Reinforcement Learning Assignment

ON

Playing Space Invaders with Deep Q-Learning

BY Group - RL3

Name of Students	ID No.
Shrish Shankar	2018B5A70707H
Keshav Kabra	2018AAPS0527H
Jayaram J	2018B5A70796H
A Rohit Pradeep Kumar	2019A7PS0154H

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(Hyderabad Campus)

(1st December 2021)

Contents

I. Original Paper

1. Introduction

We present a deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The Q-Learning algorithm for reinforcement learning is modified to work on incredibly high dimensional states (images) using a convolutional neural network called the Deep-Q learning algorithm. We apply this method to the Space Invaders game by Atari.

The current deep learning revolution has made it possible to learn feature representations from high-dimensional raw input data such as photos and videos, leading to advancements in computer vision tasks such as recognition and segmentation. In high-dimensional reinforcement learning contexts, they describe a convolutional neural network (CNN) architecture that can learn policies from raw image frame input. Deep Q-Networks is the name given to the CNN that was trained using a form of Q-learning (DQN).

We use The Arcade Learning Environment's Space Invaders game to demonstrate our method (ALE). It contains an RL testbed with a high-dimensional visual input (210 x 160 RGB films at 60Hz) and an abroad and engaging set of tasks for agents to complete. Our goal is to develop a single neural network agent that can learn to play the game successfully. The network was not given any game-specific knowledge or hand-designed visual features, nor was it given access to the emulator's internal state; it learned only from the video input, reward and terminal signals, and the set of available actions precisely like a human player would.

2. Implementation

We consider tasks in which an agent engages in a series of actions, observations, and rewards with an environment “E,” in this case, the Atari emulator. The agent chooses an action from the set of lawful game actions, A = {1,..., K}, at each time step. The emulator receives the action and adjusts its internal state as well as the game score. The agent only looks at an image $x_t \in R^d$ from the emulator. It also receives a prize r_t, which means the difference in-game score.

We evaluate the action and observation sequences, such as $s_t = x_1, a_1, x_2,..., a_{t-1}, x_t$, and acquire game tactics based on these sequences.

We assume that future prizes are discounted by a factor of γ every time-step and define the future discounted return at time t as $R_t = \sum^{T}_{t'=t} \gamma^{t'-t} r_t$; where T is the time-step when the game ends. After observing some sequences s and subsequently doing some action a, we define the optimal action-value function Q(s,a) as the expected return achievable by following any strategy, i.e.,

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

In practice, this basic technique is completely impracticable. A Q-network is a type of neural network function approximator with weights, and by minimizing a series of loss functions $L_i(\theta_i)$ that changes at each iteration i, a Q-network can be trained.

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right], \text{ where } y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a \right]$$

The parameters from the previous iteration θ_{i-1} are held fixed when optimising the loss function $L_i(\theta_i)$. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

The use of stochastic gradient descent to optimize the loss function is computationally advantageous. When the weights are changed after each time step, and the expectations are replaced with single samples from the behavior distribution and the emulator E, we get the well-known Q-learning algorithm. The algorithm solves the reinforcement learning task. It learns about the greedy strategy $a = \max_a Q(s, a; \theta)$, while following a behavior distribution.

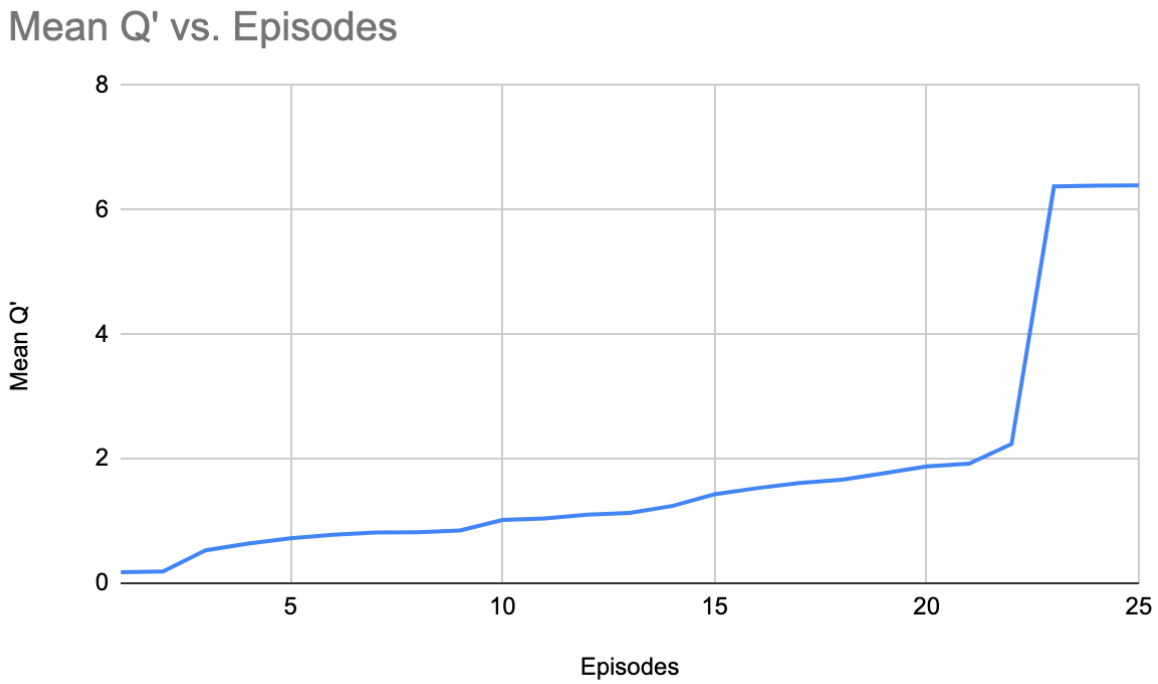
3. Results

Table

The Model trained for 25 episodes and 10000-time steps.

Time Steps	Episodes	Mean Reward'	Mean Q'
358	1	0.042	0.176547
721	2	0.289	0.187539
1027	3	0.163	0.528262
1509	4	0.373	0.637758
1941	5	0.15	0.719815
2324	6	0.196	0.776411
2750	7	0.235	0.813408
2986	8	0.191	0.820281
3556	9	0.368	0.848381
3946	10	0.141	1.015308
4402	11	0.23	1.041094
4875	12	0.19	1.100067
5333	13	0.076	1.127821
5722	14	0.347	1.240755
5954	15	0.194	1.429427
6152	16	0.152	1.527407
6439	17	0.174	1.611033
6692	18	0.079	1.662813
7198	19	0.158	1.767311
7420	20	0.158	1.877863
7792	21	0.215	1.922221
8215	22	0.189	2.236436
8426	23	0.024	6.378694
9213	24	0.267	6.389868
9597	25	0.013	6.393257

Graphs


Mean Q' vs. Episodes
Mean Q'
8
6
4
2
0
5
10
15
20
25
Episodes


Mean Reward' vs. Episodes
Mean Reward'
0.4
0.3
0.2
0.1
0.0
5
10
15
20
25
Episodes

II. Improvement - Using Double Q-Learning

1. Theory

The well-known reinforcement learning method Q-learning works sub-optimally in some stochastic settings. This poor performance is caused by large overestimations of action values. The optimal policy of the Agent in basic Q-learning is to always choose the best action in every given condition. The theory is based on the assumption that the best action has the highest expected/estimated Q-value. However, because the Agent has no prior knowledge of the environment, it must first estimate Q(s, a) and then update them at each iteration. There are a lot of noises in such Q-values, and we never know if the action with the highest expected/estimated Q-value is indeed the best.

Since the best Q(s, a) is estimated by using current Q(s, a), which is very noisy. The difference between the best and current Q(s, a) in Loss Function is also messy and contains positive biases which are caused by different noises. These positive biases impact tremendously on update procedure.

TD-target is UNKNOWN! Current Q-value

$$LostFunction = Q_{best}(s_t, a_t) - Q(s_t, a_t)$$

Estimated TD-target

Discount factor

$$Q_{best}(s_t, a_t) = R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

Reward Maximum next-state Q-value

Update Current Q-value

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Current Q-value

Double Q-Learning was proposed for solving the problem of large overestimations of action value (Q-value) in basic Q-Learning. Double Q-Learning uses two different action-value functions, Q and Q', as estimators. Even if Q and Q' are noisy, these noises can be viewed as uniform distribution. That is, this algorithm solves the problem of overestimations of action value.

The modified process for Deep double Q-learning
Q function is for selecting the best action with the maximum Q-value of the next state.
Q' function is for calculating the expected Q-value by using the action selected above.

Basic Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \boxed{Q'(s_{t+1}, \boxed{a})} - Q(s_t, a_t))$$

estimated/expected Q-value

$$\boxed{a} = \max_a Q(s_{t+1}, a)$$

$$q_{estimated} = \boxed{Q'(s_{t+1}, \boxed{a})}$$

Update Q function by using the expected Q-value of Q' function.

Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \boxed{Q'(s_{t+1}, \boxed{a})} - Q(s_t, a_t))$$

estimated/expected Q-value

Double Deep Q-Learning Network

Derived from Double Q-Learning, Double DQN uses two different Deep Neural Networks, Deep Q Network (DQN) and Target Network.

Q_{qnet} selecting the best action with maximum Q-value of the next state.

Q_{tnet} calculating the estimated Q-value with the action selected above.

Double Q Network

estimated/expected Q-value

$$Q_{qnet}(s_t, a_t) \leftarrow R_{t+1} + \gamma\, Q_{tnet}(s_{t+1}, a)$$

$$a = \max_a Q_{qnet}(s_{t+1}, a)$$

$$q_{estimated} = Q_{tnet}(s_{t+1}, a)$$

Update the parameters of Target Network based on the parameters of Deep Q Network per several iterations.

$$Q_{tnet}(s, a) = Q_{qnet}(s, a)$$

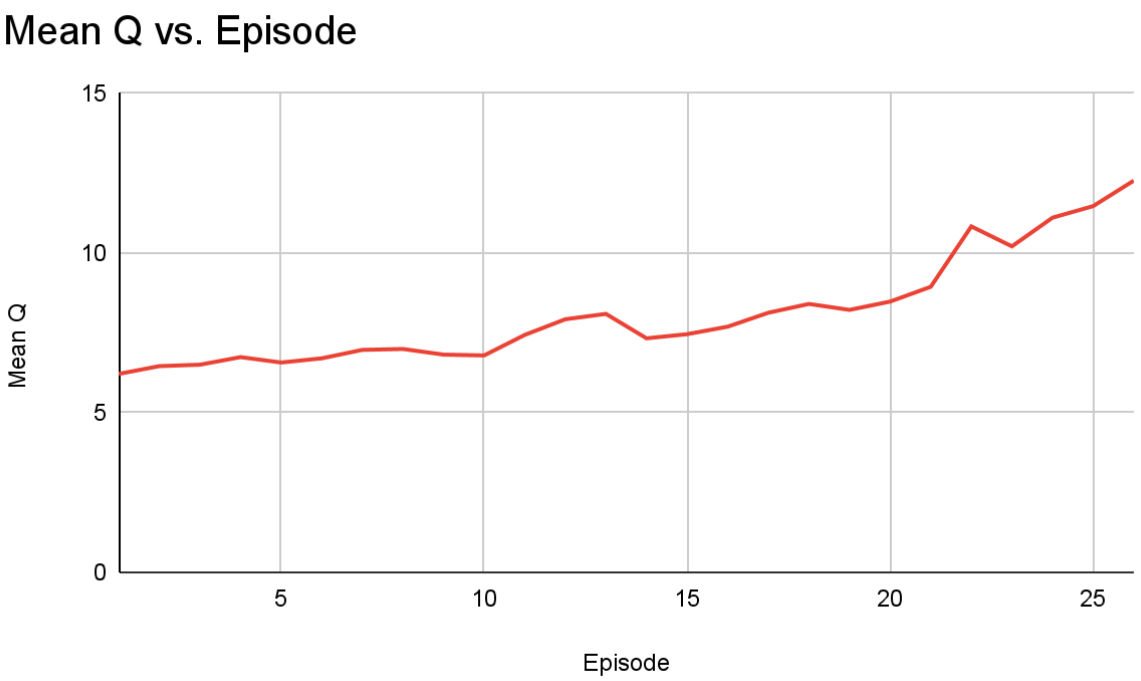
2. New Results

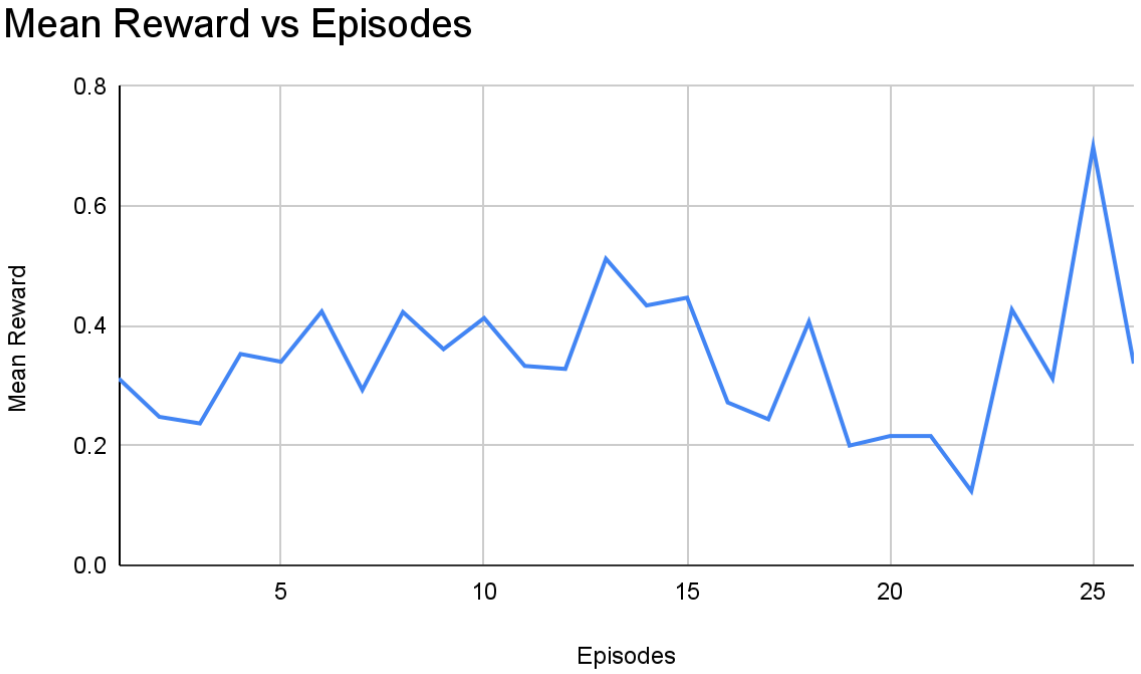
Table

The Model trained for 25 episodes and 10000-time steps.

Time Steps	Episodes	Mean Reward'	Mean Q'
430	1	0.312	6.213721
888	2	0.248	6.457474
1241	3	0.237	6.501996
1475	4	0.353	6.739651
1757	5	0.34	6.570093
2128	6	0.424	6.701147
2406	7	0.293	6.966412
1806	8	0.423	6.99407
3322	9	0.361	6.817507
3629	10	0.413	6.791816
4015	11	0.333	7.4341
4496	12	0.328	7.925118
4842	13	0.512	8.093274
5145	14	0.434	7.327957
5618	15	0.447	7.457885
6098	16	0.272	7.693295
6618	17	0.244	8.128769
6923	18	0.407	8.404425
7419	19	0.2	8.218744
7795	20	0.216	8.478813
8290	21	0.216	8.941478
8525	22	0.124	10.83271
8766	23	0.427	10.207751
9230	24	0.312	11.1065
9472	25	0.699	11.4678
10000	26	0.337	12.2654

Graphs


Mean Q vs. Episode
15
10
5
0
Mean Q
5
10
15
20
25
Episode


Mean Reward vs Episodes
0.8
0.6
0.4
0.2
0.0
Mean Reward
5
10
15
20
25
Episodes

III. Comparison

We use the mean reward after training to compare the original paper (Deep double Q-Learning) with the Deep reinforcement learning with Double Q-Learning modification.

Algorithm	Deep Q Learning	Deep Double Q-Learning
Mean Reward	53.75	**306.25**

IV. Other Experiments

1. To improve the network, we tried adding dropout. In both situations, we found that dropping out harmed gameplay. After dropout, the average score for ten games (during testing) was lower.
2. Another idea was to pre-train the network using expert demonstrations (i.e., making a human play), but it wasn't possible due to computational and programming difficulties.

V. Main Accomplishments

1. We learned and implemented Q-Learning using Deep Neural Networks. The model was tested against 10 cases, with good results. It could've been improved by training it for more episodes.
2. We made modifications to the original code and implemented the Double Q-Learning instead, and it showed a considerable improvement in a short time.
3. Therefore, we can safely conclude that Double Q-Learning performs better for Atari-like games.

Code: Github Link

VI. Contribution

1. Shrish Shankar	Worked on the implementation of the Deep Q-Learning & improved networks.
2. Keshav Kabra	Worked on the RL Agent code implementation and Open AI Gym integration.
3. Jayaram J	Worked on the report and comparative analysis of different models
4. A Rohit Pradeep Kumar	Worked on the report, analyzing results for Double Deep Q learning and plotting graphs.