Learning to play Pong with DQN

Nazifa M. Shemonti Salman Khan Şihab S. Bayraktar Yue Wang
Uppsala University

Abstract

This report covers the concepts of deep Q-network and the experimental results of the agent conducted on CartPole and Pong environments.

1 Background and Literature Review

In this section, we briefly discuss the work of Mnih et al. on developing the deep Q-network (DQN). DQN is a new deep learning (DL) model for reinforcement learning (RL) to learn difficult control policies for Atari 2600 computer games. The authors' proposed artificial agent estimates the action-value function using Q-learning through a neural network function approximator. This type of the online Q-learning algorithm that implements convolutional neural network (CNN) is referred to as the deep Q-learning method.

The extent of CNN's capability to learn successful policies was compared with the performance of an expert human player in two successive articles: In [1], their proposed approach was tested on seven games and it gave state-of-the-art results in six of them; in [2], their DQN agent achieved more than 75% of the human score on 29 out of 49 games.

2 Overview of the Deep Q-Network

The DQN agent interacts with environment $\mathcal{E}$ through actions, observations, and rewards. At each time-step t, the agent is only given a vector of raw pixel data representing the current image x_t from $\mathcal{E}$. It carries out an action a_t to the emulator that alters the emulator's internal state and game score. The change in game score means that it receives a reward r_t. In order to learn game strategies, many finite sequences of actions and observations $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$ are selected —each consisting of distinct states. The behavior policy $\pi = P(a|s)$ is obtained by choosing an action that maximizes future rewards when the agent interacts with the environment. The cumulative future reward is computed by summing the discounted rewards $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ where γ. Also known as the return, it is the discount factor per time-step and T is the termination time-step. The objective is to compute the optimal action-value function (equation 1):

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi] \tag{1}$$

A neural network with weights θ estimates the action-value function $Q(s,a;\theta) \approx Q^*(s,a)$.

The improvements which make DQN efficient for the Atari emulator are:

1. Preprocessing the input frames (210×160 pixel images, 128-colour palette) into reduced dimensionality (84×84, greyscale) with m most recent frames stacked together.
2. Implementing the architecture (Fig. 1), in which the input layer is followed by three convolutional layers and two fully connected layers with a single output for each legal action.

3. Using a neural network function approximator instead of using iterative update. The training phase involves minimizing consecutive loss functions $L_i(\theta_i)$. The target y_i changes as learning proceeds and changes network weights.
4. Using stochastic gradient descent to optimize the loss function instead of determining the full expectations of the gradient when the loss function is differentiated.
5. Adjusting the action-values (Q) towards target values that are only periodically updated. Thereby it reduces correlations with the target; if not, it easily diverges because the target continuously moves.
6. Clipping the error term from the gradient to be between -1 and 1 to stabilize the algorithm further.
7. Utilizing the *experience replay* technique to store experiences at each time-step $e_t = (s_t, a_t, r_t, s_{t+1})$ and pooling them together into a *replay memory*. In the training phase, these replays are drawn uniformly from the memory queue. The randomization over the data removes the correlation in the observation sequence and smoothens over changes in the data.

From tackling the caveats of high-dimensional sensory input to updating the target network periodically with the weights of the online network, the aforementioned measures are critical to the success of DQN.

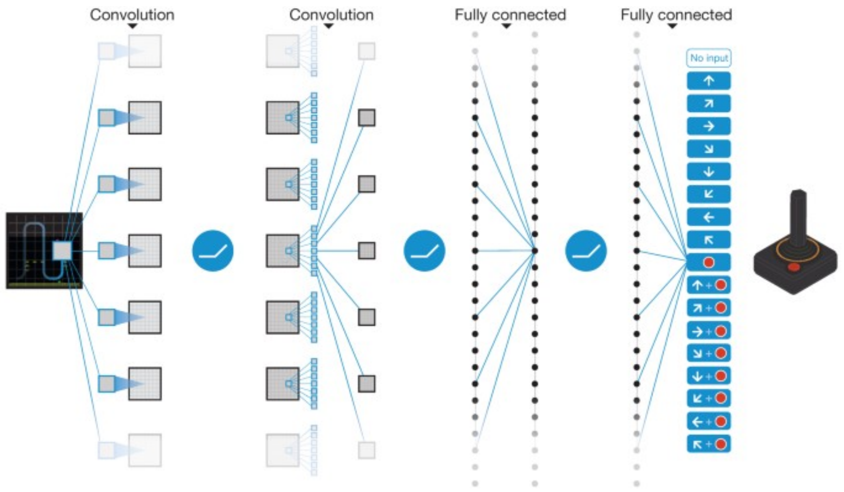
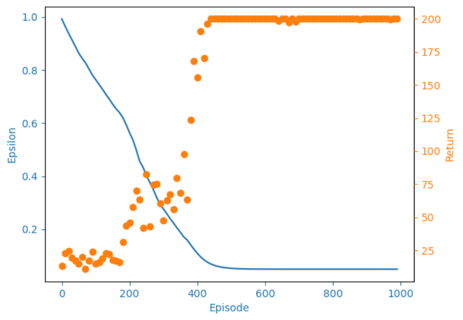


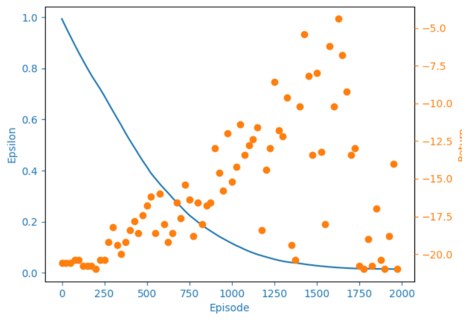
Figure 1: Schematic illustration of the convolutional neural network reproduced from [2] (only 2 convolutional layers shown, 3 stated in the text).

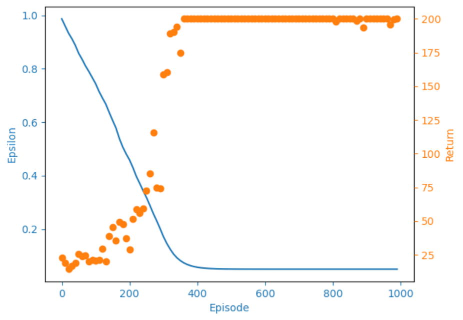
3 Experiments

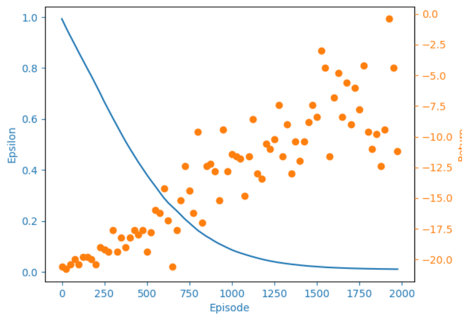
Balancing a cartpole requires moving a cart left or right to stabilize the pole attached to it in upright position. The agent receives a reward of +1 for every incremental time step. The environment is terminated when the pole falls over or the cart is far from the center by a specific distance. On the other hand, pong is a table-tennis-like game where the agent can remain stationary, or slide horizontally upwards or downwards.

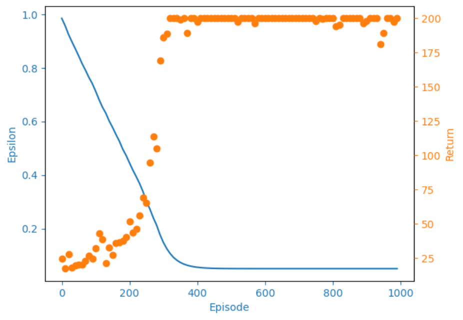
The provided code skeleton is completed according to the literature by building the network, the preprocessing, and the training prerequisites. Primarily, it is used to evaluate the mean return obtained over 2000 episodes. We find that our choice of hyperparameter values for pong environment results in a mean reward of -0.4 in a wall time of 4 hours on the GPU assigned by Google Colab. We compared rewards obtained by varying the *target_update_frequency* (Fig. 2).


(a) Episode return of CartPole when $target\_update\_frequency = 10$


(b) Episode return of Pong when $target\_update\_frequency = 10$


(c) Episode return of CartPole when $target\_update\_frequency = 500$


(d) Episode return of Pong when $target\_update\_frequency = 500$


(e) Episode return of CartPole when $target\_update\_frequency = 1000$

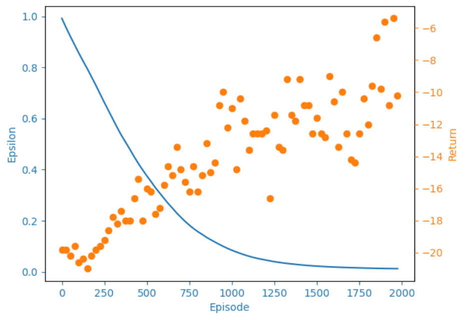

(f) Episode return of Pong when $target\_update\_frequency = 1000$

Figure 2: Behavior of epsilon and average reward with episodic runs

4 Discussion

We can see from the CartPole traces that when $target\_update\_frequency$ is higher, the mean reward is reached earlier. In case of training the agent to learn to play pong, the agent only reaches -20.0 at the beginning of the training. However, it is able to reach -0.4 when $target\_update\_frequency = 500$ after running for 4 hours.

References

[1] 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. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236