Uppsala University

Reinfrocement Learning

1RT747

September 21, 2024

Learning to play Pong with DQN

Authors:
Otema Yirenkyi
Tsion Samuel Tegegn

Contents

1 Introduction

Deep Q-Learning (DQN) is a fundamental algorithm in the field of reinforcement learning (RL) that has garnered significant attention due to its success in solving complex decision-making tasks. RL trains agents in an environment to make decisions in maximizing the rewards. DQN is grouped as a model-free RL family, which means it does not require any prior knowledge of the environment's dynamics.

It uses the Q-network, which is a neural network, to approximate the Q-function. The Q-function calculates all the future rewards for an action in a given state. The training procedure includes updating the Q-network parameters iteratively in order to minimize the differences between the predicted and target Q-values. One advantage of DQN is that it uses experience replay, a technique that is used to store past experiences in a replay buffer and sample batches in order to train the Q-network. This is useful in a way that it stabilizes training by breaking correlations between consecutive samples and makes use of past experiences effectively. The target network, which is a separate copy of the Q-network in DQN, is used to generate target Q-values during training.

It will be updated periodically along with the Q-network in order to prevent training instability.[1] This project examines the research paper ”Playing Atari with Deep Reinforcement Learning” by Mnih et al. (2013) and applies the Deep Q-learning algorithm to the CartPole-v1 and ALE Pong-v5 environments provided by the Gymnasium framework.

2 Experiments

2.1 CartPole-v1.

The CartPole environment, provided by OpenAI's Gymnasium, simulates a cart and pole system, where the goal is to balance a pole on top of a moving cart by applying appropriate left or right forces. The state space consists of four variables: Cart position, Cart velocity, The angle of the pole measured from the vertical position, and The rate of change of the pole angle. The agent can take two actions: move the cart to the left or move it to the right. The main objective is to prevent the pole from falling over by applying the appropriate force to the cart. The episode terminates if the pole angle exceeds a certain threshold or the cart position moves outside a predefined range. The task we have undertaken in this project is to develop an agent that learns to balance the pole effectively by intelligently selecting actions based on the observed state.

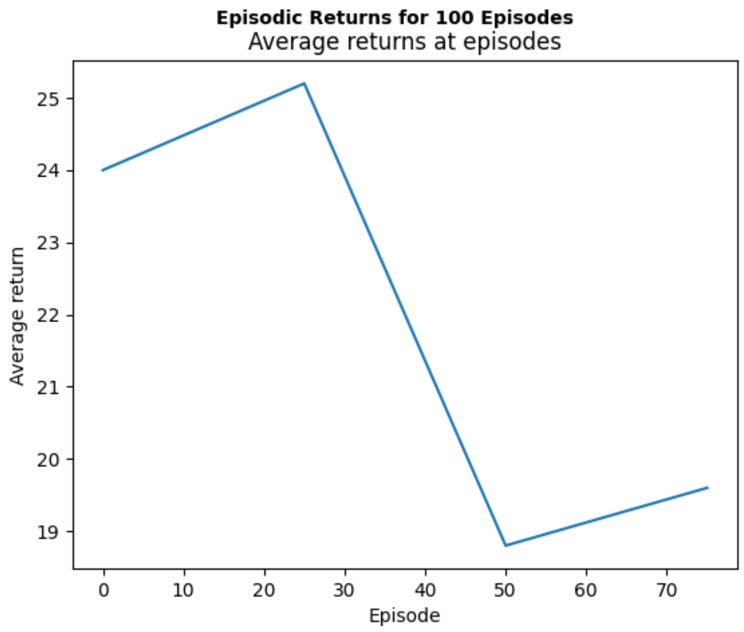

Figure 1: Epoch 1, plot of average returns during training for 1000 episodes

In our experiment, we conducted training sessions for the Cartpole-v1 environment over two different durations: 100 episodes and 1000 episodesusinfg the default configuration listed in the table.

Parameter	**Value**
Memory Size	50,000
Number of Episodes	1,000
Batch Size	64
Target Update Frequency	100
Train Frequency	1
Gamma	0.95
Learning Rate	1×10^{-4}
Epsilon Start	1.0
Epsilon End	0.05
Anneal Length	10^4

Table 1: Hyperparameters for the experiment

Throughout these sessions, we adjusted several configurations to optimize performance. One important aspect of our training strategy involved updating the weights of our target Deep Q-Network (DQN) every 100 steps or iterations. For the initial 100-episode training phase, the agent underwent learning process, gradually adapting its behavior to navigate the Cartpole environment. During this period, the agent took 1785 steps to achieve convergence, demonstrating a steady progression in learning. Upon evaluation, we observed the best performance of the Cartpole model at episode 28, where it achieved a cumulative reward of 27 over the course of 100 episodes. Notably, we conducted evaluations every 25 episodes to assess the agent's progress and adjust our training approach accordingly.

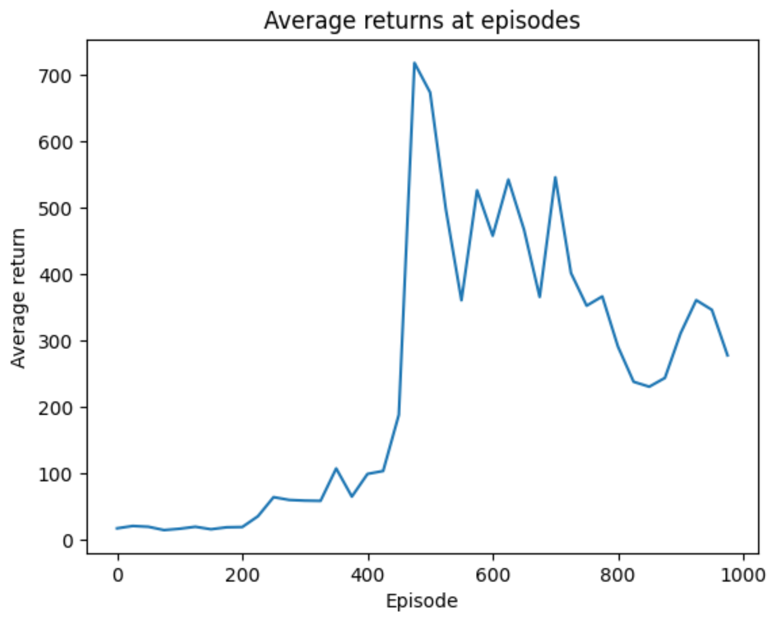

Figure 2: Epoch 0, plot of average returns during training for 1000 episodes

Furthermore, we extended the training duration to 1000 episodes to further refine the agent's capabilities. Similar to the 100-episode phase, the agent here again underwent extensive training, gradually refining its decision-making processes over a more extended period.During this extended training phase, the agent took 184516 steps to reach convergence, reflecting the continued learning and adaptation within the environment. Evaluation at regular intervals revealed the peak performance of the Cartpole model at episode 510 , where it achieved a cumulative reward of 705 over the 1000-episode duration.Throughout both training phases, our iterative evaluation approach provided valuable insights into the agent's learning progress and guided the refinement of our training methodologies

2.2 Hyper parameter Experiments

We conducted a series of experiments by varying one parameter at a time while keeping the others constant. Here, we present the four experiments that we believe have a significant impact on the model's performance.

Batchsize In this experiment, we increased the batch size to 96, three times the previous value. Ideally, using a larger batch size can be beneficial as it leads to more stable gradient estimates and reduces variance. However, it is crucial to find the appropriate batch size for our model. A batch size that is too large can result in slower model convergence. As observed, the increased batch size in this experiment led to lower model convergence compared to the default setting.

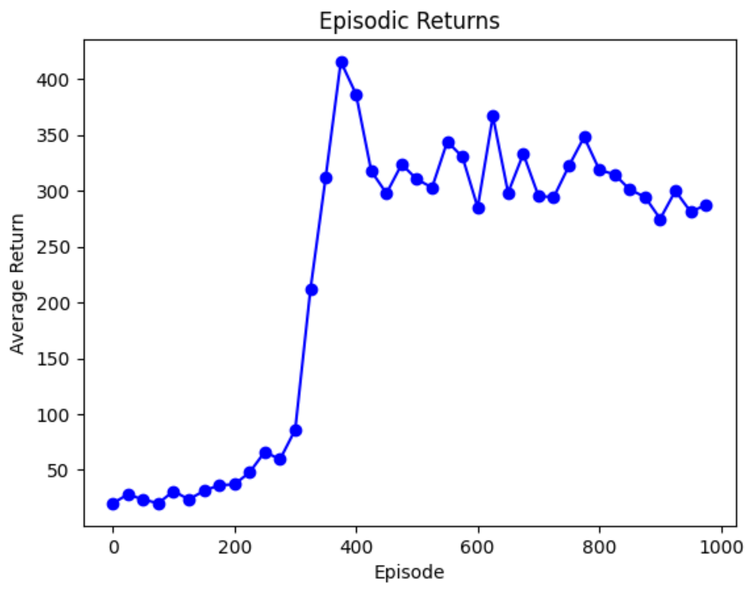

Figure 3: Batch size 96, plot of average returns during training for 1000 episodes

Learning rate In this experiment, we reduced the learning rate to 0.001. Lowering the learning rate means the model learns more slowly compared to the previous setting, updating its parameters more gradual-ly. This can be beneficial as it helps prevent overfitting and leads to more stable performance. But as we can observe in the figure, there is an occasional spike between episodes 800 and 900, indi-cating that the model discovered a particularly rewarding strategy during this interval.

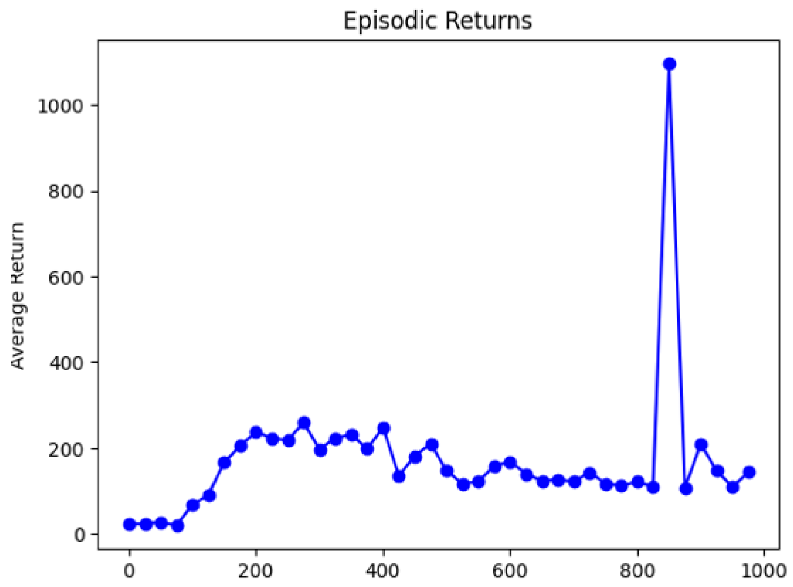

Figure 4: Average returns from episodes. learning rate 0.001

Target update frequency The target update frequency helps provide consistent Q-values to stabilize training. A low frequency means the targets are updated often, which can be challenging if it's too low as it makes convergence difficult for the Q-network. Conversely, with higher values, the target network is updated less frequently. In this experiment, we increased the target update frequency from 100 to 500. This means the target network is updated less frequently, while the main Q-network is updated more often. This approach slightly prevents rapid changes in the Q-values, leading to more stable training.

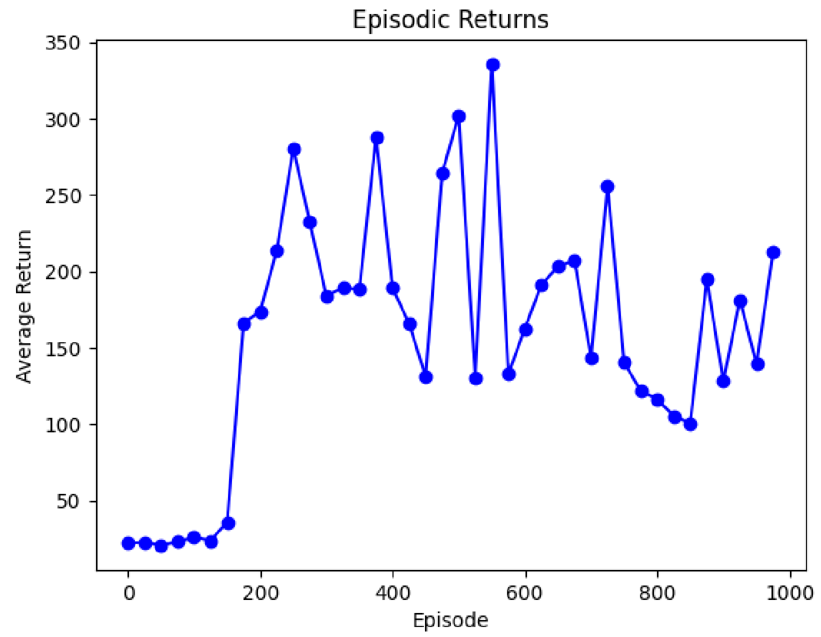

Figure 5: Average returns from episodes, target update frequency 500

Episilon start and end In this experiment, we have set the epsilon to start at 1 and end at 0.01 instead of 0.05. The initial exploration rate indicates that the agent is choosing actions randomly at the beginning, which is set to the maximum value of 1. This ensures that the agent explores a wide range of actions. Reducing the epsilon end means that the agent is choosing actions randomly only 1% of the time at the end of training. In other words, it means the agent is highly relying on its learned policy rather than random selection at the end of the training.

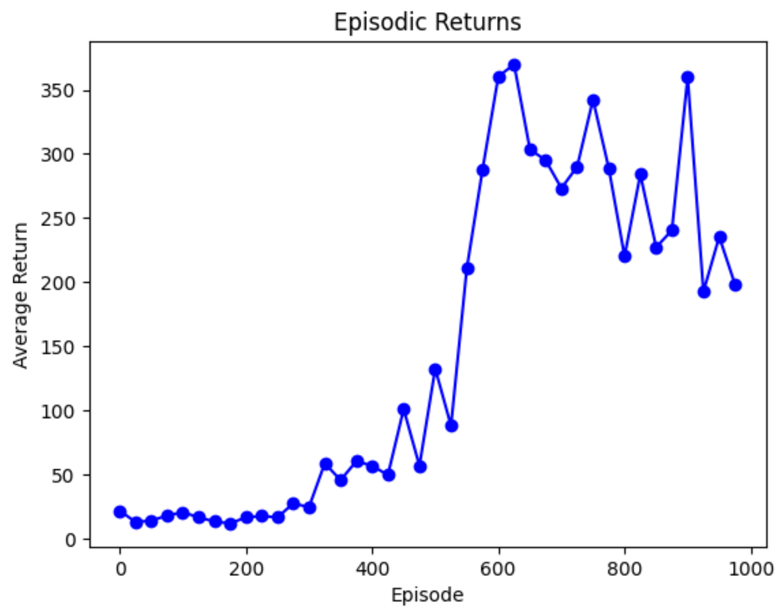

Figure 6: Average returns from episodes, Episolon end 0.01

2.3 ALE/Pong-v5

The ALE Pong-v5 environment, also provided by OpenAI's Gymnasium, emulates the game of Pong, where two players control paddles on opposite sides of the screen, attempting to hit a ball back and forth. The objective in this environment is to control one of the paddles to successfully hit the ball and prevent it from passing the other paddle. The state space in ALE Pong-v5 typically consists of the raw pixel values of the game screen, representing the visual information perceived by the agent, represented as Box(0, 255, (210, 160,3), uint8), where the pixel values are [0-255] and 3 channels of 210 by 160 frames. The actions available to the agent usually include moving the paddle up or down to control its position. The action space in ALE Pong-v5 offers 6 discrete actions, including NOOP (No operation), FIRE (unused), RIGHT, LEFT, RIGHTFIRE (equivalent to action 2), and LEFTFIRE (equivalent to action 3). Notably, only actions 2/4 (RIGHT) and 3/5 (LEFT) induce movement in the agent.

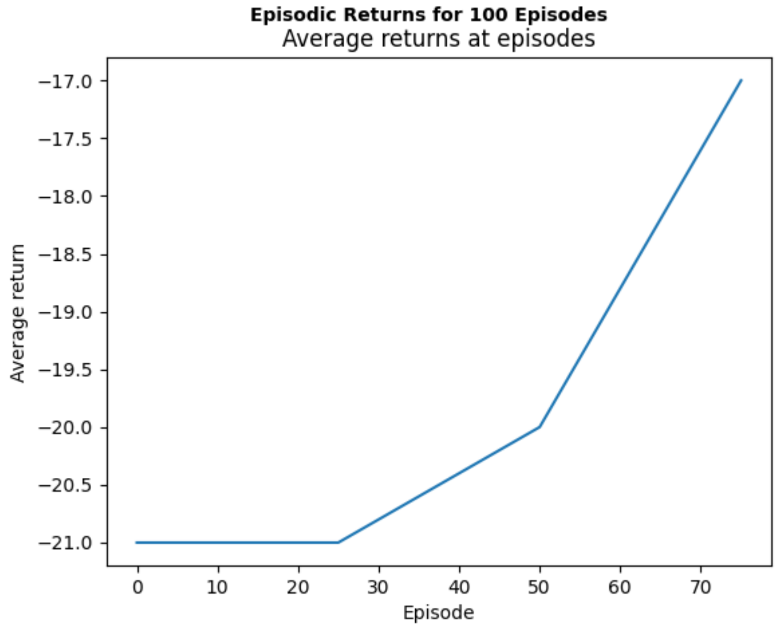

Figure 7: Average returns from episodes, epoch 2

We proceeded to conduct the Pong experiment, consisting of 100 episodes, to further evaluate our Deep Q-Network (DQN) agent's performance. During the 100-episode training phase,the agent underwent extensive learning. Throughout this process, we adjusted various parameters to optimize the agent's performance and training efficiency.Upon analysis, we determined that the agent took 152,191 steps to achieve convergence, signifying its adaptability and capacity to learn the Pong environment. This convergence metric provided insights into the agent's learning trajectory and training methodologies.The peak performance of the Pong model was observed at episode 80, where it demonstrated exceptional Replay and recorded a cumulative reward of -17.2 over the course of 100 episodes. We conducted evaluations at regular intervals, specifically every 25 episodes, to monitor the agent's progress and fine-tune our training strategies based on performance feedback.Selection of the training parameters, including episode duration and evaluation frequency, was influenced by computational constraints. Due to limitations in GPU runtime on Google Colab, we conducted the experiments only in 100 episodes using CPU resources.

2.4 Hyperparameter expeirments

Learning rate

In this experiment, we lowered the learning rate to 0.001. This reduction caused the model's learning process to slow down, leading to more gradual updates to its parameters. However, as shown in the fig-ure, the model failed to converge within the 100 episodes due to the excessively slow learning rate. as we can observe in the figure, there is an occasional spike between episodes 800 and 900, indicat-ing that the model discovered a particularly rewarding strategy during this interval.

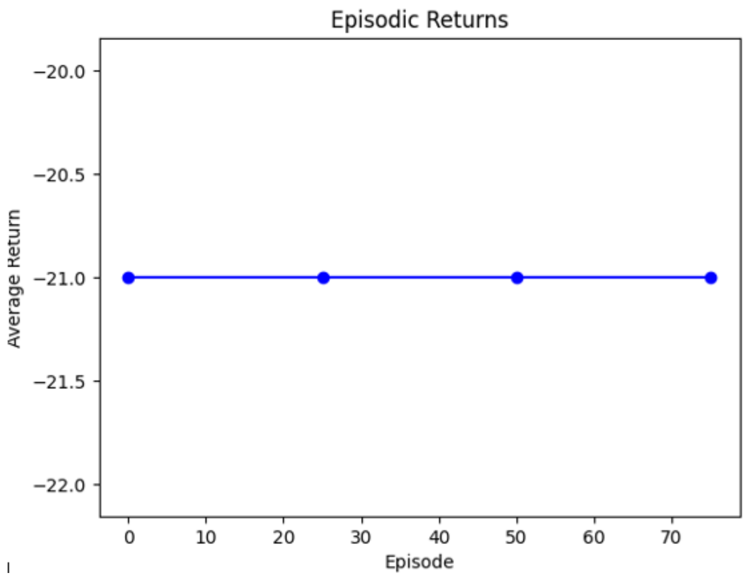

Figure 8: Average returns for 100 episodes with learning rate 0.001

Target update frequency In the later experiment, we increased our target update frequency from 1000 to 4000, intending for the main Q network to be updated more often. However, the model still failed to learn. As illustrated in the figure, the mean return remained consistent throughout the entire episode.

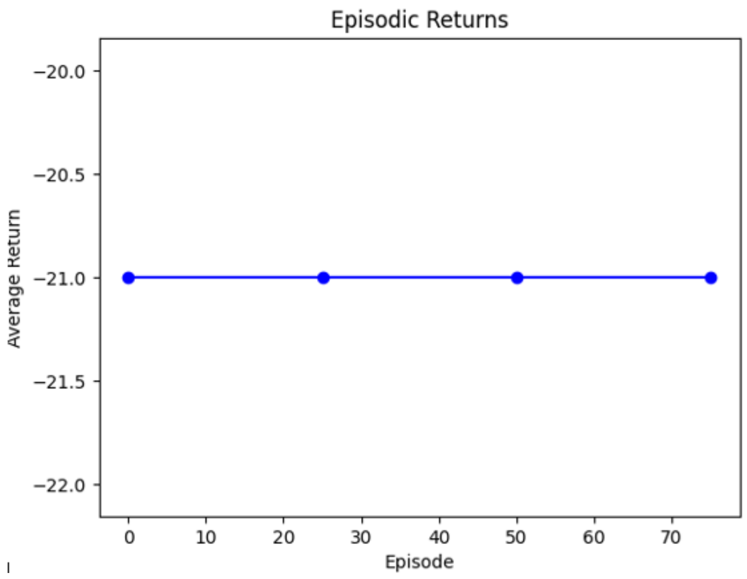

Figure 9: Average returns for 100 episodes with target update frequency 4000

Target update frequency=200 and gamma=0.95 When conducting an experiment, we changed the target update frequency to 200 and the gamma value to 0.95, aiming for the agent to prioritize immediate rewards more than in the default setting. However, we still observed the same return values.

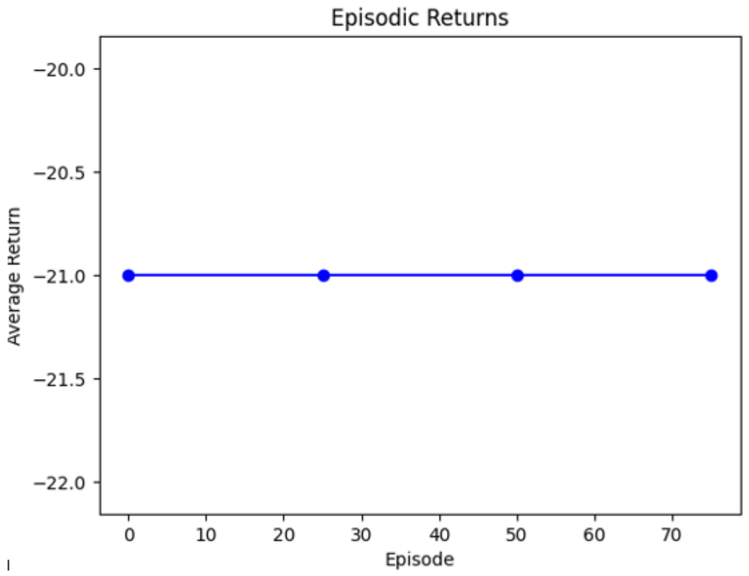

Figure 10: Average returns for 100 episodes with Target update frequency=200 and gamma=0.95

3 Discussion and Conclusion

Both experiments achieved best mean return value with the default parameters. We observe that as training progresses, achieving the model's best mean return after every 25 episodes takes increasingly longer. This is because the number of steps required before the next episode update continues to grow. From figure 1, we observe that training for a fewer number of episodes (100) does not allow the model sufficient time to achieve higher returns. In contrast, as seen in Fig 2, extending the training to 1000 episodes provides significantly better results, indicating that longer training periods enable the model to learn more effectively and achieve greater returns. From figure 3 for pong, we see a similar trend where training for only 100 episodes does not allow the model to achieve substantial returns. Additionally, we notice that the training terminates after 152,191 steps when the current mean return gets less than the best mean return. In future experiments, we will run the training for multiple epochs and extend the number of episodes. This Which will likely enhance the model's learning process. This approach allows the agent to take more steps, refine its weights iteratively, and achieve optimal convergence, ultimately improving the overall performance of the model.

References

[1] Deep Reinforcement Learning Part 1 : DQN;. Accessed: 2024-05-15. https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b.