Reinforcement Learning for Games

Abhavya Chandra*
abhavya.chandra@iitgn.ac.in
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat

Apoorv Agnihotri
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat
apoorv.agnihotri@iitgn.ac.in

Jatin Dholakia
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat
jatin.dholakia@iitgn.ac.in

Sagar Gupta
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat
gupta.sagar@iitgn.ac.in

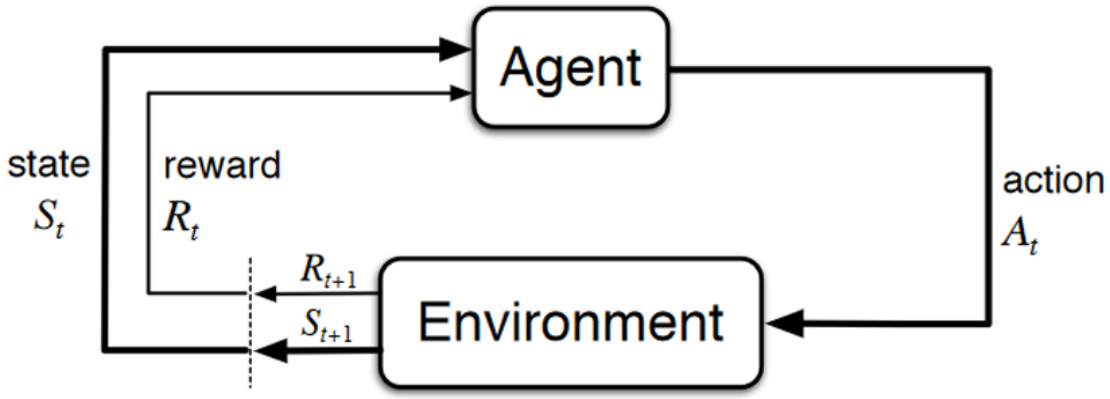


Figure 1: The agent / environment interaction in a Markov Decision Process[5]

ABSTRACT

Reinforcement learning algorithms enable an agent to optimize its behaviour by interacting with an environment. We apply various reinforcement learning approaches on games with varying number of states. In our approach we study and compare the performance of Q-learning, Deep Q-learning and a CNN based method for Deep Q-learning. While Q-learning prove to be quite effective on games with small number of states, it becomes ineffective to find optimal policy on games with a large number of states. Deep Q-learning, which uses a neural network to approximate the features in a state, are able to give much better results for games with high number of states.

CCS CONCEPTS

• **Artificial Intelligence** → **Reinforcement Learning**; • **Reinforcment Learning** → *Model Free Policies.*

KEYWORDS

reinforcement learning, artificial intelligence, neural networks, q-learning, deep q-networks, cnn, value-based-policies

*All authors contributed equally to this research.

1 INTRODUCTION

Reinforcement Learning (RL) is an autonomous, self-teaching system that learns by trial and error. It learns by performing randomly chosen actions over a given state with the aim to maximize a cumulative reward in order to achieve the best possible outcomes. This is similar to how humans learn, like riding a bike, where in the beginning we fall off (acquiring negative scores), try new tricks (randomly choosing future actions) and continue. Over time, using this feedback (+ve and -ve rewards pertaining to a given action) the learner fine-tunes its action and learns how to ride a bike. A similar feedback loop based approach is followed in RL as seen in figure(2). In our project we have aimed at implementing this feedback loop using three different methods:

- Classical Q-Learning
- Deep Q Network
- Deep Q Network using CNN

Each of these methods have their merits and demerits depending on the number of states and complexity of the game (environment) they are being trained on.
Our project aims at a detailed analysis of this dependence and the same has been performed by training each of these agents on three different environments with varying complexities as mentioned below.

2 FRAMEWORKS

We apply our approaches on a range of games (environments) with increasing number of states and complexity provided by OpenAI's Gym library [1]. The games we tested on were: CartPole, Lunar-Lander, MsPacman-ram and Ms-Pacman.

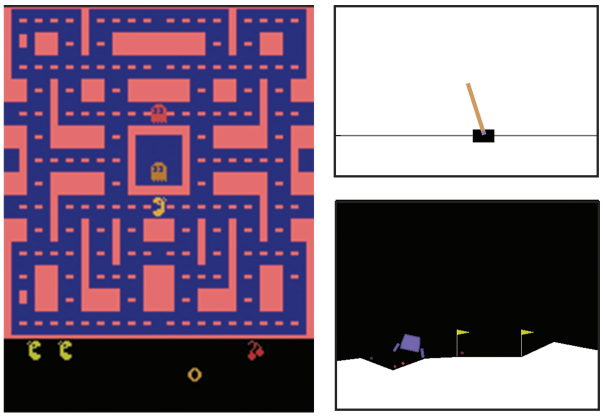


Figure 2: In clockwise order from top right : Cartpole, Lunar Lander, Pacman

2.1 CartPole

A pole is attached to a cart which moves on a friction-less rail.

- **Goal :** The episode starts with the pole being upright and the goal is to balance it in vertical position.
- **Action :** The agent can take a +1/-1 action to move the cart left or right in order to balance the pole.
- **State-space :** The state obtained from environment is a 4-dimensional vector of discrete values.
- **Reward :** A reward of +1 is given to the agent for every time-step the pole remains upright.

The episode ends when the pole is more than 15 degrees from vertical or cart moves more than 2.4 units in either direction.

2.2 Lunar Lander

- **Goal :** The objective of the game is to land the agent(lunar lander) between the flags.
- **Action :** The agent can take one of the 4 actions: do nothing, fire left engine, fire right engine, fire main engine.
- **State-space :** The state-space obtained from observing the environment is a 8-dimensional vector of continuous values.
- **Reward :** There is a negative penalty for firing the engine and landing outside the marked position.

The episode ends when the lunar lander comes to rest.

2.3 Pacman

It is one of the Atari 2600 games implemented on The Arcade Learning Environment (ALE). Each action taken by the agent is repeatedly performed for k frames where k is uniformly sampled from 2,3,4. The agent receives a positive reward for each pellet it eats and a large reward for big pellets, cherries and ghosts. The ghosts in the game take a random action, which makes the environment dynamic. A episode is completed when all the 3 lives of Pacman are over. There are two versions of the game available in Gym library:

- **Ms-Pacman-ram** - The observation is the RAM of Atari machine consisting of 128 bytes. This input was given to the Deep Q-learning network.
- **Ms-Pacman**- The observation is high dimensional visual input of 210x160 video of the environment at 60fps. This input was used with the CNN approach of Deep Q-learning.

3 RELATED WORKS

Our work focuses on analysis of various approaches explored in the existing literature. We experimented with Convolutional Neural Network based Q-value learning techniques explored in the work done by Minh et. al. [3]. The paper uses Deep Q-Learning (DQL) to train agents to maximize the scores in Atari games. The idea behind DQL is to approximate the Q function with a deep convolutions neural network (Deep Q-Network). We chose to experiment with this method due to it's success in learning policies in large state games and our environment Ms-Pacman was intractable to learn using classical methods. For our experiments on classical methods, we used the work done by Watkins et. al. [7]. The classical Q-value approach focuses on learning a 2D table where one dimension corresponds to the states and one dimension corresponds to the possible actions. Given a state, the best action to take is the action with maximum value for the elements. We also take inspirations from ideas like discretizing continuous state-spaces explored in the work of Uther et. al. [6]. For better understanding and explanations we referred to the amazing book by Sutton et. al. [5].

4 BACKGROUND

In this section we will be talking about the three methods of reinforcement learning that we have implemented.
In RL, in order to train the agent, we consider a set of tasks associated with an environment (game). In general, these environments may be stochastic.Further, the tasks associated with such environments are assumed to follow Markov Decision Process (MDP). Initially during training, at each time-step, an action is chosen randomly by the agent from the set of legal actions and passed to the emulator where the emulator modifies it's internal state and returns the reward of choosing that action. The internal state of the emulator may be observed by the agent in a vector format or in the form of an image depending on the method in question. Some important terms in this regard are as follows:

- **State space** - All possible configurations in the environment.
- **Action space** - A set of all allowed actions.

4.1 Q-Learning

In order for the agent to learn the quality of a possible action, classical Q-learning makes use of the action value function which gives the so called Q(s,a) value for that state given that action. The objective of the agent is to maximize the cumulative reward at the end of the game. Thus, the Q - values are learned iteratively by updating the current Q-value estimate towards the reward observed at the current internal state plus the maximum Q-value over all the

actions in the resulting state. The Q function (action value function) can thus be defined as:

$$Q^{\pi}(s_t, a_t) = \underline{E}[\underline{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...}|\underline{s_t, a_t}]$$

Q value for that state given that action — Expected discounted cumulative reward ... — given that state and that action

Figure 3: Action value function [4]

Using this approach the agent will be able to know the best action to take for each state. This results into populating something known as the Q-table (initialized as a zeros table) which indicates the above information in a tabular format with the columns corresponding to the different possible actions and the rows to the states.
The Q-values are updated according to the following equation:

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action — Current Q value — Learning Rate — Reward for taking that action at that state — Discount rate — Maximum expected future reward given the new s' and all possible actions at that new state

Figure 4: Bellman update equation

4.2 Deep Q-learning

The action is chosen by using *epsilon*-greedy policy. In this policy, epsilon is decreased linearly from 1 as the episodes increase and then fixed to a particular value. The action chosen by the agent is random with probability ϵ and according to the network with probability 1-ϵ. This way the agent explores the environment more in the initial stages and as the episodes increase it start exploiting the policy it has learnt.

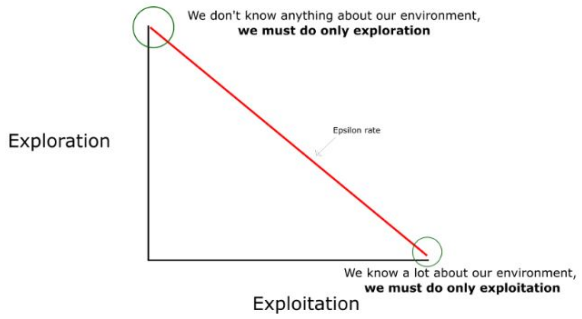


Figure 5: Epsilon - Greedy policy

4.3 Deep Q-learning

In Deep Q leaning the agent learns policy using neural networks instead of the Q-table discussed above. The main aim remains same to predict Q value for a given state and action pair. We found the neural network approach to be a better way to estimate $Q(s, a)$.

The minimization of the distance between $Q(s, a)$ and TD-Target (or temporal distance of $Q(s, a)$) is done using a different loss function: put the loss function here In Deep Q-Learning TD-Target y_i and $Q(s, a)$ are estimated separately by two different neural networks.The two networks are termed as Target Network and Q-Network. Target Network predicts the estimated Q(s,a) for the current state, while Q(s',a') is predicted by Q-Network. Here s' corresponds to future state and a' corresponds to possible actions in future state.

The two networks have same network dimensions but differ in parameters(weights, bias) involved. Networks are initialized randomly. In each iteration we update weights of Q-Network using the old back-propagation method. Further Target Network is updated based on weight of Q-Network in the previous iteration. The update can either be hard or soft. In hard update parameters of Target Network are assigned the same value as Q-Network's in $(i-1)^{th}$ iteration, while in soft update we multiply the Q-Network weights of $(i-1)^{th}$ by some factor before assigning it to Target-Network.

The minimization of the error loss function can be accomplished by usual gradient descent algorithms that are used in deep learning.

4.4 Deep Q-Learning using CNN

The features provided in the RAM version of Ms-Pacman were handcrafted and may miss out some of the important features of states. Hence a Convolutional Neural Network (CNN) is used which can "extract" features from raw image data. The first few layers are convolutional layers followed by fully connected layers which uses Deep Q-network (as explained above) to estimate the Q-values for the different possible actions in a given state.

5 APPROACHES

For the games of Cartpole, Lunar Lander and Pacman, we have used the following ways to train our agent

- **Cartpole** : Q- Learning and Deep Q-Learning
- **Lunar Lander** : Q- Learning by discretizing the feature space and Deep Q-Learning
- **Pacman** : Deep Q-Learning and Deep Q-Learning using CNN

5.1 Q-Learning

Q-learning as described by the work of Watkins et. al. [7] assumes the state space to be discrete, i.e. the states are infinitely countable. For running classical Q learning on the spaces that are uncountable, we get around this problem by discretizing the space into buckets as explored by the works of Uther et. al. [6]. For the case of Cartpole and LunarLander the state space is provided in a form of a continuous state vector. We have discretized the space by dividing each of the element in the state vector by having buckets between maximum and minimum values reached for corresponding elements. We try to get these max and min values by sampling randomly from the environment for some specified iterations.

5.2 Deep Q-Learning

The method of classical Q-learning is highly dependent on state space of the game. DQN takes a different path to deal with high dimensional states. We tried the DQN methodology on different

games mentioned above.
We utilize a technique known as experience replay to store the agent's experiences at each time step into a dataset called as replay memory. While training samples of experiences are drawn at random from replay memory. This procedure is particularly useful in removing the correlation between samples from the observation space and smoothing over changes in the data distribution.

To decide the network size we refereed literature and other implementations used in practice. From our experience and literature survey we saw no rigorously architecture for DQN. Researchers choose these parameters based on experience or by tweaking around till it works well for their purpose. Our network had 3 fully connected layers. The number of neurons in input layer has the same size as the observation space and the number of neurons in the output layer has the same size as the number of actions. We used one hidden layer with 64 neurons in it. Both Q-Network and Target-Network had the same network architecture. The hyper-parameters are as follows-

- Learning rate for Q-Network - 0.0002
- Discount rate (γ)- 0.99
- Memory size for replay buffer - 1000000
- Maximum exploration rate - 1.0
- Minimum exploration rate - 0.01
- Decay rate for exploration - 0.995

We updated Target-Network using hard update. We tried games our network on games like Pacman, CartPole and LunarLander for 200 episode.
Our agent was able to learn policies on different games without change any in hyper-parameters and network architecture.

5.3 Deep Q-Learning using CNN

Our goal is to connect a reinforcement learning algorithm to a deep neural network which operates directly on RGB images and efficiently processes training data by using stochastic gradient descent. We follow the approach mentioned by Mnih et. al. [3], which demonstrates DQN on 7 Atari games. Because their model was generalized to learn other games, we experimented with changing their model architecture which gave better average reward on Pacman.
We start with the raw Atari frames obtained from observation of the environment of dimension 210x160x3. The image frames are preprocessed by converting the RGB frames to gray-scale and downsampling to 84x84 image. It is converted to a square image because we use GPU implementation of 2D convolutions.
The input to the neural network consists of 4 frames stacked producing 84x84x4 image. The first hidden convolutional layer convolves 32 8x8 filters with stride 4 and ReLU non linearity. The second hidden convolutional layer convolves 64 4x4 filters with stride 2 and ReLU non linearity. The third hidden convolutional layer convolves 64 3x3 filters with stride 1. The final hidden layer is fully connected and consists 512 rectifier units. The output layer is a fully connected linear layer with nodes equal to number of possible actions.

6 RESULTS

Our experiments with the different approaches were able to give us insights into the positive and negative points. We trained all the agents using the OpenAI gym [1] on different environment that the library provided support for. We plotted the cumulative rewards with respect to episodes for all the agents to get a better insight at the performance of the various techniques in the literature.

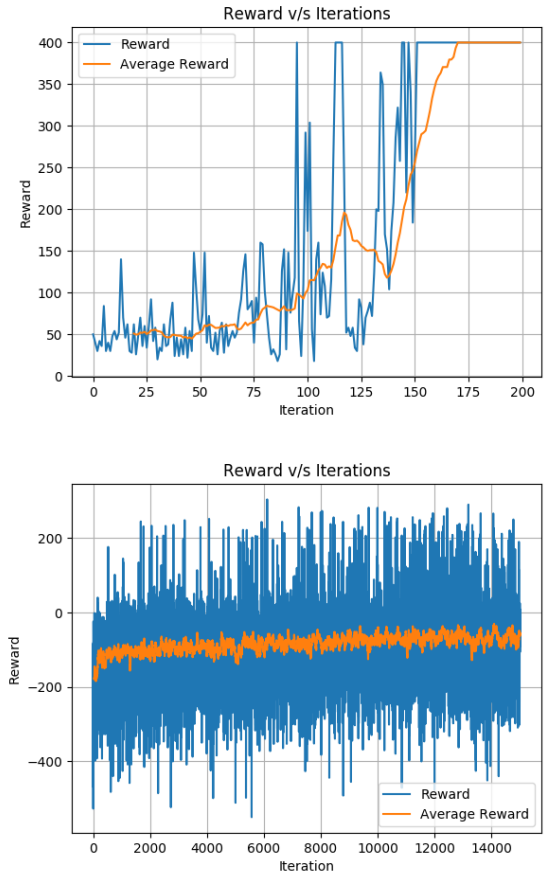


Figure 6: Q-Learning on (i) Cartpole and (ii) Lunar Lander

Our experiments with DQN on LunarLander and Ms-Pacman showed that we were able to learn a much better agent with the use of Neural Networks for approximating the Q-values, instead of the classical approach of updating the Q-table.

Further, experimenting with the CNN approach to the DQN, as proposed by Mnih et. al. [3], we tried to compare the performance of DQN agent trained on a continuous state vector and on RGB image data using CNN approach to DQN.
The plot shows that the average predicted Q increases much more smoothly than the average reward obtained by the agent. It shows an estimate of how much discounted reward the agent can obtain by following its policy from any given state.

7 CONCLUSION

Our approach showed that Q-learning performs worse than DQN as it was expected as the number of training examples needed are

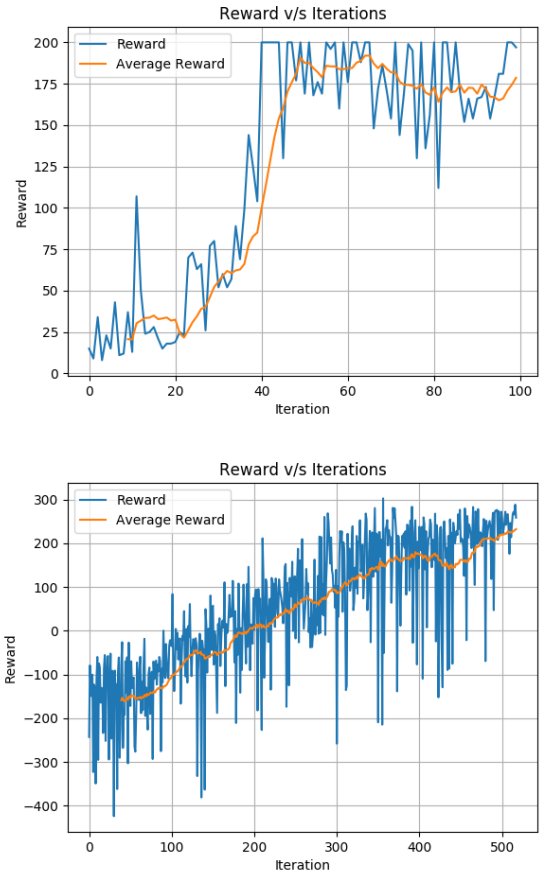


Figure 7: DQN on (i) Cartpole and (ii) Lunar Lander

extremely high to get a good enough confidence for each of the elements in the classical Q table. Whereas we propose the reason that DQN approach takes less number of iterations to learn the Q-value network is the fact that it is able to quantitatively say two states are similar (able to learn a bit abstracted state) and therefore requires less training. Further, we were able to get a surprising result regarding the performance of DQN with state vector as input vs DQN with the CNN approach. We thought the results would be in favour of a more precise state vector instead of a more general RGB data for the agent to learn the environment. We hope to look into the possible explanations for this observation in the future.

ACKNOWLEDGMENTS

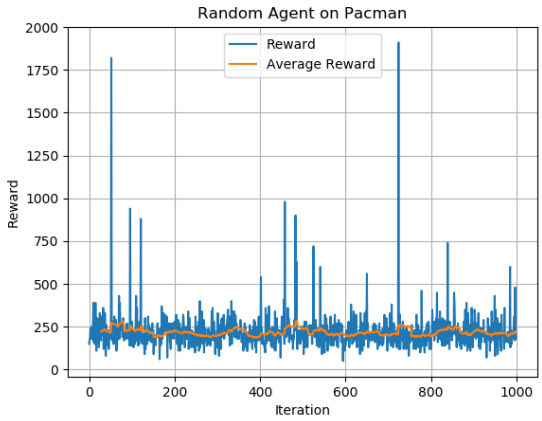
We would like to thank for the constant support we received at the hand of our mentor Shubham Singh and Prof. Nipun Batra affiliated with IIT Gandhinagar.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016).

[2] Abeynaya Gnanasekaran, Jordi Feliu Faba, and Jing An. [n. d.]. Reinforcement Learning in Pacman. http://cs229.stanford.edu/proj2017/final-reports/5241109.pdf

(1).png



(2).png

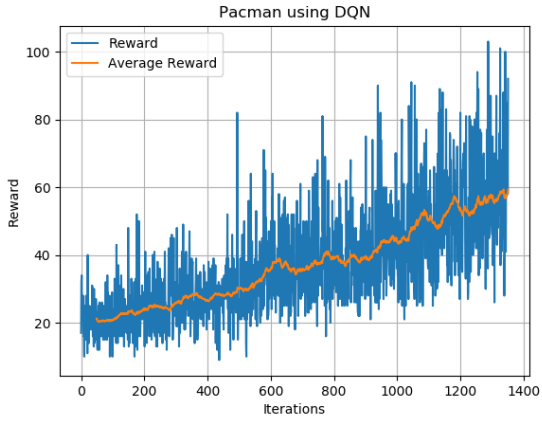


Figure 8: Agent on Pacman (i) Random and (ii) DQN using CNN

(1).png

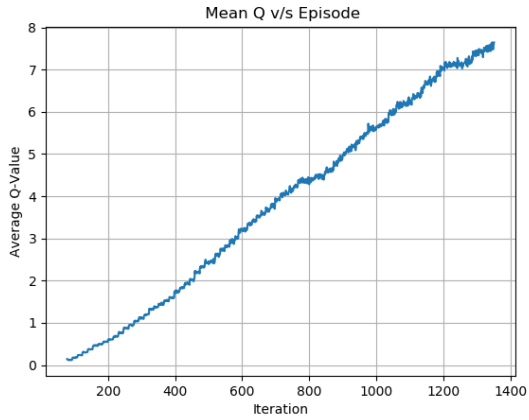


Figure 9: DQN using CNN on Pacman : Average Q-value over iterations

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[4] Thomas Simonini. [n. d.]. Diving deeper into Reinforcement Learning with Q-Learning. https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe

[5] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[6] William T. B. Uther and Manuela M. Veloso. 1998. Tree Based Discretization for Continuous State Space Reinforcement Learning. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI '98/IAAI '98)*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 769–774. http://dl.acm.org/citation.cfm?id=295240.295802

[7] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698