APPLYING DEEP Q-NETWORK TO A CUSTOM WAREHOUSE ENVIRONMENT

Zeyu Sun
CentraleSupelec
France
zeyu.sun@student.ecp.fr

Pierre Sevestre
CentraleSupelec
France
pierre.sevestre@student.ecp.fr

July 23, 2020

ABSTRACT

In this project, we wanted to use the recent dramatic advances in decision-making using reinforcement learning on our own environment. After reviewing the prominent methods on classical benchmark, we created a new environment, where the agent has to drop packages of various type into a warehouse, with the objective to minimize the operating time and energy consumption. Our code is available at https://github.com/Pierre28/WarehouseRL.

1 Problem Definition - Related Work

1.1 Context and motivations

The global objective is to learn a control-policy, that is for an agent to make sequential decision, without exact model of how the world work. Although in most cases, and particularly the warehouse hereafter, the environment can be modelled as a Markov Decision Process with known dynamics; most often one can only sample from it, or exact computation are computationally infeasible. To account for challenging situation, we will consider the samples can only be obtained from the environment, thus studying model-free reinforcement learning strategies.

1.2 Background

Deep Q-Learning [1] As most of the classic reinforcement learning problems, we consider an agent which interacts with the environment ϵ by taking actions: $A = \{1, ..., K\}$.At each discrete time step, the agent will chose, based on an observation or a sequence of observations. The goal of the agent is to choose its actions in the way that maximise the future rewards.

One consider the sequence of all actions and observations up to the current time step, $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$. Each of these sequences correspond to a Markov decision process (MDP), and the original problem can be addressed with classical reinforcement learning methods for MDP. One define the future discounted *return* at t with game termination T, that the agent will maximize: $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$. The classical optimal action value function:$Q^*(s, a) = max_\pi E[R_t \mid s_t = s, a_t = a, \pi]$ is defined, as the maximum expected return using the policy π, mapping the sequences to actions. This optimal value function obeys the Bellman equation. Knowing the optimal value of the sequence s' at all step for all actions a', the optimal strategy to maximise the expected value of $r + \gamma Q^*(s', a')$ is:

$$Q^*(s, a) = E_{s'\epsilon}[r + \gamma max_{a'} Q^*(s', a') \mid s, a]$$

By updating the value function at each step i, we obtain a value iteration algorithm which converges to the optimal value function when i tends to infinite.

$$Q_{i+1}(s, a) = E[r + \gamma max_{a'} Q_i(s', a') \mid s, a]$$

[1] Heavily inspired from [1]

In practice, a function approximator is used to estimate the action value function. It can be linear or non linear. In our case, we used a neural network function approximator with weights θ(Q-network). A Q-network is trained by minimising a loss function:

$$L_i(\theta_i) = E_{s,a\rho}[(y_i - Q(s,a;\theta_i))^2]$$

θ_{i-1} from the previous iteration is fixed while optimising the loss function. The target for step i is :

$$y_i = E_{s'\epsilon}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a]$$

The target depends on the weight of neural network. The loss is differenciable with respect to the parameter of the neural network, and one may obtain the following gradient :

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a\rho;s'\epsilon}[(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)]$$

Hence, Q-learning algorithm is performed by using the stochastic gradient descent, updating weights at every step and replacing the expectations by samples from the behaviour distribution ρ and the emulator ϵ.

Double Q-Learning Greedy policy w.r.t estimated action-state value can yield maximization bias in finite-sample learning. To remove this bias, [2] considered two estimates Q_1 and Q_2, one estimate to select the action, i.e. $a^* = \arg\max_a Q_1(s,a)$, another to estimate the value of a^*: $Q_2(s,a^*)$. Using independant samples to estimate the value and chose the max yields an unbiased estimate.

Dueling architecture Another key concept, introduced in [3], by designing a network suited for model-free RL. They created two separate streams after classical the encoding, one dedicated to estimating the state-value, and the other for the advantage function. It is thus possible to estimate the value of a state without exploring each actions. It is particularly interesting when each action have similar impact at a given state.

1.3 Gym

Gym [2] is a toolkit provided by Open AI for developping reinforcement learning application. In particular, it provides a unified framework suited to build environments and test algorithm performances easily. Many environmnet are already implemented in this library, including the toy example of cartpole, or more advanced Atari games that will be studied later. We implemented the aforementioned warehouse environment using this framework.

2 Experiments

2.1 Environment setup

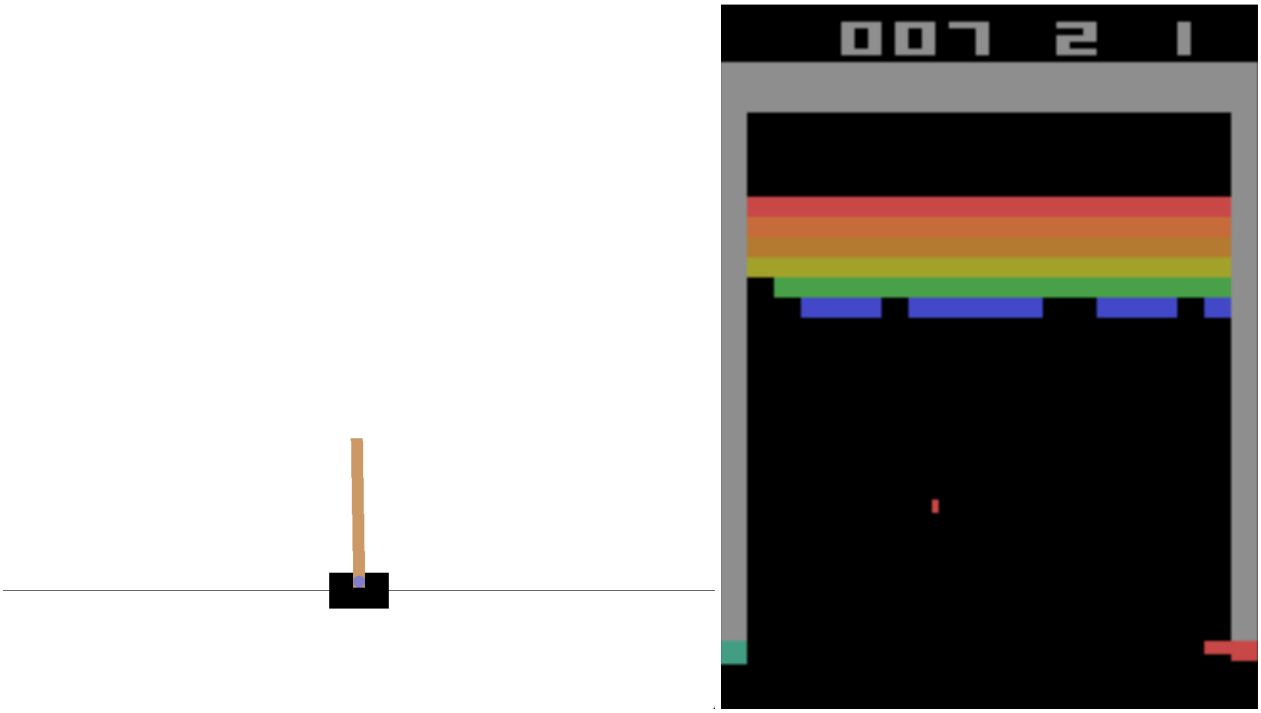
Cartpole

In this toy experiment, a pole is attached to a cart, and the objective is to train the agent to chose on which side to apply a force to, to prevent the pole from falling. The action space is discrete and contains two actions : moving to the left or to the right. The observation space contains for continuous variables referring to the position, velocity, rotation and rotational velocity of the cart and the pole. The agent receives a reward of +1 after each step, each episode having at most 200 steps. Figure 1a illustrates this environment.

Attari Breakout

Atari 2600 games present interesting challenges to evaluate RL methods, with its high dimension input and diverse strategies. In Breakout, a wall of brick lies on top of the screen, and the agent has to break this wall using a bouncing ball. The input is a 210x160 RBG frame. To lower computational cost, a prepossessing is performed, reducing the scale, cropping and turning the frames to a single gray-scale channel. To account for the dynamic of the problem, 4 frames are stacked together to form each input. The observation space is the a 84 x 84 x 4 frame. The action space is of dimension 4, representing respectively the idle, fire of new game, move right, and move left actions. Figure 1b illustrates this environment. During each episode, the agent has 3 lives, and will receive a reward for each broken tile.

[2] http://gym.openai.com



(a) Cartpole (b) Atari - Breakout

Figure 1: Visualisation of Cartpole and Atari Breakout environment from Gym

Warehouse

The objective is to train an agent to optimally place packages into a particular warehouse. This warehouse contains stacked shelves, that can slide away from the center to free space for forklift to place a package to a given spot. Due to symmetry, only half of this warehouse is studied, represented in 2a. The objective is twofold : the agent has to minimize both the average dropping and retrieval time, while minimizing the energy consumption. These stacked shelves allow the warehouse to occupy the minimum surface possible, so one need the agent to guarantee this reduction in space occupation is not at the expense of energy consumption to move the shelves, or the global operating time.

In this simulation, as represented in 2, we considered 3 levels of shelves on both sides, each shelves containing 2 rows of multiple spots. Each spot is represented by a square in the rendering. The single spot on the top-left of the warehouse corresponds to the package to drop.

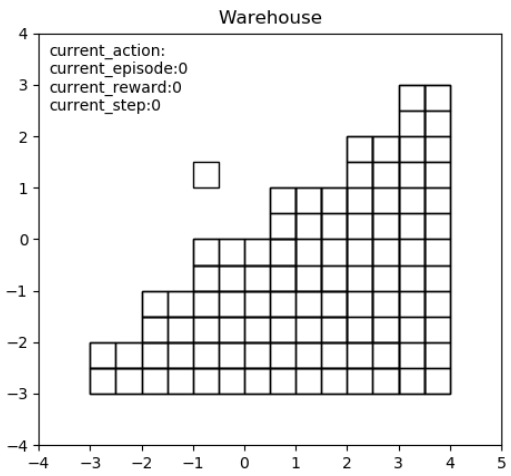
To train the agent, we designed the following scenario : Different types of packages will be considered, with varying weights and appearance frequency. The warehouse is first randomly partially filled according to this appearance frequency of packages up to a given proportion of the total number of spots. Then, two actions will occur alternatively, simulating the delivery of packages to the warehouse by a supplier, followed by the retrieval of packages as a backlog of orders, simulating the shipping to clients. More precisely, the delivery of packages consist of a list of packages spawning on the queue sampled from the frequency of appearance of each package. Once every package on the queue is dropped, the list of packages types to remove is sampled from the very same frequency of appearance, and the actual packages removed are the easiest to access of each type.

The variation in frequency is represented by the color shade : light shade represents packages with low frequency of appearance, darker shade higher frequency. Hatching account for the varying weight, with the following mapping :

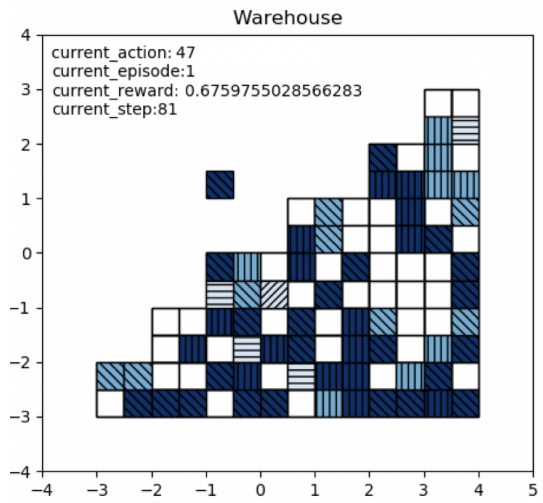
- Diagonal hatching: very light
- Back diagonal hatching = light
- Vertical = Heavy
- Horizontal = Very Heavy

We designed multiple reward systems. In the final version, the agents receives a reward after dropping each package, that is the sum of two normalized terms. The first accounts for the operating time, and the second for the energy consumption. Each term is ranging from 0 to 1. Dropping a package to an invalid position, i.e. one where a package already lies, gives a -1 reward. An additional reward is given periodically, accounting for the efficiency of the current overall placement, i.e. the average normalized operating time and energy consumption of all currently placed packages.

With this setting of three levels of shelves, there are 99 spots to drop packages to, resulting in an observation space of dimension 100 (99 packages in the warehouse, and the package to be dropped), and an action space of dimension 99.



(a) Empty Warehouse



(b) Partially filled Warehouse

Figure 2: Visualization of the warehouse environment. Blank spots are empty, and color shade account for frequency of appearance of the dropped package. Hatching account for the weight.

2.2 Results

Cartpole

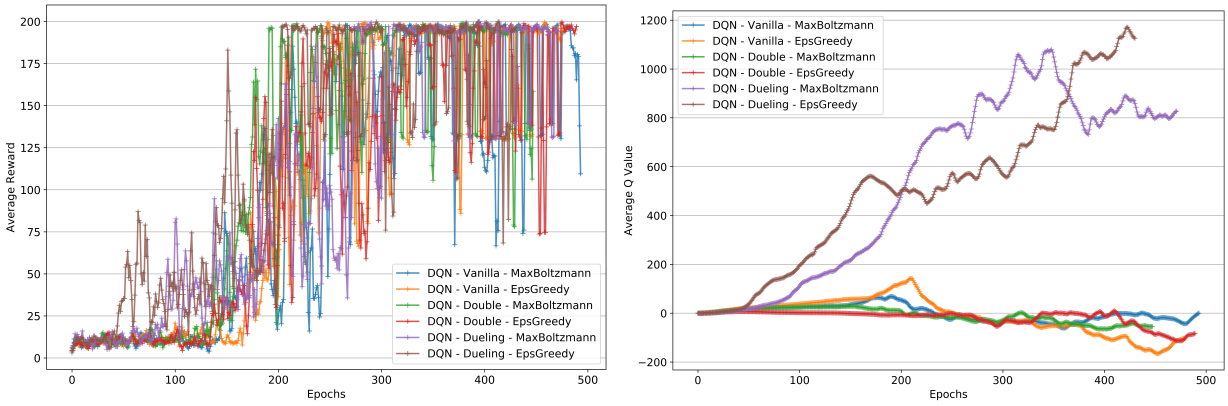
Since training time is reasonable, we were able to try multiple models on this cartpole environment. We experimented with the three variations of the method, that are vanilla Deep Q Network, along with double and dueling. Both Epsilon greedy policy and MaxBoltzmann policy are used with method. We kept the default parameters to train these models. As depicted in 3a, after around 200 epochs, every model manage to complete the each episode, reaching a reward of 200. As suggested in [1], we also plotted the average maximum action-value of a collection of states, that proved to present a clearer view of the convergence. However, in 3b, we can see exploding values using the duelling network, that is not detrimental to the efficiency of the agent.

Atari Breakout

In this section, we only considered double and dueling network, since they offer the best performances on the benchmark, and because the important training time (around 20h per experiment), prevented us from trying additional setup. The traditional epsilon greedy policy was used in both experiments. For both methods, we kept the parameters used in the original paper. As on can see in 4, both method follows the same trend. The agent start to make meaningful actions around 2000 epochs, and reaches an average of 50 tiles broken per episode after 4000 epochs. After visualizing the trained agent play, we notice at a good level, but still encounters failure when the ball has too much speed. This could be solved by continuing the training of the agent.

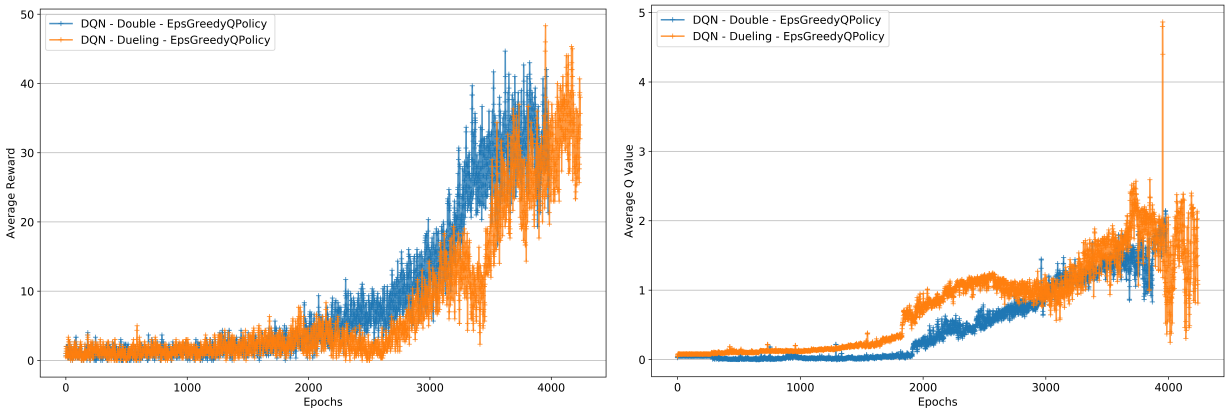
Warehouse

Due to its satisfactory performances on Breakout, we only considered Double Q Network in this section. We found it very difficult to find a converging model. Most of the time, the training collapsed to a single value, with the agent always predicting the same spot to drop the package to, without respect of the particular state the agent was in. We



(a) Average Reward (b) Average Q-value

Figure 3: Training results on cartpole for various models



(a) Average Reward (b) Average Q-value

Figure 4: Training results on Atari Breakout for double and duelling network

tried multiple architecture for the neural network, multiple policies, and parameters. We also tried to refine the reward strategy so the rewards would be less sparse, to give a stronger signal to compute the loss. None of these enabled us tackling the collapsing issue. We considered the difficulty came from the high dimension of the action space, 99 in the experiment, corresponding to each spot of the warehouse. Thus, we eventually implemented a trick to help the agent learn : at each iteration, we create a mask to hide the actions that were not feasible; i.e. dropping a package at spot where another packages lies. The agent then just had to choose the best spot out of all possible spots. This trick, hiding instead of giving negative rewards to impossible actions, prevented the training from collapsing, and allowed us to have a functional agent. A sample of the experiments can be found in 5, with the masking trick referred to as MaskingBoltzmannQPolicy.

3 Conclusion

Although Deep Q-Network proved to be very efficient and outperformed many classical methods in multiple benchmark, with some outstanding performances, we found it hard to apply it to a new task. The design of the observation space, action space, reward system, and the neural network and RL-method has to be though carefully. In particular, since the critical points lies in obtaining a converging model, the action space as simple as possible to lower the difficulty of the task. Regular feedback should be made to the agent, reducing sparsity of the reward. We eventually used simple neural network architecture, and set the parameters of method and policy matching existing similar solution in the literature.

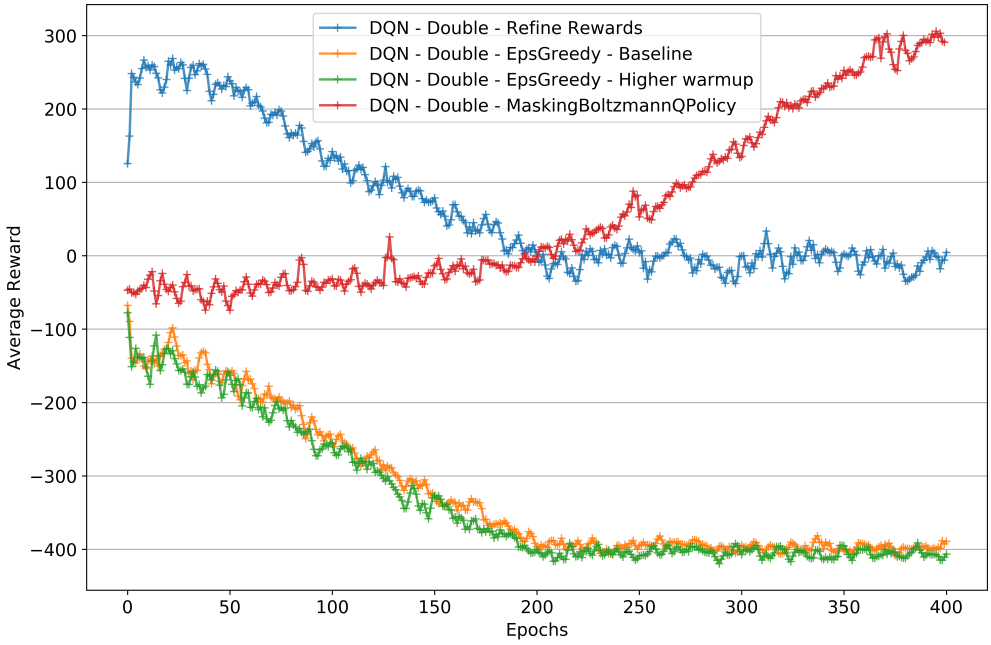


Figure 5: Average Reward on Warehouse environment

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[2] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[3] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.