Final Project Report: Deep Reinforcement Learning for Atari Games

Akshata Desai, Ran Liu, Anni Lu, Anusha Srikanthan, and Sarah Wiegreffe

Project summary

Deep Reinforcement Learning is a subset of AI which lies at the intersection of Deep Learning and Reinforcement Learning(RL). Generally speaking, learning methodologies fall into three major categories - supervised learning, unsupervised learning and reinforcement learning. Reinforcement learning is a self supervised learning algorithm involving the use of an RL agent which interacts with its environment, over time, to learn an optimal policy for a sequential decision making problem. From this definition, the key idea is that RL is applicable for problems that can be setup as a sequence of time steps where experience gained over time can improve the agent's understanding of its environment. At each time step, the agent receives a state from the state space and chooses an action from the action space to maximize the expectation of long term return from each state [1]. We obtain Deep RL methods by using neural networks to represent the state/ observation and to approximate the components of RL such as value function, policy and model. Deep Learning is effectively combined with RL where the neural networks serve as strong function approximators [2].

However, there are issues with Deep RL such as sample efficiency, sparse reward, credit assignment, exploration vs exploitation and representation. In an effort to mitigate some of them, we have investigated the models and pre-processing methods used for training an agent to learn to play a game efficiently using Deep Q-networks(DQN) and its variants [3]. To address the sample inefficiency of using Deep RL, we have explored ways such as limiting the number of epochs during training and tuning the hyperparameters to address the overestimation problem. Reproducing results from experiments in Reinforcement Learning are tricky: performance can be noisy, algorithms have many moving parts which allow for subtle bugs, and many papers don't report all the required tricks[1]. For this very reason, we have utilized the OpenAI baselines and Gym environment[2] to test our methods and it functions as a benchmark dataset to validate our claims. On training CartPole and Acrobot [4] using DQN and A2C algorithms, we found that Dueling Double DQN outperforms the others. We tuned the hyperparameters such as learning rate, batch size and buffer size to study its effects.

In this project[3], we have performed extensive experiments to tap into state-of-the-art deep RL algorithms. We began with tuning the hyperparameters of DQN to examine the effect of learning rate, batch size and buffer size on the algorithm performance. Following this, we compared the DQN with DQN variants in terms of its sample complexity, including Double DQN, Dueling DQN, and Dueling Double DQN. We further compared DQN with another type of algorithm, namely A2C to scrutinize the differences. Our experiments reveal the impressive results such as 1) Sample complexity of DQN and its variants is measured in terms of the reward; 2) Performance of A2C deteriorates by a large margin with change in hyperparameters; 3) DQN and its variants are more suitable for CartPole and Acrobot compared with A2C.

Detailed project description

1 Introduction

Over the past few years, the field of deep reinforcement learning (RL) has become increasingly popular due to its success in addressing challenging sequential decision-making problems. The main idea of RL is that an artificial agent tries to learn the acting policy from the experience gathered by interacting with its environment to optimize some objectives given in the form of cumulative rewards. Combining reinforcement learning and deep learning, deep RL has achieved success in domains such as games, robotics, computer vision, finance, self-driving and natural language processing in recent times [1].

[1] https://github.com/openai/baselines
[2] https://gym.openai.com/
[3] https://github.com/Nusha97/Deep-RL-for-Atari-Games

One of the earliest and biggest successes of deep RL is deep Q-learning (DQN), which opens up the possibility for an agent to successfully learn from inputs in high-dimensional space like visual perceptual images [3]. The authors developed a deep convolutional neural network trained with a variant of Q-learning as a novel agent to learn the policy end-to-end. Further progress has been made through the proposed approaches of other deep RL algorithms such as the asynchronous advantage actor-critic (A3C) [5].

Our study is geared towards exploring the applicability of DQN, its variants and A2C algorithms on CartPole and Acrobot. The results from the experiments will give a clear idea of the performance of different algorithms on each game and our analysis is a major contribution to confirm an algorithm's suitability to different games.

2 Algorithm background

The problem setup of RL can be summarised as follows: an assortment of ”actions” ($a \in A$) for an ”agent” learned from ”rewards” ($r \in R$) gained via an ”environment” with many states ($s \in S$). When positioned within the environment, the agent chooses the action to switch between states and will receive the corresponding reward. The speciality of RL is that it only has relatively simple feedback for the agent to actively adapt to the environment to maximize future rewards.

To understand RL algorithms, we introduce two essential components: value and policy. The RL environment can be viewed as a sequential model with transitional states which takes actions as input and produces rewards as output. Thus, the value function $V(s)$ is the expected amount of rewards of each state, and the agent's policy $\pi(s)$ is the agent's probability of producing actions in each state. RL algorithms are categorized as on-policy and off-policy according to their use of deterministic outcomes from the target policy to train the algorithm [1]. The whole RL process can be summarized by the below episode:

$$\text{process} : S_1, A_1, R_2, S_2, A_2, \ldots, S_T \quad \text{value} : \overset{\frown}{S_1, R_2} \quad \text{policy} : \overset{\frown}{S_1, A_1}$$

In this project, we examined two types of RL algorithms: deep Q-learning (DQN) and Advantage Actor-Critic (A2C). According to traditional RL algorithm classification, DQN is a value-based approach and A2C is a policy-based approach. We introduce their algorithmic background separately in the following section.

2.1 DQN: a value-based approach

It is important to first understand the value-based RL approach and Q-learning. Relying on an incomplete piece of episode, the prediction of value-based RL algorithms is updated according to only the reward and the difference between the value function of S_t and the value function of S_{t+1}, which is $V(S_t) \leftarrow R_{t+1} + \gamma V(S_{t+1})$. Replacing the vanilla value function with a function of state-action combinations, namely the Q value, and replacing the deterministic component $Q(S_{t+1}, A_{t+1})$ with stochastic component $\max_{a \in A} Q(S_{t+1}, a)$, the prediction update procedure equals $Q(S_t, A_t) \leftarrow (1-\alpha) Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a))$. This is the update method of **Q-learning** [6]. With this stochastic part as its component, the biggest challenge Q-learning faces is the size of the memory pool of Q value estimations of all state-action pairs.

Whereas in **deep Q-learning** (DQN) [3], a deep convolutional neural network has been trained as a novel agent of Q-learning to learn the policy end-to-end, which solved the memory pool size problem of Q-learning via deep learning. The DQN algorithm learns from nothing but the video input, the reward, terminal signals, and the set of possible actions like a human player would, and it can generalize to successfully learn to play many games with a single neural network model. The network is updated by stochastic gradient descent with the raw pixels as the input and a value function estimating future rewards as the output.

Double DQN (DDQN) [7] is a performance-based variant of DQN based on the same architecture avoiding the overoptimistic value estimates caused by selecting overestimated action. DQN uses the same value to do both the selection and evaluation, while DDQN decouples them with online network selecting and target network evaluating actions. The **dueling** architecture [8] is a complexity-based variant of DQN by separating the state values and state-dependent actions into two streams. Through this separation, the dueling architecture is able to return only valuable actions instead of all actions, and thus reduces memory cost. The combination of dueling architecture and double DQN is called **Dueling Double DQN**.

2.2 A2C: a policy-based approach

While value-based RL aims to learn the value function with ”nearby” states and rewards, another branch of RL method is based on policy gradient methods, which learns the policy directly without using the value function and instead applies the gradient ascent technique to find the highest reward. With functions randomly initialized, the policy function is updated via gradients according to the divergence that exists between estimated rewards and real rewards. **Advantage Actor-Critic** (A2C) [5] is based on this idea: the actor learns the optimal policy via traditional gradient method, and the critic evaluates the action and computes the value function as an auxiliary branch. Without the need to memorize all state-action combinations, policy-based approaches may be more computationally efficient than value-based approaches [5].

3 Game Environments and Technical Details

Atari 2600 games are common testbeds of deep RL algorithms, as shown in the original DQN, DDQN, and A3C papers [3,5,7]. In full-scale deep RL research, many computational resources are generally used in order to train models (for example, the DQN paper proposes a network on a single GPU for approximately one week for each game studied). In order to perform our project experiments, we utilize Google Colaboratory, and focus on two games available in OpenAI Gym with relatively low computational overhead: CartPole and Acrobot. Both games are motion simulators which involve some understanding of physics principles and gravity. We additionally use and modify the algorithm implementations of DQN and A2C available in OpenAI Baselines for all experiments.

CartPole A vertical pole is attached to a moving cart through an under-actuated joint. The pole must be balanced by applying a force on the cart which is constrained to move in only one dimension. The game can be characterized by four state variables namely: x - position of the cart in the 1D track, θ - angle made by the pole with the vertical, $\dot{x}$ - cart velocity, and $\dot{\theta}$ - rate of change of the angle [9]. The game is initialized with the pole in the vertically upright position. The possible allowed actions are pushing the cart to the left or right, and the objective is to prevent the pole from exceeding an angle of 12 degrees from upright. The observed state includes information such as the velocity and position of the cart, the angle of the pole, and velocity of the tip of the pole. A reward of ’1’ is earned for every step taken. The episode ends if the one of the following criteria are satisfied: the pole angle exceeds 12 degrees, the cart has moved more than 2.4 units from its initial position, the episode length is greater than 200, or the average reward achieved is greater than 195.0 over 100 consecutive trials.

Acrobot Originally studied to observe task dynamics and convergence on control problems [10], Acrobot consists of one actuated joint connecting two links and an under-actuated joint that holds the upper link in position. The lower link is free to swing about the actuated joint and the two joints do not collide. The goal is for the lower link to rise to a particular height. The observed state consists of the *cos* and *sin* of the two rotational joint angles, as well as joint angular velocities. Here allowed actions are to apply either a torque of ’-1’, ’0’ or ’1’ to the actuated joint. A reward of ’0’ is awarded if the joint is able to swing to a certain height otherwise a negative reward ’-1’ is awarded. The aim is to achieve a 0 total reward; games begin with -500 reward.

4 Results and Discussion

Our results are divided into three sections. We first report the effects of hyperparameter tuning (learning rate, batch size, and buffer size) on vanilla DQN. Secondly, we investigate sample complexity (the number of episodes needed to reach maximum reward) and the optimality of reward as a means to compare DQN, its variants, and A2C. Lastly, we compare the algorithms across the two different game environments.

4.1 Learning rate and batch size: hyperparameter tuning in DQN

In this experiment, we establish the relationship between the learning rate and batch size with the reward gained at the end of the episodes of training. We have also computed the time taken to completion of training (in seconds). Our results are listed below:

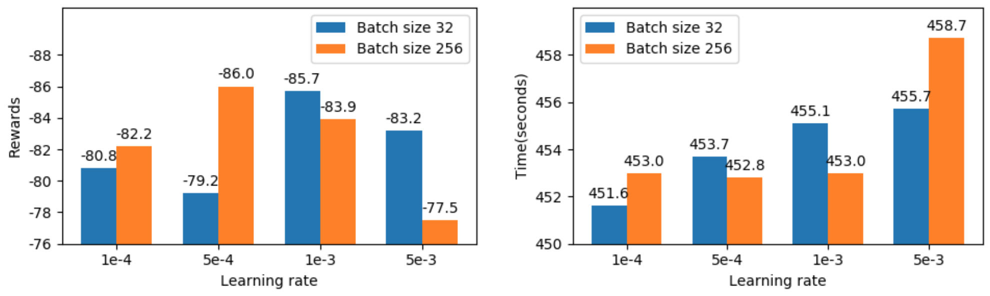

Figure 1: Figure comparing the rewards obtained and training time required for vanilla DQN on Acrobot.

We observed that a good combination of batch size and learning rate is important for the performance of DQN, though their individual changes do not show meaningful trajectory to cause a significant difference in reward optimality. They also have a directly proportional relationship: when the learning rate is low, a lower batch size is optimal; when the learning rate is high, a larger batch size is optimal. However, the overall difference between the rewards is small, and there is a non-linear change of model performance as learning rate changes. As for training time, generally speaking, we observe that the bigger the learning rate the longer the time it takes for the algorithm to converge, though the difference is small because of the use of a light-weight game. However, the differences between different episodes are comparably small. It is highly possible that this is due to the low complexity of the game.

We can additionally see the effect of learning rate on sample complexity from the results in Figures 2 and 3. When the learning rate is low enough, the learning path is steady and maximum reward is firmly achieved. As the learning rate becomes higher, the learning process becomes bumpy. Batch size of 32 is slightly better for Vanilla DQN, and batch size 256 for the other algorithms.

4.2 Sample complexity: comparison between DQN, DQN variants, and A2C

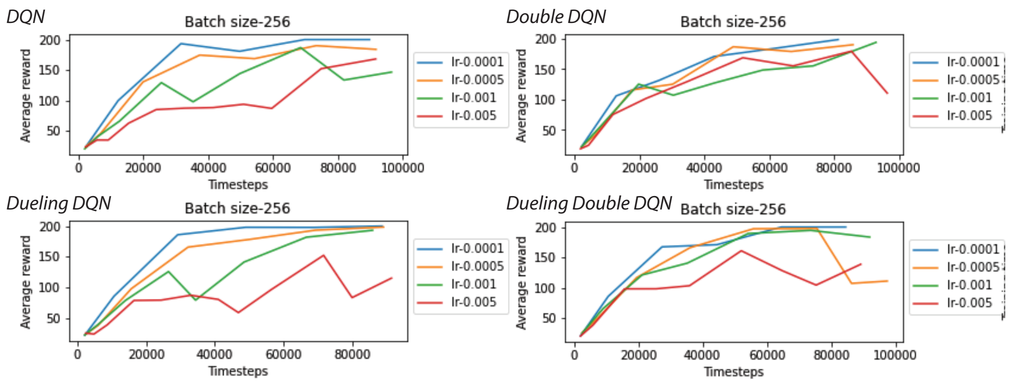

Figure 2: Plot showing average rewards gained by DQN, Double DQN, Dueling DQN, and Dueling Double DQN for different learning rates on CartPole. More training results can be found in the Appendix.

Average episode reward as a function of time can be found in Figures 2 (for DQN and DQN variants) and 3 (for A2C). Sample complexity can be measured in terms of number of iterations required by the algorithm to reach a state where the environment is considered solved. For our experiments this state is quantified by the reward achieved by an algorithm for each of the games it is trained on. Higher number of data samples correspond to learning that happens over an increased number of epochs to reach an optimal reward. For Vanilla DQN trained on Cartpole, a maximum reward of 200 is achieved over 69700 timesteps. Sample complexity is improved for DDQN, Dueling DQN and Dueling DDQN as they are able to achieve max reward in 67600, 68000 and 44500 timesteps respectively (all with a learning rate of 0.0001). For Acrobot, the maximum reward is achieved at about 57600, 58600, 49600, 51100 timesteps with Vanilla DQN, DDQN, Dueling DQN and Dueling DDQN respectively. For A2C, the models took far longer to reach optimal reward (1152000 timesteps for CartPole and 9836800 for Acrobot).

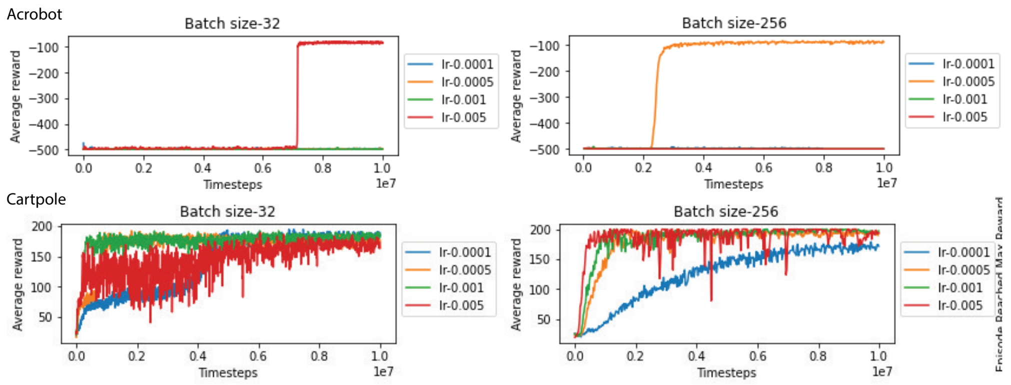

Figure 3: Plot showing average episode reward gained by A2C on CartPole and Acrobot for different learning rates and batch sizes.

On studying the effect of buffer size on training, it is observed that fastest convergence to the maximum average episode reward is achieved with a buffer size of 1000. We also observe that the larger the size of buffer, the longer the training time required. It can be expected that a larger buffer means more memory and slower training, but it is less likely that correlated sample elements are selected, which is beneficial to the training stability. However, an excessively large buffer can take longer time to become refreshed with new good trajectories. According to our results, the buffer size of 1000 is best choice for both games of CartPole and Acrobot considering the trade-off of training time, stability and efficiency, as shown in Figure 4.

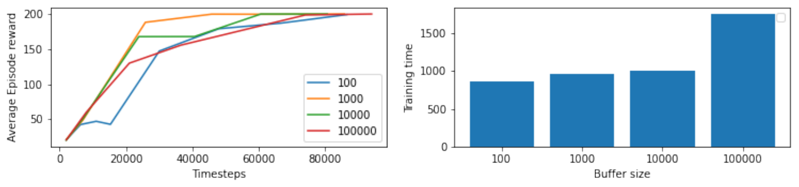

Figure 4: The left plot shows model's (Dueling DDQN on CartPole) average episode reward for different buffer sizes, lr 0.0001 and batch size 256. The right plot shows the increase in training time(sec) with increase in buffer size used for prioritized replay. Results on Acrobot can be found in the Appendix.

4.3 Comparison of Algorithms and Games

Overall, the vanilla DQN and variants vastly outperforms A2C on Acrobot. Additionally, A2C is *extremely* sentitive to hyperparameters, and can only do well given certain combinations (see Figure 3; note that in some cases models never even beat the default -500 reward). For Acrobot, only the hyperparameter combination of (batch size 256, learning rate 0.0005) excels at reaching optimal reward with A2C. Among DQN and its variants, we observe that Dueling Double DQN outperforms on CartPole in terms of sample complexity and maximum reward gained. This is evidence that algorithmic improvements like avoiding overoptimistic value estimates and advantages of dueling architecture and prioritized replay functionality are helpful. We hypothesize a few reasons why A2C may have performed so poorly: 1) The algorithm depends more on random initialization than DQN, 2) Q-learning may be more appropriate to small-scale games since the approximation of state-action combinations is limited, and 3) The use of policy gradients in A2C could result in vanishing gradients with changes to the learning rate. Additionally, due to computational limitations, we were unable to search a larger subspace of hyperparameter values as the paper authors.

5 Conclusions and Contributions

This project examines both the theoretical and experimental study of DQN, its variants and A2C to present a thorough view of the pros and cons of the different deep RL algorithms. The idea for this project was conceived as a result of multiple brainstorming sessions over a span of weeks with the sole motivation of exploring and expanding our knowledge base in the field of Reinforcement Learning. We have successfully translated our findings through this meticulous report which is a significant contribution to the community developing algorithms for different games. As a team of 5 members working remotely, we had a well-crafted plan to divide the work. The careful survey of the current deep RL algorithms was carried out by Ran and Anusha. The design of elaborate experiments to examine and compare the performances of the various algorithms for different games was led by Akshata, Anni and Sarah. At the completion of each phase, we connected and gave each other feedback as a team which has resulted in a comprehensive review of deep RL algorithms and novel observations about the challenges in this field which would be beneficial for the RL community.

References

[1] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, pp. 26–38, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, pp. 529–533, 2015.

[4] A. Hill, A. Raffin, M. Ernestus, Gleave, A. Kanervisto, R. Traore, P. Dhariwal, Hesse, O. Klimov, A. Nichol, M. Plappert, Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, pp. 279–292, 1992.

[7] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[8] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pp. 1995–2003, 2016.

[9] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 834–846, 1983.

[10] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, (Cambridge, MA, USA), p. 1038–1044, MIT Press, 1995.

Appendix

More Details on Algorithms

In the DQN algorithm, the environment ε, in this case the Atari emulator, is a sequence of actions (legal game actions $A = \{1, \ldots, K\}$), observations (image from the emulator $x_t \in \mathbb{R}^d$), and rewards (change in game score r_t). They consider the sequences of actions and observations as state $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, and define the future discounted return at time t as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}. \tag{1}$$

The optimal action-value function is defined as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a, where π is a policy mapping sequences to actions. With Bellman equation[4], the optimal action-value function can be converted to

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathscr{E}} \left[r + \gamma \max_{a'} Q^* \left(s', a' \right) | s, a \right]. \tag{2}$$

where r is the reward, γ is the discount factor. The DQN refers to a non-linear function (i.e. a neural network function with weights θ) approximator to estimate the Q value. The Q-network is trained by minimizing the loss function,

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^- \right) - Q\left(s, a; \theta_i \right) \right)^2 \right]. \tag{3}$$

To make the training more stable, the target network (output $Q(s', a'; \theta_i^-)$), which is a periodic copy of the online network (output $Q(s, a; \theta_i)$), is proposed to "delay" the update of parameters. The experience replay mechanism, the other key point of the algorithm, is a memory that stores the most recent states. It is employed to solve the challenges of highly correlated states and dynamic data distributions.

Double DQN (DDQN) [7] is an improved version of DQN (based on the same architecture) by avoiding the overoptimistic value estimates caused by selecting overestimated value. In other words, DQN uses the same values both to select and to evaluate an action, while DDQN decouples them with the online network (with θ_t) selecting an action and the target network (with θ_t^-) evaluating an action:

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a; \theta_t^- \right) \tag{4}$$

$$Y_t^{\mathrm{DoubleDQN}} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a}{\mathrm{argmax}} Q\left(S_{t+1}, a; \theta_t \right), \theta_t^- \right). \tag{5}$$

[4]Bellman equation: if the optimal value $Q^*(s', a')$ of the sequence s' at the next time-step was known for all possible action a', then the optimal strategy is to select the action a' maximizing the expected value of $r + \gamma Q^*(s', a')$.

Results from Experiments(Not Included in the Paper)

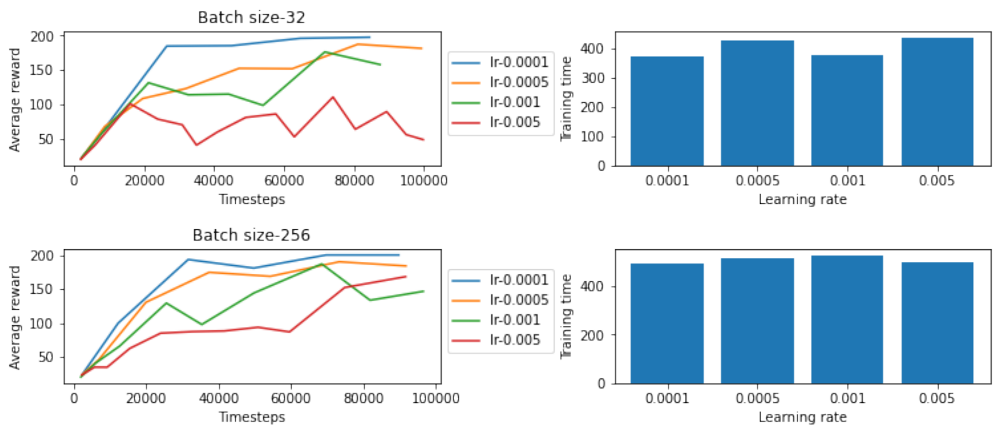

Figure 5: Figure comparing the rewards obtained and training time required when training vanilla DQN on the game CartPole with different batch size and learning rate.

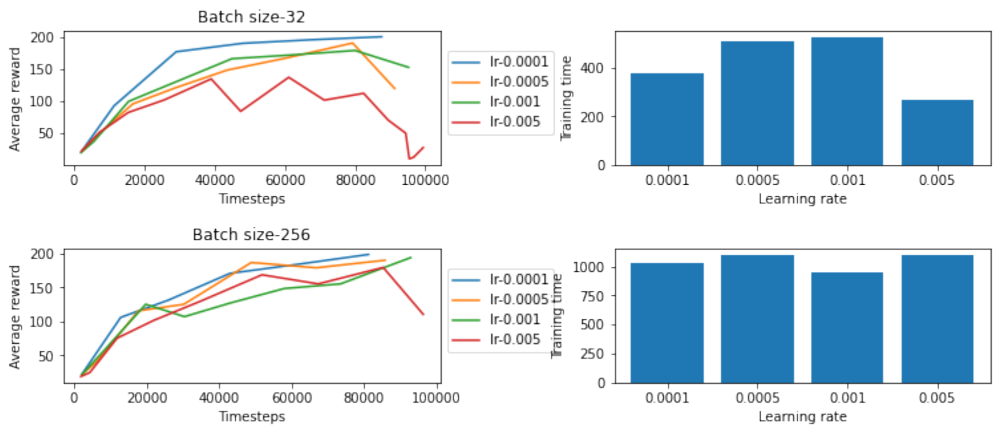

Figure 6: Figure comparing the rewards obtained and training time required when training double DQN on the game CartPole with different batch size and learning rate.

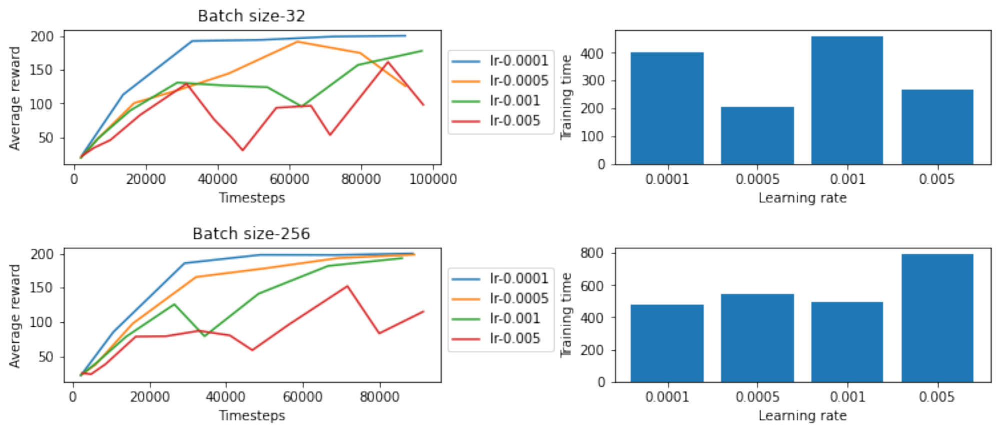

Figure 7: Figure comparing the rewards obtained and training time required when training dueling DQN on the game CartPole with different batch size and learning rate.

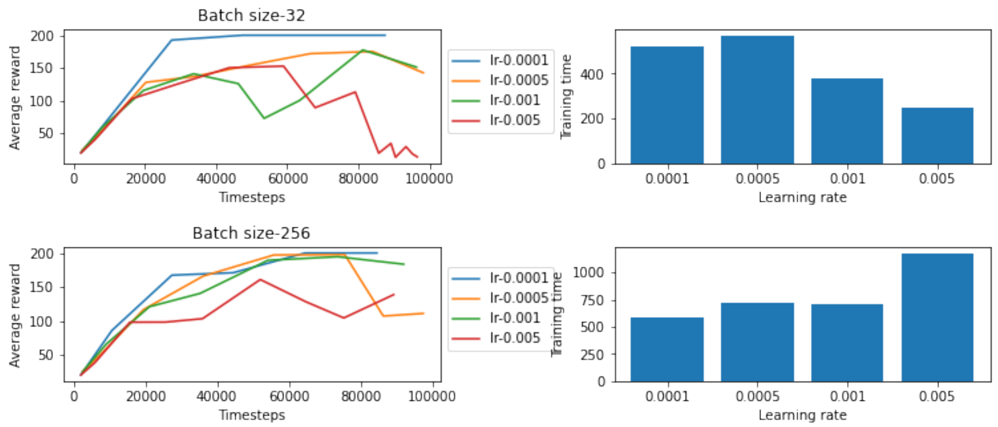

Figure 8: Figure comparing the rewards obtained and training time required when training dueling DDQN on the game CartPole with different batch size and learning rate.

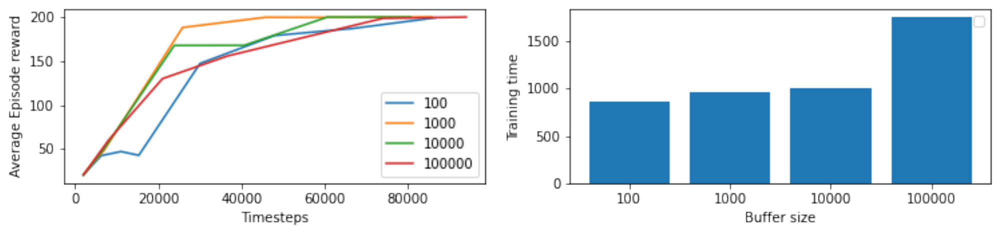

Figure 9: The rewards obtained and training time required when training dueling DDQN on the game CartPole with different buffer size, learning rate 0.0001 and batch size 256.

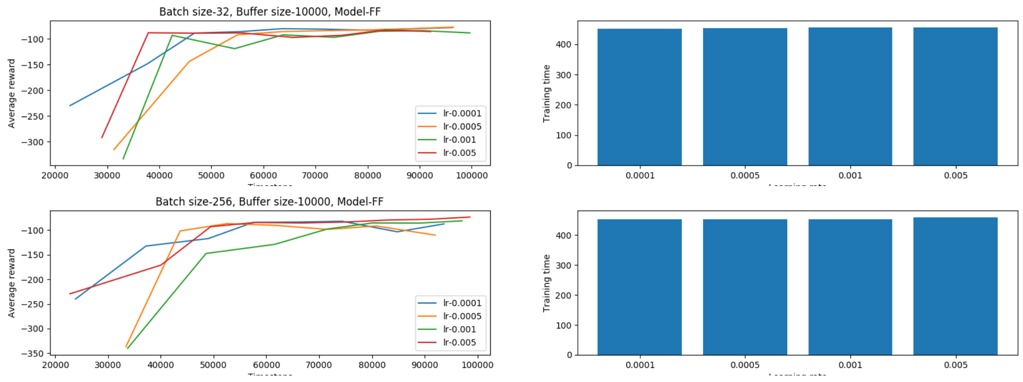

Figure 10: Figure comparing the rewards obtained and training time required when training vanilla DQN on the game Acrobot with different batch size and learning rate.

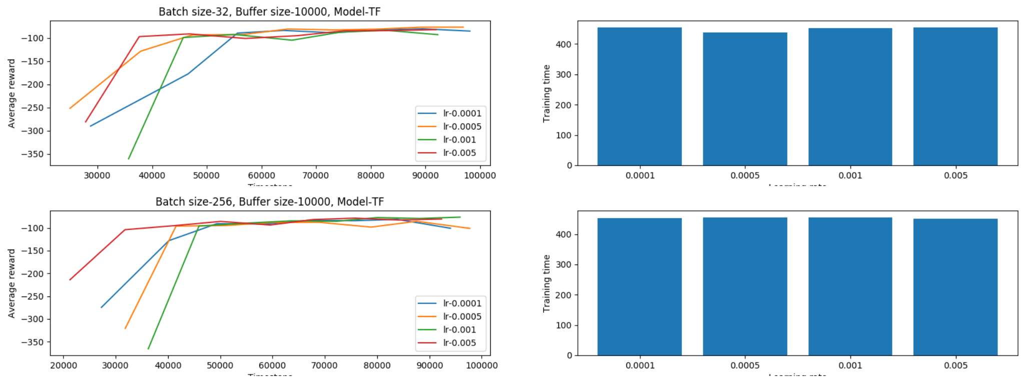

Figure 11: Figure comparing the rewards obtained and training time required when training double DQN on the game Acrobot with different batch size and learning rate.

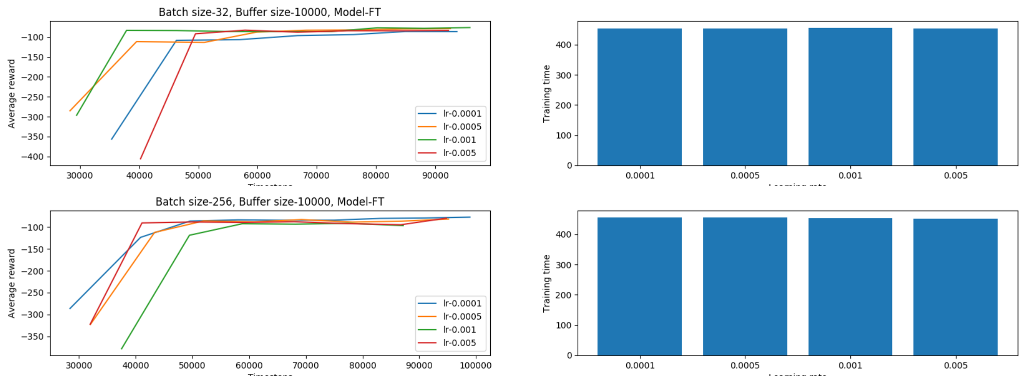

Figure 12: Figure comparing the rewards obtained and training time required when training dueling DQN on the game Acrobot with different batch size and learning rate.

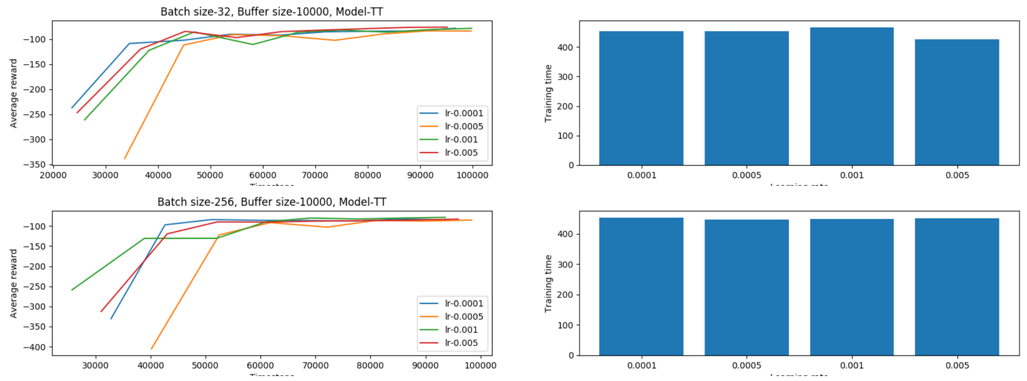

Figure 13: Figure comparing the rewards obtained and training time required when training dueling DDQN on the game Acrobot with different batch size and learning rate.

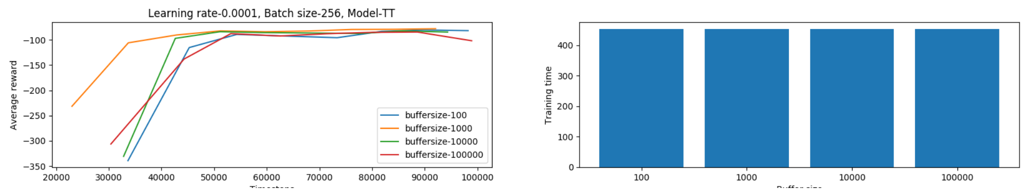

Figure 14: The rewards obtained and training time required when training dueling DDQN on the game Acrobot with different buffer size, learning rate 0.0001 and batch size 256.

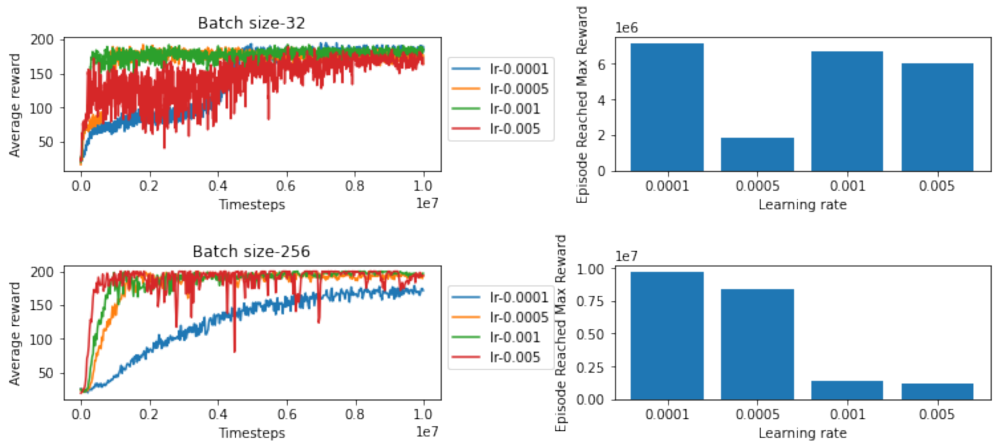

Figure 15: Figure comparing the rewards obtained and training time required when training A2C on the game CartPole with different batch size and learning rate.

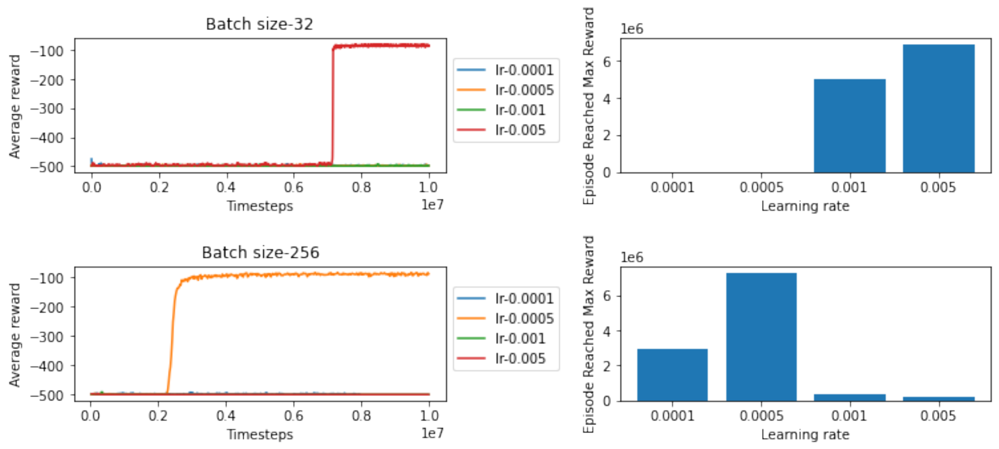

Figure 16: Figure comparing the rewards obtained and training time required when training A2C on the game Acrobot with different batch size and learning rate.