A Comparative Study of Model-Free Reinforcement Learning Approaches

Anant Moudgalya[1], Ayman Shafi[2], and Amulya Arun B[3]

PESIT South Campus, Bengaluru, Karnataka, India

Abstract. This study explores and compares 3 model-free learning methods, namely, Deep Q-Networks(DQN), Dueling Deep Q-Networks(DDQN) and State - Action - Reward - State - Action (SARSA) while detailing the mathematical principles behind each method. These methods were chosen as to bring out the contrast between off-policy(DQN) and on-policy(SARSA) learners. The DDQN method was included as it is a modification of DQN. The results of these methods and their performance on the classic problem, CartPole were compared. Post training, testing results for each of the models were as follows: DQN obtained an average per episode reward of 496.36; its variant and improvement, DDQN obtained a perfect score of 500 and SARSA obtained a score of 438.28. To conclude, the theoretical inferences were decisively reaffirmed with observations based on descriptive plots of training and testing results.

Keywords: Reinforcement Learning · Q-Learning · DQN · DDQN · SARSA.

1 Introduction

Reinforcement Learning(RL) refers to mapping scenarios to actions, with the purpose of increasing a reward [1]. The RL problem involves a goal-directed agent which interacts with an unknown environment, by choosing actions to be performed on the environment, resulting in a change of state. Each new state has an associated reward, which is used by the agent to make decisions, in order to maximise accrued reward scores. RL is distinctly different from other forms of learning in that the agent starts with little or no awareness of the environment but learns through conversance with it. One of the main challenges faced in RL is maintaining a balance between exploration and exploitation. The agent needs to be able to determine when to exploit what it already knows and when to explore other options to make significantly better action choices in the future [1]. This form of learning based on actions and rewards/punishments originates from behavioral psychology and its analysis in humans and animals.

Deep Learning architecture when combined with principles of RL results in what is called Deep Reinforcement Learning(Deep RL). This combines the features of RL with the capability of deep architecture to progressively extract higher level features from the input data. Recently, Google DeepMind developed

the Deep Q-network (DQN), a novel deep neural network architecture, that has been shown to be proficient in learning(and surpassing) human-level control strategies on a collection of different Atari 2600 games [2].

In this report, attempts have been made to understand how Deep RL can be applied and how to overcome the challenges faced. DQN [3] and its variants have been used in an attempt to compare results and performance statistics. Results of these algorithms are used on the environment, "CartPole-v1" of the OpenAI Gym as a common measure of performance. An implementation of DQN is studied, and the architecture is modified to form a Dueling DQN (DDQN) [4], then SARSA(State - Action - Reward - State - Action) algorithm is implemented. A comparative study of these architectures is performed and the experimental results are analysed.

2 Related Work

2.1 Reinforcement learning

A Reinforcement Learning(RL) task deals with training an Agent in an Environment. The Agent performs actions to transition between different scenarios of the Environment, referred to as states. Actions, yield rewards, which could be positive, negative or zero. The purpose of an Agent is to maximize the total reward it accrues over an episode. Hence, the Agent is reinforced to perform certain actions by providing it with favourable, mathematically significant rewards, and to avoid other actions by providing unfavourable rewards. This is how an Agent learns to develop a a policy/strategy. RL has many learning methods, but only model-free learning methods have been explored as they are robust and independent of an environment description.

There are two types of learning within RL based on policy, an off-policy learner derives the optimal policy irrespective of the Agents actions, provided it explores to a good extent. If the learning Agent chooses only the best action learned up to that point, it may not explore to an extent enough to obtain the optimal action on the whole. The policy value is not learned in off policy learning because exploration steps are included in the policy. On the other hand, in on policy learning, the policy that the agent was following, specified previously, is learned. Deep RL has been used in the past for surpassing human scores on Atari games, few of them are, Stadie et al. [5]; Mnih et al. [2]; Guo et al. [6]; Bellemare et al. [7]; Nair et al. [8]; Schaul et al. [9] and van Hasselt et al. [4]. This study incorporates knowledge gained from their experiments, and puts it to good effect while performing the experiments and inferring from results in the sections that follow.

2.2 Q-Learning

Q-Learning is a Model-Free RL algorithm, meaning, it does not require a model/description of the environment to be applied [10]. The main motive of Q-Learning is to help

an agent formulate a strategy, which will help determine the most suitable action to be taken from any state. Q-Learning as a technique maximizes potential cumulative reward to all future states, from the current state [11]. To better understand, let us consider a state S, an action A, and the associated reward for that action as R. The relationship between Q, S, A and R can be denoted as follows:

$$Q : S \times A \rightarrow R \tag{1}$$

Before learning begins, the Q-value is randomly assigned to a fixed, initial value, chosen by the programmer. At timestep t, the agent chooses action a_t from the set of valid actions, A = {0,1...K}. Each action a_t has an associated reward r_t. For every such action performed from state S_t to state S_{t+1}, Q-Learning performs an iterative value update using a weighted average of current and old state information, as follows:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(S_{t+1}, a)] \tag{2}$$

The learning rate, denoted by α in the given equation(2) [1], denotes the rate with which the agent learns new things about the environment. If $\alpha = 0$, then the agent essentially learns nothing, but in practice, α is generally considered to be a constant value, such as 0.1. The discount factor γ is the measure with which the agent prioritises short and long term rewards, i.e, with a smaller discount factor, or with $\gamma = 0$, the agent always takes short-term rewards. Similarly, with a high discount factor, or $\gamma = 1$, the agent will make moves for long-term high rewards. Short term rewards are when the agent picks the move that provides for instant maximum rewards, whereas long term rewards are when the agent makes moves that compromises highest possible reward at current state, so that it can obtain higher rewards at a later state.

Generally, the basic implementation of Q-Learning is with a Q-Table, which serves as storage for Q-values of state-action pairs. But with an increase in the size of state/action spaces, this approach is expensive and inefficient. Therefore, this overhead can be reduced by combining Q-Learning with function approximation. A good function approximator to speed up learning in finite spaced environments is an Artificial Neural Network.

2.3 Deep Q-Networks

Deep Q-Networks(DQN) as designed by DeepMind [3] used Convolutional Neural Networks (CNN) to filter out frames of Atari games as game memory and use the consequent Neural Network as function approximator.

DQNs also use a technique called Experience Replay, a biologically inspired process that uniformly samples experiences from memory instead of the most recent action and updates their Q-value. Experience Replay and a memory replay buffer helps in maintaining a balance between exploration and exploitation for the agent. Nevertheless, DQNs and Experience Replay tend to falter in environments with sparse rewards. The solution for this problem was proposed by Nair et al. [12], the depths of which are not explored in this paper.

Since Q-values are being updated in batches from previous experience, it allows for better efficiency and makes data distribution more uniform. As per the Algorithm Deep Q-Learning with Experience Replay [3], only the last N state-action tuple values are being stored in the experience-replay buffer and sample D number of tuples randomly. The algorithm can be understood in-depth as explained in [3]

2.4 Dueling Deep Q-Networks

It has been observed in DQNs that the Q-value is assigned for particular state-action pair, and that it determines the quality of the state and the action that can be taken from that state. In the architecture for Dueling Deep Q-Networks(DDQN), two separate estimators are represented, a state-value function and an action-advantage function [13]. Therefore, for state S and corresponding action a, the computation of Q-value can be divided into two different, Value(V) and Advantage(A) streams.

$$Q(S, a) = A(S, a) + V(s) \tag{3}$$

The sum of these two functions is what gives the correctly estimated Q-value for that particular state-action pair. Two networks can be used for these two functions and a special aggregation layer that results in the Q-value. By decoupling the estimation, the DDQN can begin to perceive which states are worthwhile without needing to learn the impact of every action at every single state. (since it's also calculating V(s)). In DQNs, the small Q-updates can cause noise and disruption in the order of actions, which can cause the policy to make drastic, inefficient decisions. But since Advantage values are being calculated separately, the dueling architecture is robust to such circumstances.

2.5 SARSA

State-action-reward-state-action (SARSA) is an algorithm in RL, extensively used for learning a decision strategy inspired by the mathematics behind Markov chains. [14]. In SARSA, the Q-value is derived from the current game-state(S_1) of the agent, the game-action chosen by the agent(A_1), the reward score(R) associated with that game-action, the new game-state(S_2) entered by the agent after taking the game-action and the next game-action(A_2) chosen by the agent from its new game-state. The acronym for the quintuple,

$(S_t, A_t, R_t, S_{t+1}, A_{t+1})$ is SARSA [15]. As opposed to DQNs SARSA is an on-policy learning algorithm. It is so called because at every gamestep, the agent updates the currently followed policy based on its interchange with the game environment for the corresponding action in that game-step.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{4}$$

The Q-value for a state-action pair is updated at each step by a value, which corresponds to the resultant error. This is adjusted by using the learning rate

α. Each Q-value is a combination of both the probable reward score that will be obtained in the following game-step, for taking the game-action A in the current game-state s, and the discounted/adjusted future reward score that it receives from the subsequent state-action pair, as explained in the works of Perez et al. [16].

In SARSA the Q-value is updated taking into account the action A performed in state S. This is in contrast to Q-learning, where the action with the highest Q-value in the next state S is used to update the Q-value. The basic difference between SARSA and Q-learning is that SARSA agent chooses its action based on the same current policy and updates its corresponding Q-values, whereas the agent in Q-learning chooses the action which gives maximum Q-value for the state.

3 Experiments and Results

3.1 Model Architecture and Result Metrics

The models used in this project were taken from the keras-rl [17] repository on GitHub. The model architecture is identical for all 3 algorithms barring intricate differences in implementation. On the whole, as an alternative to a Q-table (which maps current state to next state of highest reward), function approximators are used. In this case, the function approximator used is an artificial neural network. Before analysing the results of the 3 algorithms, the following are the meanings of a few important terms.

Reward is a term that denotes feedback obtained from the environment. Reward can be positive or negative. Normally, the aim is to maximise the reward obtained throughout the course of an episode.

Mean-Q value refers to the mean of Q-values (denoted by $Q(S, a)$, which is a computational estimate of the anticipated reward assuming that the Agent is in state S and performs action a, and then plays until the end of the current episode following some policy).

The hyperparameters of the model were set as follows, $\alpha = 0.1$, $\gamma = 0.99$. The model was trained for 50,000 game steps as a limit, where any number of steps can occur in an episode, from initial state to terminal state. The rewards per episode during training were observed and plotted, as shown in Fig 1

From the graph in fig 1, The validity of the mathematical functions discussed in the previous sections can be observed. It can be concluded that DQN takes the least number of episodes to conclude training and reach max reward per episode. Duel DQN takes the second most number of episodes, more than DQN because of the duel estimation architecture. It can also be seen that although Duel DQN takes longer to conclude training, it learns better, i.e, experiences episodes with bigger rewards faster than DQN, and during the end of the training period, rewards are consistently higher. SARSA on the other hand, being an on-policy learning method, does not prioritise cumulative rewards, and hence takes longer

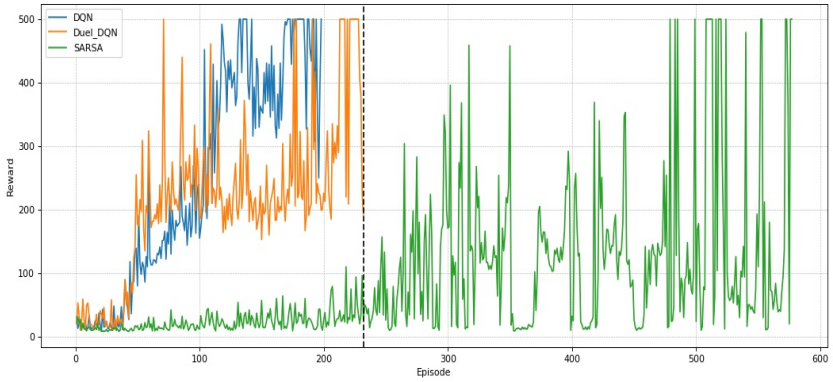


Fig. 1: Rewards per Episode across all methods

to learn. Even during the end of the training period, rewards per episode are not consistently high.

The next result metric in the observations is Loss per Episode during training. Training loss in most Deep Learning algorithms can be plotted and inferences can be made to decide upon the quality of the said algorithm. The loss plotted for each algorithm is as shown in Fig 2

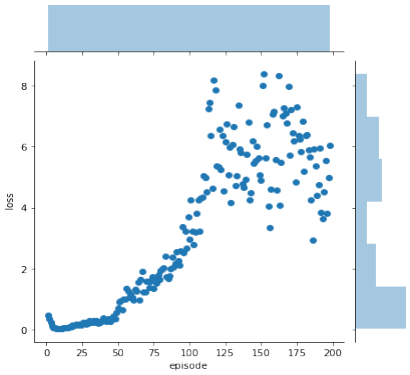
From the scatter plots, the Loss per Episode can be observed during training for the first method, a DQN. Training loss is generally high in off-policy RL algorithms as the agent must explore the environment first and exploit its past experiences eventually.

For DQN (2a), it can be observed that training loss is high as the agent explores the environment, gradually forming the policy. Also, from the plot, it is concluded that the range of values of training loss is large and that the algorithm incurs heavy losses at some points during training. Once a policy is formed, loss per episode decreases as the policy chooses the optimal Q-values and prioritises cumulative rewards.

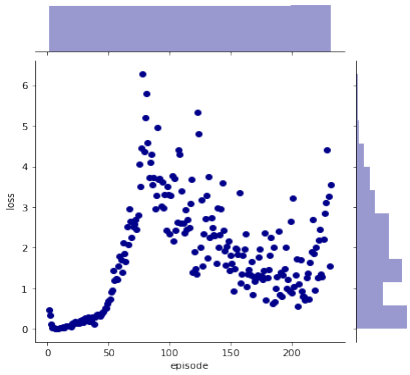
Next the graph for Loss per Episode for DDQN is observed (Fig 2b) which is a variant of the DQN. It is already noted that it uses the Dueling Network architecture and that it takes more number of episodes than DQN during training.

It is observed that the loss pattern for DDQN is similar to that of a DQN, only difference being the policy is formed faster and there is a sharper decrease in loss after formation of the aforementioned policy. It is also observed that the range in training loss values is lesser compared to DQN.

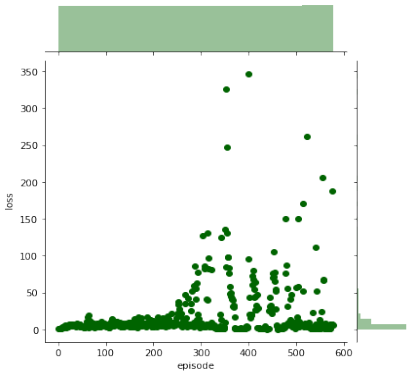
The variation in training loss for SARSA can be observed from the plot in (Fig 2c). The general pattern of training loss is very low, this is because SARSA



(a) Loss Per Episode-DQN



(b) Loss per Episode-DDQN



(c) Loss per Episode-SARSA

Fig. 2: Loss per Episode graphs

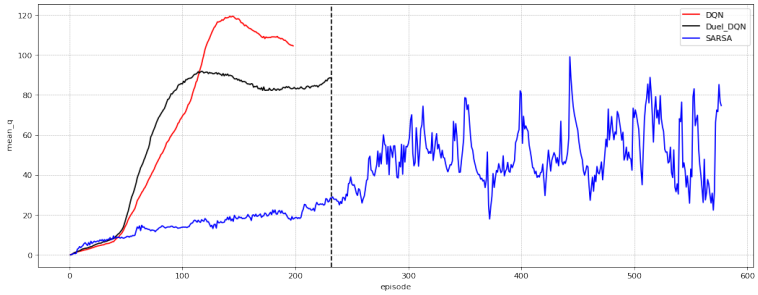


Fig. 3: Mean Q-values across all methods

is an on-policy learning method. Therefore, unlike DQNs and DDQNs, SARSA does not incur losses during the policy-formation phase, as it follows a policy from the first stage. Loss incurred may become very high (Occurrence of 350 in (Fig 2c)) as the agent may sometimes not obey the policy in every episode and therefore, suffers great loss for that episode.

Next, the Mean Q-value per Episode is observed during training for all 3 algorithms. It is observed that the behaviour of Q-values aligns with the mathematical equations as described above. Hence, from the graph it is inferred that Q-values for a DQN increases during exploration, as the Q-values for state-action pairs are updated. It is also noted that the Q-value stabilises, and also decreases after a point, which is at the completion of exploration, as the actions are chosen according to the maximum Q-value for that state in each particular episode.

Duel DQN behaviour is also similiar to DQN, except it again confirms that exploration phase is completed quicker as the stabilisation on Mean Q-value per Episode occurs earlier than DQN, and the updates to Q-values are minimal after completion of exploration.

Mean Q-values in SARSA on the other hand are gradually increasing in nature which reaffirms the fact that it is follows on-policy learning. The updates to Q-value follow the previously described policy, and that it updates Q-values based on current reward and does not take cumulative rewards into consideration.

Table 1: Result Metrics

Algorithm	Reward	Loss	Mean_Q	Test Reward
DQN	250.08	3.25	64.86	496.36
DDQN	214.31	1.81	63.29	500.0
SARSA	86.56	17.19	35.81	438.28

3.2 Test Results

After 50,000 timesteps of training, the model is trained on 150 episodes, consisting of 500 timesteps each, on the same environment, i.e, CartPole. The Rewards per Episode results were plotted in a graph. It can be observed as shown in Fig 4. Training and testing score averages can be referred to in table 1. After calculations, it is seen that average score obtained for DQNs was 496.36, which is close to almost a perfect score on each episode.

The dashed vertical line at the end of the graph is indicative of the Dueling DQN's test scores. The average Reward per Episode obtained was 500.0. Astonishingly, DDQNs gave a perfect score on all 150 test episodes, conclusively proving that it is invariably an improvement upon the DQN model, beating its near-perfect score to an actual perfect score.

On the contrary, SARSA, being an on-policy learner, produces erratic Rewards per Episode during testing. Similar to the training phase, the Q-values are

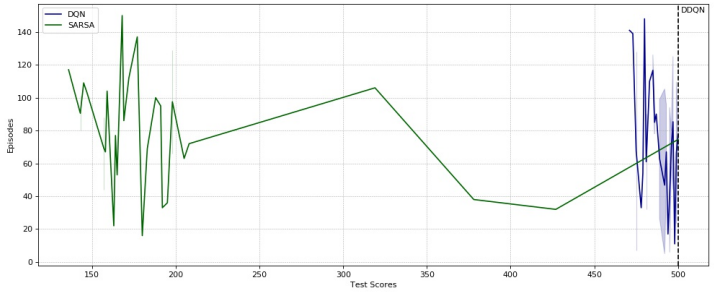


Fig. 4: Test Results for DQN, DDQN and SARSA over 150 episodes

updated according to the policy and not in an optimal manner, and cumulative rewards are not prioritised, therefore a lot of changes in reward score obtained in each test episode can be observed. Despite this, after 50,000 timesteps of training, the model does produce good results in a number of episodes. The calculated average Reward per Episode for SARSA is 438.28. While it may seem tawdry in comparison to the DQN and DDQN, it is still a substantially good score per episode.

Therefore, it is observed that the Agent does well in all 3 environments producing a sufficiently good score, proving that these algorithms can indeed perform as well, if not better, than human level in any environment when trained for an adequate amount of time with fine-tuned hyperparameters. Although these results are observed for the CartPole environment, these results will translate correspondingly with other environments, albeit the probability of achieving perfect scores with expanding state and action spaces may vary.

Finally, we can observe that with these performance characteristics, we can look to their practical applications. RL is already being used to provide value; in Robotics to design and automate control principles, in Chemistry to optimize chemical reactions such as in [18] and in Financial Trading, to make smarter risk-averse trades, as described in [19].

4 Conclusion

This paper consists of detailed observations on the performance of 3 model-free learning algorithms, namely Deep Q-Networks, Dueling Deep Q-Networks and SARSA (State-Action-Reward-State-Action). Multiple inferences are produced from these observations, mainly confirming the validity and performance of an Agent using these 3 algorithms on the selected environment, CartPole. Based on the experimental results, it is conclusively proven that off-policy learning is more effective than on-policy learning and that Duel DQNs are a definite improvement on its primitive form in DQNs. Knowing that these algorithms are model-free, it only engages one's curiosity about its innumerable applications in different

environments. We look forward to learning more and hope to make a significant contribution to the field.

References

1. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction. MIT press*, second edition ed. The MIT Press, 2014,2015.
2. 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. Legg1, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature, 518 (7540):529533*, 2015.
3. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.
4. H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.
5. B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," 2015.
6. X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," *NIPS, pp. 3338 3346*, 2014.
7. M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas., and R. Munos, "Increasing the action gap: New operators for reinforcement learning," *AAAI*, 2016.
8. A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," 2015.
9. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *ICLR*, 2016.
10. C. Watkins, "Learning from delayed rewards," 1989.
11. Melo and F. S, "Convergence of q-learning : A simple proof," 2001.
12. A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.
13. Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning,," 2016.
14. G. Rummery and M. Niranjan, "Online q-learning using connectionist systems," 1994.
15. W. Marco and S. Jurgen, "Fast online q(λ)," *Machine Learning 33 (1): 105-115*, 1998.
16. J. Perez, C. Germain-Renaud, B. Kégl, and C. Loomis, "Grid differentiated services: A reinforcement learning approach," in *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE, 2008, pp. 287–294.
17. M. Plappert, "keras-rl," https://github.com/keras-rl/keras-rl, 2016.
18. Z. Zhou, X. Li, and R. N. Zare, "Optimizing chemical reactions with deep reinforcement learning," *ACS central science*, vol. 3, no. 12, pp. 1337–1344, 2017.
19. A. PIVA, "Dynamic trading strategy for a risk-averse investor via the q-learning algorithm," 2019.