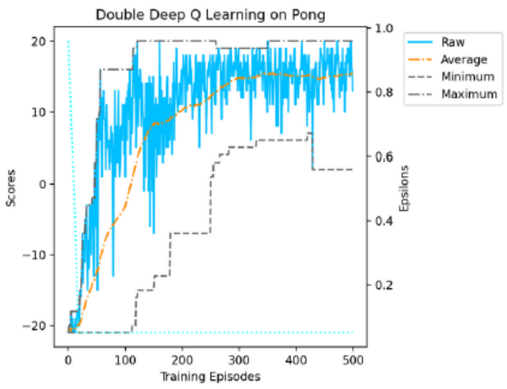

Fig. 8. Image-based Double Deep Q-Learning scores for *Pong-v0*

The *Pong-v0* environment is set up such that every time the agent scores a point it gets a +1 reward and -1 otherwise. The highest score is 20, corresponding to a winning streak of 20 games. We can see in Fig. 8 that the win-loss ratio approximates $50 - 50$ after 150 games. The agent obtains an average of 17 points over the most recent 100 games by 500 training iterations. The minimum score seems to have initially plateaued at around 50-50 win-loss ratio, but it quickly reaches a much more comfortable $+10$ win delta towards the end of the training session.

Fig. 9 shows the results of the Dueling Deep Q-Network. Both the Double Deep Q-Network and the Dueling Deep Q-Network converge to a score of 16, but the latter achieves the convergence by playing only half the number of games required by the former.

V. Conclusion

We implement multiple Q-Learning algorithms to three OpenAI Gym environments. We demonstrate that Deep Q-

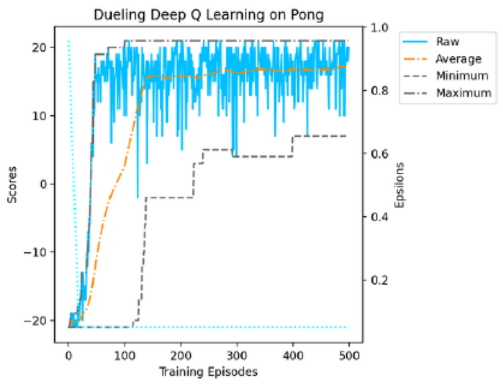

Fig. 9. Image-based Dueling Deep Q-Learning scores for *Pong-v0*

Learning algorithm outperforms the basic Q-Learning algorithm in *CartPole-v1* while Deep Q-Learning and Double Deep Q-Learning yield similar performance in *LunarLander-v2*. Using static one-frame images as input instead of numerical states, the networks fail to achieve meaningful learning in both environments. We believe incorporating the frame stacking technique can potentially fix the issue. Lastly, we implement the Double Deep Q-Learning and the Dueling Deep Q-Learning algorithms to the Atari 2600 *Pong-v0* environment. The benefit of using the Dueling model is significant since it can prioritize states that are important.

One other observation we have for the Atari environments is that it is relatively costly to train (6 hours to finish 500 episodes of *Pong-v0* with Titan GPU vs. 3 hours to finish $10,000$ episodes of *LunarLander-v2* with RTX 3080). Since the game is played in real-time, each episode will take longer to finish as the agent improves and survives longer in the environment.

For future iterations, in addition to implementing frame stacking to the first two environments, we also plan to upgrade our current replay buffer to a prioritized replay buffer. As explained previously, the replay buffer stores the historical agent-environment interactions. Since not all such memories are created equal, being able to prioritize the more salient memories based on the value function in the Dueling Deep Q-Network could lead to additional increase in performance.

Acknowledgement

While our work relies primarily on the key papers mentioned in the report, we have benefited from alternative perspectives online [5] and [6].

Team Evaluation

After multiple individual and group study sessions and making sure that each member can code the Q-Learning algorithms discussed in this report independently, each of us is responsible for one game from the OpenAI Gym environment (Priyam: *Pong-v0*, Gowthami: *CartPole-v1*, Yang: *LunarLander-v2*). We propose a grading schema that assigns Priyam a score of 33.34 and the other two members a score of 33.33.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[2] C. J. Watkins and P. Dayan, "Q-learning", Machine learning, vol. 8, no. 3-4, pp. 279-292, 1992.

[3] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning", Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[4] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, "Dueling network architectures for deep reinforcement learning", International conference on machine learning. PMLR, 2016.

[5] D. Seita, "Frame skipping and pre-processing for deep Q-networks on Atari 2600 games", 2016, https://danieltakeshi.github.io/about.html.

[6] P. Tabor, "Modern reinforcement learning: deep Q-learning in PyTorch", Udemy, 2020, https://www.udemy.com/course/deep-q-learning-from-paper-to-code/.