Demonstration of a modified Double Deep Q-Learning algorithm in Atari shooting games

Maryam Alghfeli
Maryam.Alghfeli@mbzuai.ac.ae

Mario Cantero
Mario.Cantero@mbzuai.ac.ae

Nouf Alshamsi
Nouf.Alshamsi@mbzuai.ac.ae

1 Introduction

Machine learning is about analyzing data to make decisions or predictions. It is usually categorized as supervised, unsupervised, and reinforcement learning. In supervised learning the main objective is to find an optimal, generalized model for prediction from labeled data. In unsupervised learning, the machine is trained using an unlabeled dataset, and the main objective is to learn similarities, differences, and patterns. Reinforcement learning is about building an agent that learns from actions by interacting with the environment through trial and error to receive rewards as feedback to solve control tasks or decision making problems. This approach tries to learn an optimal policy and takes action for a given state to maximize the rewards through training [1]. The integration of reinforcement learning and deep learning gives a wide range of field applications in games, natural language processing, computer vision, robotics, energy, business management, finance, healthcare, education, etc.

1.1 Project objective

The objective of this project is to connect a reinforcement learning algorithm with a deep neural network (DNN) to create an agent that learns to play shooting games from the Atari environments. These games were implemented in the Arcade Learning Environment (ALE), and most of them share the same or similar action space. The objective is to train an agent that plays Atari games better than an agent that follows a random policy. A general view of how a reinforcement learning task works is shown in figure 1.

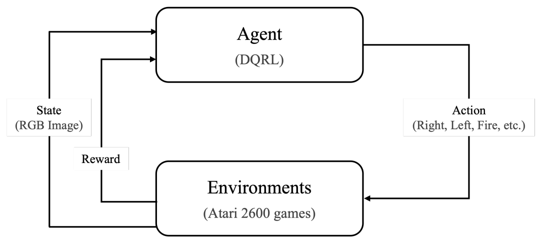


Figure 1: Reinforcement learning cycle

1.2 Motivation

We already know that reinforcement learning can be useful in real-life applications like healthcare, energy, transportation, or robotics, but in this case, we are leveraging the simplicity of a video game application to build a solid foundation on this paradigm to later work on the former situations on a research level. In addition, we chose this application because the intuition of how the algorithm works in terms of rewards and actions is more friendly to grasp in a video game.

1.3 Project overview

The agent learns to play shooting games from the Atari games environments: Space Invaders, Carnival, and Air Raid. All three of them share the same action space. In the implementation of the project, we used the OpenAI Gym environments. OpenAI Gym is an open-source python library for testing different reinforcement learning algorithms on various simulated environments. It works with a variety of frameworks, including Keras and TensorFlow. It is straightforward to grasp, makes no assumptions about the agent's structure and provides an interface to all reinforcement learning tasks [2].

The convolutional neural network (CNN) acts as the agent's brain, which can operate directly on RGB images. However, we are feed-forwarding the CNN with four consecutive stacked frames of the environment, as seen in figure 2, and then we will get an output of one vector of Q-values for each of the six actions. The Q-values represent the quality of an action. More specifically, it's the expected future reward if we select the corresponding action. So, in order to maximize future reward, we select the action with the highest Q-value.

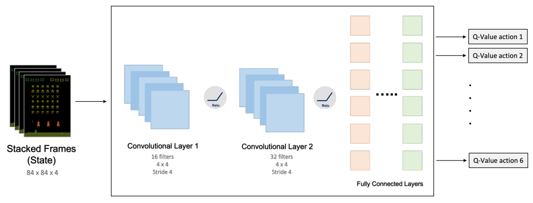


Figure 2: Double Deep Q-Learning Network Architecture.

2 Contributions

The main activities in this project and their respective in charge person are summarized in the following table 1:

Table 1: Main activities during project development

Activity	Person in charge
Training algorithm implementation with added techniques	Mario
Class of replay memory	Mario
Image preprocessing functions	Nouf
Class of CNN with PyTorch modules	Nouf
Evaluation phase of the agents	Maryam
Comparison between trained agents and agents with full exploration	Maryam

3 Related work

Mnih et al. in [3] propose a deep learning model that follows a reinforcement learning approach to learn control policies from raw pixels as input. Convolutional Neural Network (CNN) model was used to output a value function to predict future rewards. A single neural network agent trained using variant Q-learning algorithms. This method was applied to a range of Atari 2600 games from the Arcade Learning Environment. This approach produces cutting-edge outcomes in six of the seven games it was tested on, with no modifications to the architecture or hyperparameters.

Learning to play Atari 2600 games with frame skipping is critical to the performance of the agent's learning, and frame skipping is the number of frames an action is repeated before selecting new action [4]. Unfortunately, Q-learning algorithms sometimes perform poorly due to overestimating the action in some Atari games. This issue can be solved using double Q-learning that uses a double estimator approach to determine the value of the next state [5].

Another notable approach is using the epsilon greedy for the model to explore different strategies to maximize the rewards [6]. Another method achieved is prioritized experience replay, which assumes that not all the transitions have the same importance, which leads the agent to learn more effectively [7].

Nicolas Maquaire in [8] discusses the difference between the deep mind paper nature 2015 [9] that uses Arcade Learning Environment (ALE), which is deterministic, and the OpenAI Gym that injects stochasticity in the games. The ALE is deterministic and memorizes the sequence of actions instead of learning them; therefore, it's easy to get high scores. However, stochasticity makes the reinforcement algorithm more robust to transfer to other tasks. Nicolas experimented with three ways to inject the ALE with stochasticity: frame skipping, initial no-ops, and random action noise.

Moreover, Chen et al. in [10] proposed in a course project to add of a recurrent neural network on top of the conventional DQN architecture with the idea of adding importance to information learned on several previous time steps. This idea came from the fact that there are some games where past information is relevant to continue successfully in the game. They trained four different policy architectures in two different games: a conventional DQN, a conventional deep recurrent neural network (DRQN), a DRQN with linear attention, and finally, another DRQN with global attention. In the end, the conventional DRQN performed better than the conventional DQN in both Q*bert and Seaquest games. Although they did not use Atari shooting games, the idea of using a more complex deep neural network architecture on top of a simple one can make a significant difference in the performance of the agent, and we can leverage that flexibility if we have time for improvement in this month, and depending on how our DQN agent performs.

4 Proposed Methods

4.1 Preprocessing

Raw Atari game frame has high-resolution images, which is computationally expensive. To reduce the state complexity, we preprocess the game screen by converting the RGB representation into a grayscale image and cropping the regions to capture the playing area and remove areas that provide no value for the algorithm. Then the cropped image is downsampled to 84x84. This preprocessing is applied to the last four frames of the state and stacked to produce 84x84x4. Stacking the frames in order is important as it gives the network a sense of motion. Finally, the input is scaled between [0, 1].

4.2 Convolution Neural Network (CNN)

Figure 2 demonstrates the CNN archteciutre. The input to the neural network is 84x84x4, produced by the preprocessing step. The first hidden layer convolves 16, 8x8 filters with a stride of 4 and Relu nonlinearity, then the second hidden layer convolves 32, 4x4 kernel with a stride of 2 and followed by Relu nonlinearity. The final hidden layer is a fully connected layer with 256 rectifier units. Finally, the output layer is a fully connected layer that outputs Q values of all actions in a state: "noop," "fire," "right," "left," "rightfire," and "leftfire."

4.3 Modified Double Deep Q-Learning Algorithm

We used as a base the algorithm proposed by Deep Mind titled "Playing Atari with Deep Reinforcement Learning" [3], and we modified it. The first modification (highlighted in yellow in algorithm 1) is adding a new technique called "Double Q-Learning", following Hado van Hasselt's approach [5], to address the problem of using the same parameters (weights) for estimating the predicted Q value and target value. This problem implies correlation in the learning step, slacking the training phase and making it less efficient. Hence, we update the second network (the one used to calculate the target value) every C steps, where C is a hyperparameter.

The greedy epsilon policy with decay was also used as an added technique. The agent initially does not know how the environment works, but it can get more knowledge about it by exploring. This technique utilizes an exponential function on the ϵ, which is the probability of choosing a random action, and it decays over each episode. As a result, the exploration becomes less with time, which can be seen by part of the graph decaying as shown in figure 3. Then, the exploitation phase tries to maximize the return by exploiting the already known information. From figure 3 we can see that before the 1500Th episode, the agent was mostly exploring, and after that, it started exploiting more.

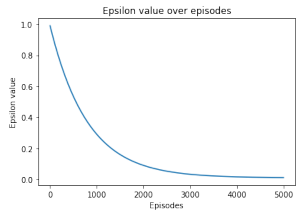


Figure 3: Epsilon Greedy Policy Plot.

Another technique was added where we used stochastic frame skipping (highlighted in blue in algorithm 1) to let the agent repeat a selected action for S consecutive frames, where S is a hyperparameter representing the number of skipped frames that range between 3 and 5. Frame skipping accelerates execution and as a result lowers the training time, while retaining the consistency of the predictions. For instance, in the space invaders game, when the spaceship (agent) shoots the enemies, the bullet takes some time to reach the target, and at that time, the agent will perform different actions that will be counted falsely as the optimal action once the bullet hits the enemy.

Furthermore, we randomized the initial states to prevent the agent from memorizing a sequence of optimal actions and induce generality. To achieve the randomness on initial states, the agent will perform a random action other than shooting by moving either to the right or left for k timesteps at the very beginning of an episode (highlighted in red in algorithm 1).

Algorithm 1 Modified Deep Q-learning with Experience Replay and Weight Update Delay

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weight $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for $episode = 1, M$ do
    Get $\varepsilon$ value from the decay function according to current episode
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    Initialize $skip\_counter = 0$
    Get a random integer value $n\_skipped\_frames$ between 3 and 5 inclusive
    Get a random integer value $K$ between 10 and 20
    for $counter = 1, K$ do
        Perform a right or left movement action $a_t$ randomly
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_(s_{t+1})$
    end for
    while $!Done$ do
        if $skip\_counter \ \% \ n\_skipped\_frames == 0$ then
            Select a random action $a_t$ with probability $\varepsilon$, otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
            Get a random integer value $n\_skipped\_frames$ between 3 and 5 inclusive
        end if
        Increase $skip\_counter$ value by one
        Execute action $a_t$ in emulator and observe reward $r_t$, frame $x_{t+1}$, and $Done$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1}, Done)$ in $D$
        Sample random mini-batch of 16 transitions $(\phi_j, a_j, r_j, \phi_{j+1}, Done)$ from $D$
        Set vector $y_j = \begin{cases} r_j, \text{ if episode terminates at step } j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-), \text{ otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
        Reset $\hat{Q} = Q$ every $C$ steps
    end while
end for
```

5 Setup of experiments

In reinforcement learning, the agent learns by interacting with the environment, which is done in real-time with continuous feedback. Thus, it requires a lot of computational resources and takes a long time to train. Table 2 summarizes the configuration of DDQN hyper-parameters of the first experiment. We run the model on three machines in parallel, with 20000 episodes for each environment (game). Unfortunately, all the machines crashed before reaching 5000 episodes. Then, we changed the hyperparameters, as shown in Table 3. In the second experiment, the model successfully completed the training phase of 5000 episodes. To ensure that the agent is learning, we computed the mean squared error and plotted it (see figures 5, 6, 7).

Table 2: Experiment 1 Hyper-parameters setting of DDQN

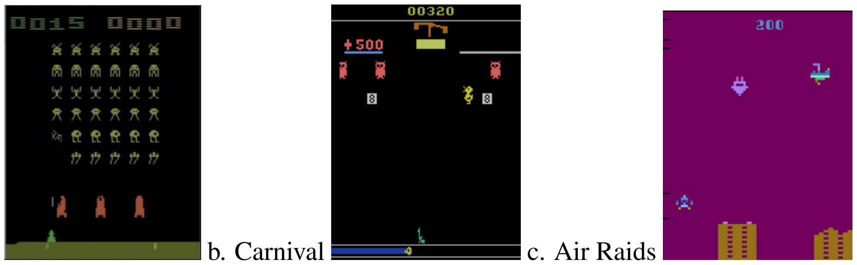
Parameter	Value
Episodes	20000
Replay memory capacity	1000000
Initialization of the replay memory	1000
Decay constant	3500
Range of random frame skipping	3 - 5
Range of number of initial random actions	10 - 20
C	15

Table 3: Experiment 2 Hyper-parameters setting of DDQN

Parameter	Value
Episodes	5000
Replay memory capacity	20000
Initialization of the replay memory	256
Decay constant	1200
Range of random frame skipping	3 - 5
Range of number of initial random actions	10 - 20
C	15

6 Results and Discussion

Using DQN instead of DDQN results in the agent getting stuck on one side without performing an action, as figure 4 shows. Accordingly, we used the DDQN algorithm for an agent to perform better.



a. Space Invader b. Carnival c. Air Raids

Figure 4: Agent Sticking Using DQN.

Comparing the three environments, we can notice from figures 5, 6, 7 that the network is learning as the loss keeps decreasing. The agent is learning well, especially in a space invader environment followed by an air raid and then the carnival. The average cumulative loss at 2500 episodes for the space invader is around 2, the air raid is around 35, and the carnival has a high loss, around 80. The cumulative score of each environment is shown in figure 5, 6, 7, respectively.

Comparing the agent with a random policy with the trained agent, as shown in histogram and kernel density estimation plot, in Space Invaders, the trained agent is a bit better than the random agent. However, in the Carnival environment, the random agent performed better than the intelligent agent. Finally, for the Air Raids environment, the agent is barely better than the random agent.

The results obtained are for 5000 episodes. However, training for a larger number of episodes will result in an intelligent agent performing better than a random agent. Additionally, all three environments have the same network hyperparameters. Nevertheless, each environment should be trained with its hyperparameters.

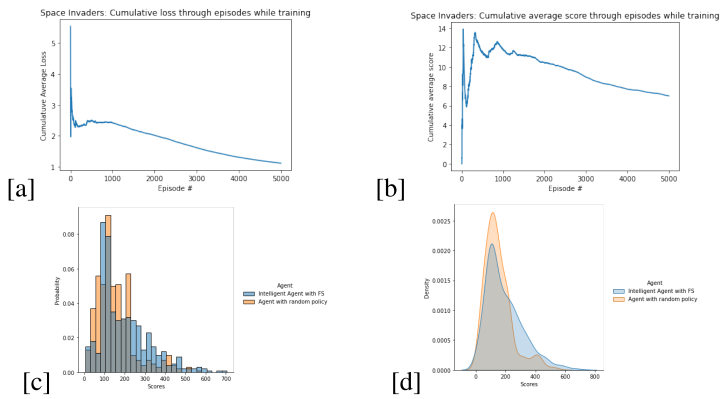


Figure 5: Space Invaders Evaluation. a) Cumulative Loss, b) Cumulative Average Score, c) Histogram Density Estimation, d) Kernel Density Estimation.

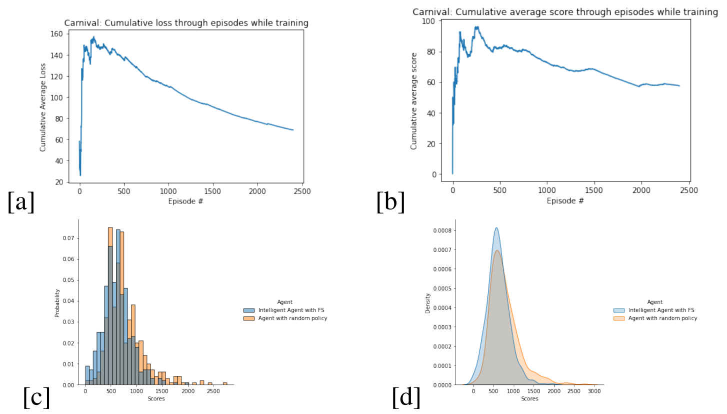


Figure 6: Carnival Evaluation. a) Cumulative Loss, b) Cumulative Average Score, c) Histogram Density Estimation, d) Kernel Density Estimation.

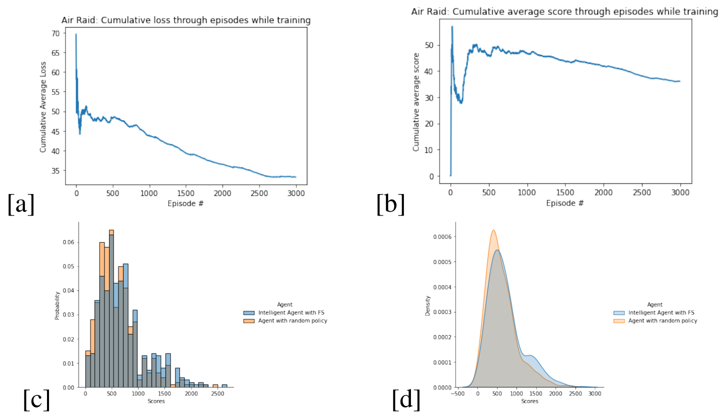


Figure 7: Air Raids Evaluation. a) Cumulative Loss, b) Cumulative Average Score, c) Histogram Density Estimation, d) Kernel Density Estimation.

7 Conclusion

This paper introduced several modifications to the paper proposed by Deep Mind titled "Playing Atari with Deep Reinforcement Learning" [3]. The results were based on 5000 episodes; however, the agent's performance can be improved by training the agent for a large number of episodes. Moreover, the network parameters can be further optimized using grid-search or genetic evolution algorithm [11]. Furthermore, instead of using random memory initialization, we can prioritize experience that replays important transitions more frequently instead of random transitions [7]. Finally, after achieving high agent performance, we can implement the idea of transfer learning to transfer knowledge from one environment to another for the agent to adapt to different environments [12].

References

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

[2] Sudharsan Ravichandiran. *Hands-on reinforcement learning with Python: master reinforcement and deep reinforcement learning using OpenAI gym and TensorFlow*. Packt Publishing Ltd, 2018.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[6] Richard S Sutton and Andrew G Barto. Reinforcement learning: an introduction mit press. *Cambridge, MA*, 22447, 1998.

[7] Ionel-Alexandru Hosu and Traian Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*, 2016.

[8] Nicolas Maquaire. Are the space invaders deterministic or stochastic? 2020.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[10] Clare Chen, Vincent Ying, and Dillon Laird. Deep q-learning with recurrent neural networks. 2016.

[11] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[12] Akshita Mittel and Purna Sowmya Munukutla. Visual transfer between atari games using competitive reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.