Playing Atari with Deep Reinforcement Learning

Matjaž Zupančič Muc
mm1706@student.uni-lj.si
Faculty of Computer and Information Science
Vecna pot 113
Ljubljana, Slovenia

ABSTRACT

In this seminar we implement and train an agent to play two Atari games of varying complexity (Pong and Freeway) using an off policy, model free method, known as Deep Q Learning. We are able to achieve great performance on the simpler game and decent performance on the more complex game. We expose the main advantage and weakness of this approach.

KEYWORDS

Deep Q Learning, Double Deep Q Learning, Atari, Pong, Freeway

1 INTRODUCTION

[1] introduces a deep Q learning Algorithm. The novelty of this approach was developing a model free agent, which learns from a high dimensional state space. Method out-performed the best existing reinforcement learning methods on 43 out of 49 Atari games without incorporating any additional prior knowledge about Atari games. [2] introduces a modification of the classic Q learning known as double Q learning, which prevents overestimation of action values. In this seminar both algorithms were implemented and evaluated on two games. The two games have an identical actions space but are of varying complexity.

2 MATERIALS AND METHODS

2.1 Background

Deep Q Learning Algorithm builds on top of classic Q-learning which is impractical for environments with a large state space. Classic Q learning assumes that we can visit each state enough times to decide on the optimal action to take in each state. *DQN* uses a deep neural network which is used to approximate the optimal Q-function. Neural network with weights θ approximates $Q(s,a) \approx (s,a,\theta)$.

Experience replay is introduced to remove correlations in the sequence of observations, it also prevents the agent to get stuck in a poor local minimum. The concept is based on storing experiences $e_t = \{s_t, a_t, r_t, s_{t1}\}$ at every time step. The agent stores experiences in a data set $D = \{e_1, e_2, e_3, ..., e_t\}$. Data-set D is formally called replay memory buffer and is of size N. When its size exceeds N, old experiences are overwritten with new ones. A sub set of experiences is sampled at random from D at every time step t and used to update the online network Q.

A separate neural network (target neural network) is used for generating the targets $y_j = r_j + \gamma \cdot max_a Q^{'}(\phi_{j1}, a^{'}, \theta^{-})$. Target network gets updated using online network every C steps. Generating the target values using an older set of parameters θ^{-} adds a delay between the time an update to Q is made and the time the update affects the targets y_j, making divergence or oscillations less likely. [1] introduces the following algorithm.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

[2] showed that Deep Q Learning tends to suffer from overestimation of action values. Since Q-learning selects optimal action by picking the maximal Q value for a given state, it tends to prefer overestimated Q values. The max operator in standard Q-learning and Deep Q learning, uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this they they suggest we use one network Q to select the max action and the second network $Q^{'}$ to evaluate the value of that action.

2.2 Methods

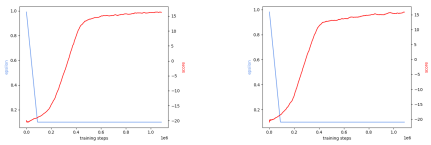
2.2.1 Action and State space. Actions space of both games is composed of three actions, the agent can either move up, down or stay still. The size of the state space is $256^{84\times84\times4}$. The state space is the set of screen pixels (Image size: 84×84, Grayscale: 256 gray levels, 4 consecutive images).

2.2.2 Preprocessing. Pixel images are pre-processed before we learn from them. Images are downscaled and converted to grayscale, flickering of pixels is removed, the last 4 observations of environment are stacked.

2.2.3 Neural Network. Structure of the neural network is taken from [1]. The network has 3 convolutional layers and 2 fully connected layers. Network uses *RMSProb* optimizer and a *MSE* loss function. The agent is trained using epsilon-greedy approach

2.2.4 Pong. A *DQN* agent is trained to play the game of Pong. We let the agent train until its score stabilizes. Figure 1 shows the running average of score over time. We can see that most of the learning occurs as soon as the epsilon drops down to its minimum

of 0.1. To see if action overestimation plays a role in pong, we will also train a *DDQN* agent. Results are similar, but the slope of performance seems slightly steeper at the last few episodes, which suggests that agent could pick up more skill, with more training time. Figure 1 shows the distribution of actions that a trained agent performs. Note that action 0 represents no action, action 1 represents move up, action 2 represents move down.



(a) DQN **(b) DDQN**

(c) DQN, Action distribution

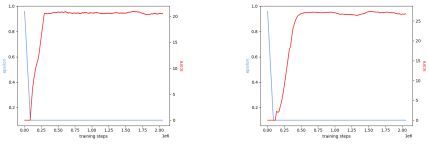
Figure 1: Pong, performance after 500 **episodes**

2.2.5 Freeway. The goal of the game is to safely cross ten lanes of freeway traffic as many times as possible in a set amount of time. When the player is hit by a vehicle the game does not restart, but the player is set back 2 lanes and spends a fraction of time recovering before it can move again. *DQN* and *DDQN* are trained for 1000 episodes, using a replay buffer of 70000. Figure 2 shows the running average of score over time and action distribution. We can see that *DQN*s score tops out at around 21 points, but the agent learns to simply move straight. In this game going straight makes sense most of the time, by stopping or going back the agent looses time. In fact an agent that always moves straight achieves an average score of 21. *DDQN* score stabilizes at around 28. We can see that agent picks forward action most of the time, but also learns that sometimes performing action 0 (staying still instead of just running for the prize) or even performing action 2 (stepping back) makes sense.

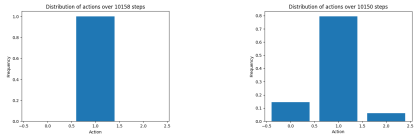
Finally we let the *DDQN* agent play another 3000 episodes, to see if it can pick up more skill. The learning curve is shown on Figure 3. It seems like the agents performance does not increase much from 2 to 5 million steps, but it seems like it does increase from 5 to 8 million steps. If we take a look at action distribution, agent seems to pick action 2 more frequently than action 0.

3 RESULTS

Table 1 shows average score over 10 episodes for all agents (trained agents use $\varepsilon = 0.01$). *DQN* denotes and agent trained for 1 million steps in case of pong & 2 million steps in case of freeway. *DDQN* denotes an agent trained for 2million steps in case of pong & 2 million steps in case of freeway. *DDQN* with subscript 8 denotes an agent trained for 8 million steps. *NAIVE* denotes agent which always selects forward action in freeway.

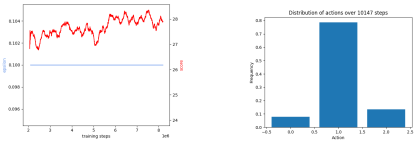


(a) DQN **(b) DDQN**



(c) DQN, Action distribution **(d) DDQN, Action distribution**

Figure 2: Freeway, performance after 1000 **episodes**



(a) DDQN **(b) DDQN, Action distribution**

Figure 3: Freeway, performance after 4000 **episodes**

Game	*DQN*	*DDQN*	$DDQN_8$	*NAIVE*	*RANDOM*
Pong	20.8	20.8	/	/	-21.0
Freeway	22.0	31.5	32.5	21.2	0

Table 1: Average score over 10 episodes

4 CONCLUSION

We implemented and evaluated *DQN* and *DDQN* agent on two games of varying complexity. We show that both methods are capable of achieving decent results. Being able to transfer an agent from one game to another is a main benefit of these methods. Sample inefficiency is the main weakness of these methods. We showed that as soon as the complexity of environment increased the training time necessary to achieve decent results increased substantially. Further research could include kick staring a model with a model trained on a different game, to see if training time decreases.

REFERENCES

[1] 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. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.