ENPM667: Project-01

A Technical Report on

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller

Deep mind

Sanchit Tanwar
UID: 119167717

Date of Submission: 12^{th} *December, 2022*

A. James Clark School of Engineering (Robotics)
University of Maryland, College Park, MD - 20742

Table of Contents

List of Figures

Abstract

Reinforcement learning is used to learn the controller based on the states of the environment. The reinforcement learning agent learns to predict the action from a set of possible actions (depends on action, for example in case of a car, we have control of gas, brake and the steering angle). Q learning is one of the approach used to solve the markov decision processes (MDP). Q learning works fine for the low dimensional data like grids but for a real world environment like visual environment reinforcement learning relies on the hand crafted features of the environment and thus the performance of these models thus depends on these features. With advent of deep learning and the significant improvement of visual information processing using deep learning models like [1], [2] [3], it seems only natural to use these models to improve the performance of existing reinforcement learning approaches like Q learning. Thus authors presents Deep Q learning [4]. The deep q learning model takes visual world (in our simulation, pong game) as input, process it using Convolutional neural network and returns a value function to take action.

Keywords: *Reinforcement learning, deep reinforcement learning, deep learning, Atari*

Chapter 1

Introduction

Reinforcement learning is machine learning that involves training agents to make decisions in a given environment to maximize a reward signal. This can be used as a different way of learning compared to supervised and unsupervised learning methods. Reinforcement learning agents need to interact with the environment to get the current state; often, this environment is highly dimensional, like in the case of the visual world or speech. The interaction of computers with the high dimensional data depended on the hand-crafted features designed by engineers and scientists. The performance of the reinforcement learning agent depends a lot on the performance of the feature extraction module. However, with the advent of deep learning, these features are learned using large-scale datasets, and we have seen a sudden jump in the performance of computer vision systems [1]. Thus these systems can be used to improve the reinforcement learning models' performance, which depends on the feature extraction modules.

Modelling reinforcement learning as a deep learning problem can be very challenging as most successful deep learning system works with a sufficiently large dataset annotated by human annotators; thus, the learning signal is usually noise free. In comparison, reinforcement learning only receives a scalar reward signal which in most cases is sparse and noisy. In challenging situations like computer games, reinforcement learning models must learn strategies that increase immediate rewards and maximize future rewards. For example, in a game of chess, an AI must consider how to maximize immediate rewards (such as killing an opponent's piece) and the long-term goal of winning the game in as few moves as possible. This is different from deep learning, where rewards are often immediate and noise-free (because they come from the labels provided by humans).

[4] demonstrates that using a convolutional neural network helps solve these challenges, and the model learns the controller using the raw video data of the games. The method is applied to multiple atari games and achieves human-level performance in these games. Video games are often used to simulate deep reinforcement learning models because they provide a convenient and controllable environment for testing and evaluating the

performance of the models. Video games have a well-defined set of rules and goals, making them suitable for reinforcement learning algorithms. Additionally, many video games have a wide variety of environments, objects, and interactions, which provide ample opportunity for the reinforcement learning agent to explore and learn. Furthermore, video games often have a graphical user interface that allows the agent's actions and decisions to be visualized and analyzed, making it easy to assess the model's performance. Overall, the use of video games in simulating deep reinforcement learning models offers several benefits that make it a popular choice for research and development in this field. Video games are also close to the real world, and the hypothesis proved on video games can be extended to more practical problems. One example can be a more recent paper by deepmind, which created an approach to beat humans in a complicated game of Go [5]. This model has since been extended to solve some of the most complex human problems, which includes [6], which helps in modelling the protein's 3d structure and helps researchers to understand better the function and behaviour of proteins, which is essential for a variety of applications in medicine and biotechnology. [7], which recently outperformed a 50-year-old method for matrix multiplication. Matrix multiplication is an integral part of most computer algorithms, especially deep learning; thus, improving the method of matrix multiplication can help improve the speed of these algorithms without any change.

Chapter 2

Background

2.1 Markov decision processes (MDP)

MDP is an extension of the Markov chain and provides a mathematical framework for modelling decision-making situations 2.1. MDP can be represented by five essential elements, a set of states (S) the agent can be in, a set of actions (A) that an agent can perform for moving from one state to another, transition probability which is the probability of moving from one state to another state by performing some action, a reward probability which is the probability of a reward acquired by the agent for moving from one state to another state by performing some action, a discount factor, which controls the importance of immediate and future rewards. Now lets define these elements in a bit more detail for better understanding [4].

2.1.1 Rewards

In reinforcement learning, an agent receives rewards from the environment based on the actions it performs. Rewards are numerical values that reflect the quality of the agent's actions, such as +1 for a good action and -1 for a bad action. The goal of the agent is to maximize the total amount of rewards it receives from the environment, known as the returns. This is different from maximizing immediate rewards, and is typically done by considering the long-term effects of actions. The total amount of rewards the agent receives can be formulated as:

$$returns = \sum_{t=0}^{\infty} \gamma^t r_t$$

where γ is the discount factor (a value between 0 and 1 that determines how much future rewards should be considered when making decisions) and r_t is the reward received at time t.

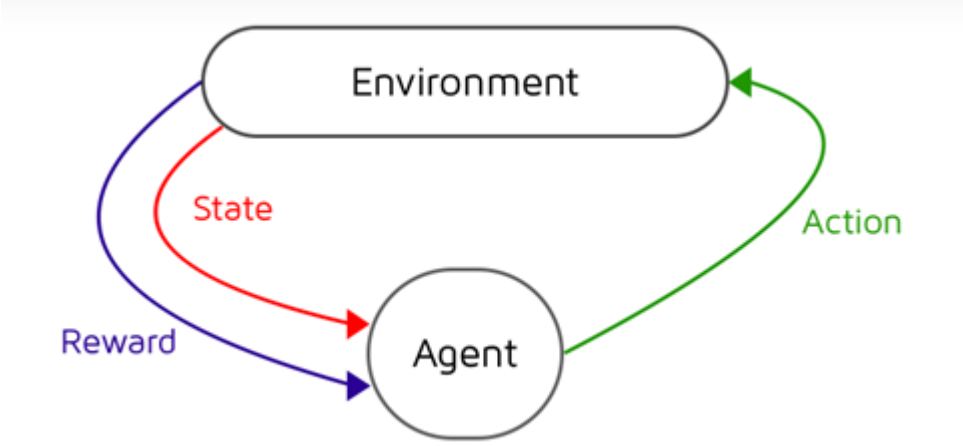

Figure 2.1: Markov Decision Processes

2.1.2 Policy function

In reinforcement learning, a policy is a function that maps states to actions. It is typically denoted by π. A policy specifies what action an agent should take in each state. The goal of reinforcement learning is to find the optimal policy, which is the policy that maximizes the expected cumulative reward. The optimal policy determines the optimal action to take in each state, and can be found using reinforcement learning algorithms such as Q-learning or policy gradient methods.

2.1.3 State Value function

The state value function, denoted by $V^{\pi}(s)$, specifies the expected cumulative reward for an agent following a policy π when starting in a given state s. The state value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t|S_t = s]$$

where $\mathbb{E}_{\pi}$ is the expected value operator with respect to the policy π, R_t is the reward at time t, and S_t is the state at time t.

The state value function depends on the policy π and will vary depending on the policy chosen. The goal of reinforcement learning is to find the optimal state value function, which is the state value function for the optimal policy. The optimal state value function is defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

where the maximum is taken over all possible policies.

2.1.4 State action value function

The action-value function, also known as the Q-value function, is a function that estimates the expected future rewards for an agent taking a specific action in a specific state. The action-value function is typically represented as a table or a function approximator, such as a neural network, that maps states and actions to Q-values. The action-value function is used to guide the agent's decision-making process by providing it with information about the expected rewards of different actions in different states.

The equation for the action-value function is as follows:

$$Q(s, a) = \mathbb{E}[R_t | S_t = s, A_t = a]$$

where $Q(s, a)$ is the action-value for taking action a in state s, $\mathbb{E}$ is the expected value operator, R_t is the reward at time t, S_t is the state at time t, and A_t is the action taken at time t.

2.2 Bellman Equation

Bellman equation is the basic block of solving reinforcement learning and is omnipresent in RL. It helps us to solve MDP. To solve means finding the optimal policy and value functions.

The optimal value function $V^{\pi}(s)$ is one that yields maximum value.

The value of a given state is equal to the max action (action which maximizes the value) of the reward of the optimal action in the given state and add a discount factor multiplied by the next state's Value from the Bellman Equation.

$$V^*(s) = \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right\}$$

2.3 Q learning

Q learning [8] has been used to solve the Markov Decision Processes. The goal of Q learning is to learn the optimal action-value function, which maps

each state-action pair in the MDP to the expected future reward for taking that action in that state. The algorithm does this by iteratively exploring the environment and updating the action-value function based on the rewards it receives. The algorithm selects the action that maximizes the expected reward in each iteration and updates the action-value function using the observed reward. Over time, this process allows the algorithm to learn the optimal policy for the MDP.

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

The basic idea behind many reinforcement learning algorithms is to the action value function by using bellman equation as an iterative update $Q_{i+1}(s,a) = \mathbb{E}_{s' \sim \epsilon}[r + \gamma \max_{a'} Q_i(s', a')|s, a]$. Such value iteration algorithm can converge to the optimal action value function, $Q_i \to Q^*$. In practice, this approach fails because the action-value function is estimated separately for each sequence without generalisation. Thus, it is common to use function approximator to estimate the action-value function, $Q(s, a; \theta) \approx Q^*(s, a)$. A neural network can be used as the function approximator with weights θ and can be trained by minimising loss functions

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim p(.)}[(y_i - Q(s, a; \theta_i))^2]$$

2.4 Convolutional Neural networks

Convolutional neural networks(CNNs) [9] [3] [2] [1] is a type of artificial neural network that is commonly used in computer vision tasks. CNNs are designed to process data with a grid-like structure, such as an image and composed of multiple layers, including convolutional, pooling, and fully-connected layers. The convolutional layers are responsible for extracting features from the input data. This is done by applying filters to the input, which scan the input and produce a set of feature maps. The pooling layers are used to downsample the feature maps, reducing the dimensionality of the data and making the network more computationally efficient. This also helps to make the network more robust to small changes in the input data. The fully-connected layers are used to make predictions based on the extracted features. These layers take the extracted features as input and produce an output, such as a class label for an image. One of the critical advantages of CNNs is that they can learn hierarchical representations of the data. This means that the network learns to extract increasingly complex features at each layer, starting with simple edge detectors in the early

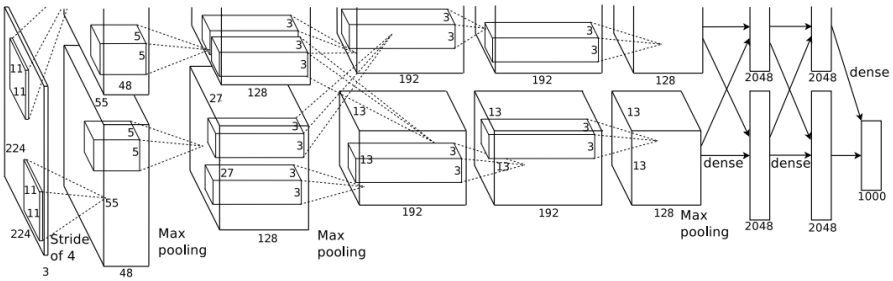

Figure 2.2: Alexnet architecture (Convolutional neural network)

layers and moving on to more abstract features in the deeper layers. This allows the network to learn powerful data representations well-suited to the task. Another advantage of CNNs is that they are translation invariant. This means that the network can recognize an object in an image even if it is not perfectly aligned with the input. This is because the convolutional filters can detect the presence of an object at different positions in the input. CNN's have proven to be highly effective for a wide range of computer vision tasks, including object recognition, image classification, and object detection.

Chapter 3

Methodology

In this section, I explain the deep q-learning and the algorithm other implementation details discussed in the paper. First important useful technique discussed in paper for the success of the approach is the experience replay [10]. Experience replay is used to improve the stability and performance of the learning algorithm by randomly sampling from a replay buffer that stores the agent's past experiences. Each experience comprises the state, action, reward, and next state observed by the agent at a given time step. During training, the agent samples a batch of experiences from the replay buffer and uses them to update the Q-value function. This has several benefits over learning from the current experience alone: It decorrelates the samples, which helps to improve the stability of the learning algorithm. If the samples are correlated, then the updates to the Q-value function will be correlated as well, which can lead to oscillations and other instability issues. Experience replay helps to break this correlation by sampling from a large number of past experiences. It makes the learning algorithm more efficient. By using samples from past experiences, the agent can learn from a much more significant number of samples than it would by using only the current experience. This allows the agent to learn more quickly and use their experiences better. It allows the agent to learn from rare or otherwise hard-to-observe experiences. If the agent only learned from the current experience, it might miss out on important information contained in rare or hard-to-observe experiences. By using experience replay, the agent can also learn from these experiences.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

The basic idea of the algorithm is as follows:

1. Initialize the Q-value function Q(s, a) with random weights.

2. At each time step, the agent observes the current state s and selects an action a according to an exploration policy, such as epsilon-greedy.

3. The agent then takes action a and observes the next state s' and the reward r.

4. The agent adds the experience (s, a, r, s') to a replay buffer.

5. At regular intervals, the agent samples a batch of experiences from the replay buffer and uses them to update the Q-value function using the Bellman equation:

$$Q(s, a) = r + \gamma * \max Q(s', a')$$

where γ is a discount factor that determines the importance of future rewards.

6. The updated Q-value function is then used to select the next action.

This process continues until the Q-value function has converged to the optimal solution.

3.1 Preprocessing and Deep learning model architecture

The raw atari frames are fetched from the game and downsampled to a size of 110x84 and finally cropping a region of 84*84 which roughly captures the playing area to reduce the computation, this image is passed through a convolutional neural network and the output is through a fully connected

linear layer with number of dimensions equal to the number of actions in the game. The outputs correspond to the predicted Q-values of the individual action for the input state.

3.2 Experiments and training details

The model was trained using RMSProp optimizer with a minibatch size of 32. The behavior policy during training was greedy with annealed linearly from 1 to 0.1 over the first million frames, and fixed at 0.1 thereafter. The model was trained for 10 million frames and used a replay memory of one million most recent frames. Frame skipping strategy is also used, i.e the agent sees and select actions from every k^{th} frame and the last action is repeated on every skipped frame. The average total reward in a episode is used to evaluate the model while training.

We trained our model on the game of pong. Pong game is atari version of the real world pong, the model takes input frame sequences and task is to defeat the game AI. The model we trained destroyed the game AI by 21-0 in all the tests we ran which shows how good the DQN algorithm can be in comparison to AI with hand written rules.

The sample video can be seen here: link

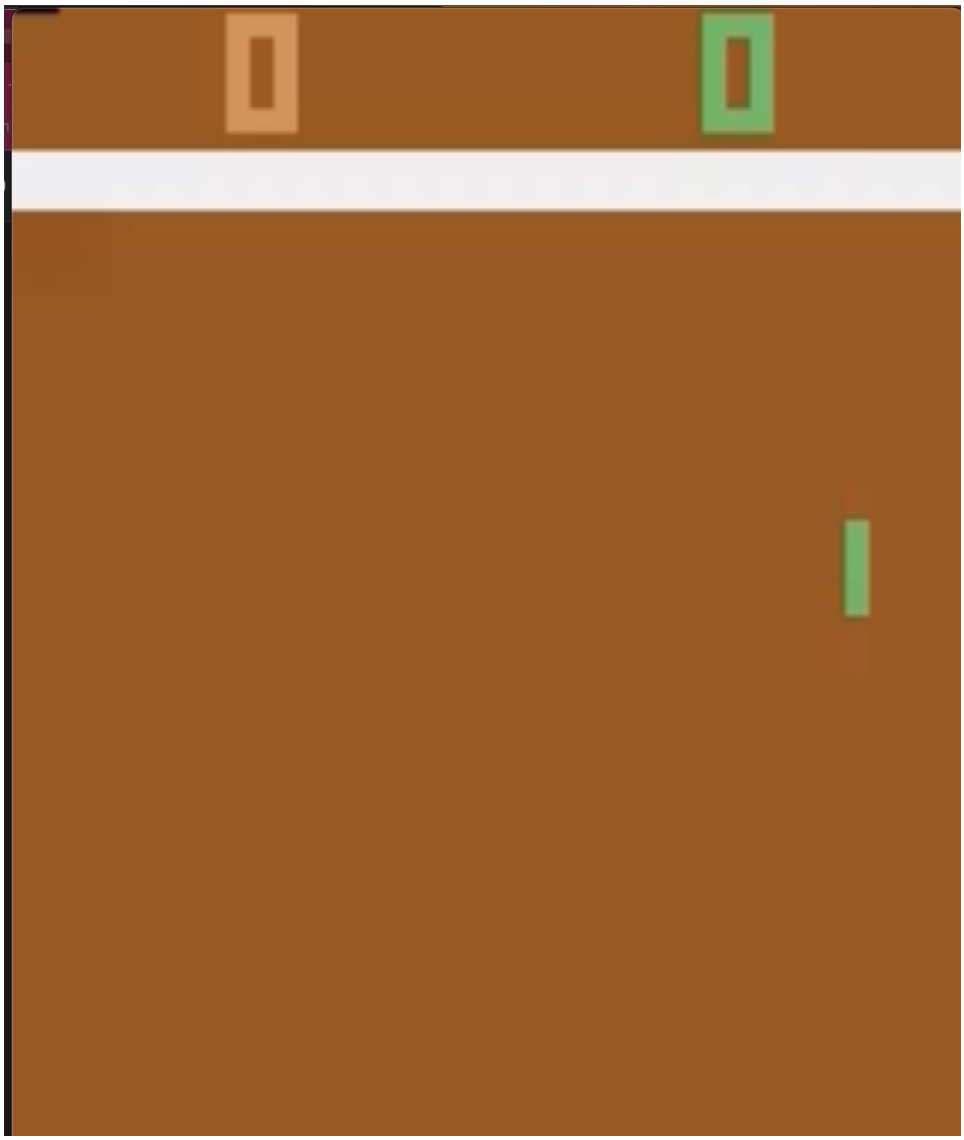

Figure 3.1: Sample screen from the game of Pong

Chapter 4

Discussion and Future Work

The deep q-learning can be used in several real world applications. One such application can be to make a controller for redlight on cross roads with the cameras installed on each end of the road which can help us in counting the cars on each end and the controller optimizes the total amount of time it takes to clear the traffic or the amount of time each car stays in the frame. This can be useful espcially in cities where the cameras are already deployed. One future work can be to try out more recent deep reinforcement learning methods like double deep q-learning, soft actor critic, PPO [11], [12]. These methods show promising performance on more challenging scenerios and should be applied to more real world applications by the engineers. One other application to show the practicallity of the reinforcement learning to design controllers is the algorithm developed by google to manage the Cooling in there data center, this algorithm takes input from several sensors scattered around the data center and controls the Air cooler, the algorithm is based on deep reinforcement learning and helped Google to reduce the electricity consumption in datacenters by upto 30%.

Bibliography

[1] G. E. Hinton, A. Krizhevsky, and I. Sutskever, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, no. 1106-1114, p. 1, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[7] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

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

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] L.-J. Lin, *Reinforcement learning for robots using neural networks.* Carnegie Mellon University, 1992.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning.* PMLR, 2018, pp. 1861–1870.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.