Testing DQN variants with Space Invaders

Nicholas Antonio Carroll*
Department of Computer Science and Engineering
Alma Mater - University of Bologna
Bologna, Italy
nicholas.carroll@studio.unibo.it

Abstract

In the few last years a lot of research has been conducted in reinforcement learning using videogames as a testbed. In this project the Double DQN reinforcement learning algorithms have been tested with the videogame "space invaders" and results are compared with the ones from the original paper. Then another algorithm, Duel Double DQN is implemented and it's results are compared to those of Double DQN.

1 Introduction

1.1 Space Invaders

Space Invaders is a classic arcade videogame initially released in 1978, and it was later released for the home console Atari 2600 in 1980. The game quickly became iconic and was one of the best-selling early videogames, greatly influencing successive titles.

The game is quite simple: the player controls a spaceship that can move on the bottom of the screen while many alien spaceships descend in a zig-zag pattern down the screen. The player must shoot at the alien space ships to destroy them while dodging the aliens' incoming shots. Every time one destroys an alien, the speed of the others increases. The player wins if he manages to shoot down all enemy ships and loses if they manage to touch the ground or if he is destroyed by their shots three times. There is also another kind of spaceship, the mothership: a small purple sphere-like object that moves at highs speeds at the top of the screen, which, if destroyed, gives a large number of points. Lastly, there are three orange "shields" that can block incoming shots.

Each episode corresponds to a game of space invaders, divided into one timestep per frame. At each time step, the agent can take one of six different actions: move right, move left, fire, fire right (a combination of fire and move right), fire left (fire and move left), or noop (no operation). The emulator executes the action, and the reward is calculated.

1.2 OpenAI Gym

In this project, OpenAI Gym was used to train the agent to play Space Invaders as it offers a standardized and easy to set up and use environments to speed up development. There are various available versions of Space invaders, and the chosen one represents the state of the game with an RGB image with size 210x160x3, but there is also a version that uses the game's 128 bytes of RAM as input. The original paper tested both versions, but only agents for the first one have been developed in this project.

*

Preprint. Under review.

1.3 Preprocessing

To preprocess the input, openAI Baselines has been used. It is a library that offers a quick way to set up openAI Gym environments with a wrapper that provides various useful tools to manipulate the state. In particular, it:

- Reduces input frames size from 210x160 to 84x84. This alteration drastically reduces the observation space, reducing network size.
- Transforms frames from RGB to grayscale to reduce the state size.
- Stacks frames: the last four observations are stacked to form the game state. This is useful because networks take decisions based on the previous sequence of frames. For example, by observing only a single frame, it would be impossible to understand what direction a spaceship is moving. However, by having multiple successive frames, we can understand if it is going right or left and intercept it.

2 Architecture

2.1 Deep Q Network

Deep Q Network (DQN) is the ancestor of Double DQN and Duel DQN. It is an algorithm that improves Q-learning, which had trouble dealing with very large state sizes such as the one in the current environment. DQN uses a deep neural network that approximates the action-value function.[2]

2.2 Double Deep Q Network

DDQNs are an improvement over DQNs as these can be unstable during training as the target is not fixed but changes with every training step. To improve this, we use a second network called target network, which is identical to the main model, except that its weights are copied at fixed intervals from the original network and not modified the rest of the time. The target network is used to choose which action to take. The Architecture of the network is identical to a standard DQN and is composed of the following layers:

- nput layer with an input of 84x84x4.
- A convolutional layer with 32 filters with size 8x8, stride 4 and relu activation.
- A convolutional layer with 64 filters with size 4x4, stride 2 and relu activation.
- A convolutional layer with 64 filters with size 3x3, stride 1 and relu activation.
- A Densely connected layer of 512 units.
- A Densely connected output layer of 6 units.

2.3 3.3 Duel Double Deep Q Network

DDDQN is an extension of DDQN, which bifurcates the network in two separate heads, called value stream, which represents the state value V(s), and advantage stream, which represents the state-dependent advantage A(s,a). These streams then are combined to estimate the state-action value Q(s,a). [3] The dueling network aims to learn which states are valuable without learning the effect of each action for each one of them. The Architecture of the Duel DQN network is composed by the following layers:

- nput layer with an input of 84x84x4.
- A convolutional layer with 32 filters with size 8x8, stride 4 and relu activation.
- A convolutional layer with 64 filters with size 4x4, stride 2 and relu activation.
- A convolutional layer with 64 filters with size 3x3, stride 1 and relu activation.
- A Densely connected layer of 512 units.

The Value stream:

- A Densely connected layer of 512 units.
- A Densely connected layer of 1 unit.
- A RepeatVector layer that repeats it's input 6 times.

The Advantage stream:

- A Densely connected layer of 512 units.
- A Densely connected layer of 6 units. from each of these values their mean is subtracted.

Lastly the advantage and values streams are added together to give the output.

3 Development

3.1 Experience Replay

Experience replay is a technique proposed by [2]; it randomly samples experiences from a replay memory consisting of sets of values (State, action, reward, next state) instead of taking the most recent experiences to train the network. This helps to deal with the convergence problem of Q learning, as choosing highly correlated actions limits the exploration of the state space. It is possible to train the network with randomly sampled experiences because DQN is an off-policy algorithm.

3.2 Rewards

Rewards are clipped to be in the range [-1,1] to ensure that the magnitude of gradients during backpropagation is bounded. [1]

3.3 Frame Skipping

As suggested in [2] also frame skipping is used. It consists in picking an action only on the kth frame instead of doing it for every frame, and then this action is repeated on the skipped frames. This reduces training time by a factor of k. In this project, a value of k=4 was used, as suggested in [1].

3.4 Exploration

Epsilon greedy was used as an exploration policy. It balances exploration and exploitation, choosing the best option with a probability of 1-epsilon and a random move in the other cases. The value of epsilon linearly decreases from 1 to 0.1 over 1 million steps to ensure that the network begins by exploring a lot and then, in the later stages, chooses mostly the best actions.

3.5 Hyperparameters

To train the algorithm, Adam was used as an optimizer, with a learning rate of 0.0002. The discount factor was 0.99. The experience replay buffer was 100'000, and a batch of size 32 was used during experience replay.

4 Training

Each agent has been trained for 1 million steps on a laptop. Ideally, they would have been trained for longer as they still showed signs of improvement, but due to hardware and time constraints, it proved impossible. Also, Google Colab was tested, but the training speed did not improve, so it was abandoned due to its usage time limitations.

5 Results

5.1 Random Baseline

Before testing the networks a test was conducted with only random moves. The mean reward over 1000 episodes was 2.6.

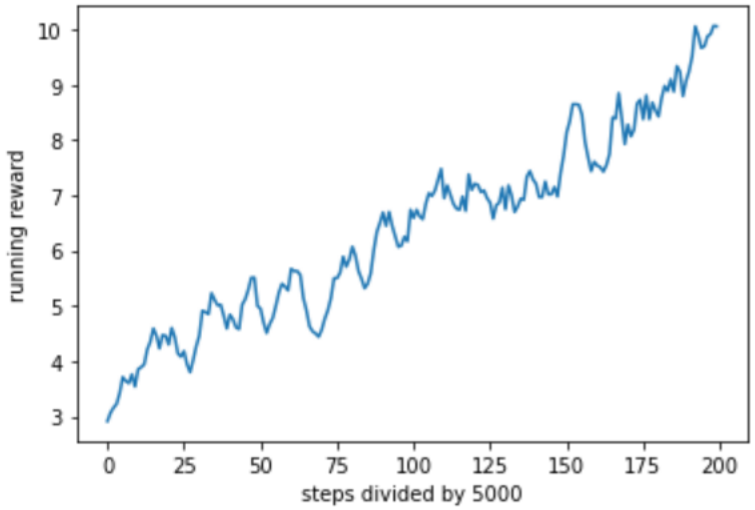

Figure 1: DDQN running reward.

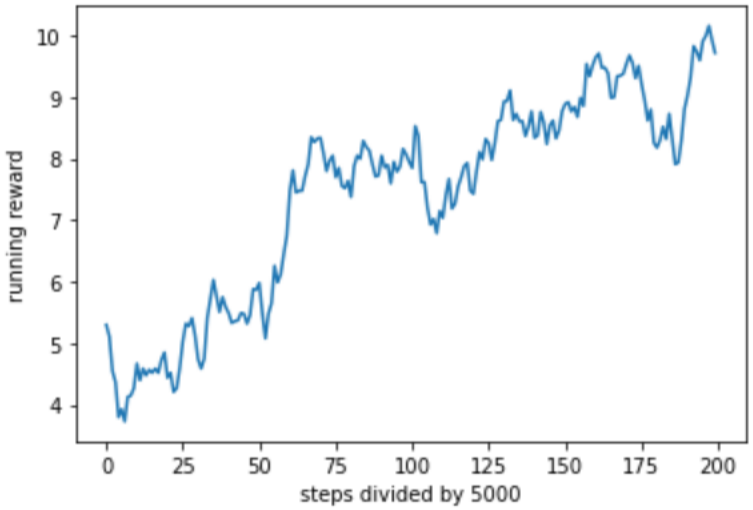

Figure 2: Duel DDQN running reward.

5.2 DDQN

We see in figure 1 that During training, the running reward of the last 100 episodes for DDQN grew steadily over time, up to it's peak of 10.07.

When testing, the network results varied slightly based on the value of Epsilon that was chosen. Each parameter value was tested over 100 episodes, and the best result was reached with an epsilon value of 0.1, obtaining a maximum reward of 35 and a mean reward of 11.21. With Epsilon = 0.05, the maximum reward did not change, but the mean was 10.53. Lastly, with epsilon = 0, the mean reward was 11 with a maximum of 23.

5.3 Duel DDQN

AS can be seen in figure 2, the running reward for Duel DDQN started slow but then grew rapidly up to its peak of 10.16.

Similarily to DDQN, also Duel DDQN had the best results when testing with an epsilon of 0.1. It had an average reward of 9.72, with a maximum of 30. Epsilon = 0.05 had similar results with identical

maximum reward and a mean reward of 9.44, while epsilon = 0 had a mean reward of 7.8 and a maximum of 18.

5.4 Considerations

Overall results are very similar with both networks, being only marginally better with DDQN. These results can not be considered final as both networks were still growing rapidly at the end of the training, which could not go further due to time constraints.

References

[1] Desai, Nihit and Abhimanyu Banerjee. "Deep Reinforcement Learning to play Space Invaders." (2017).

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing Atari with Deep Reinforcement Learning."

[3]Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot and Nando de Freitas. "Dueling Network Architectures for Deep Reinforcement Learning."