Pong using Deep Reinforcement Learning

Matjaz Zupancic Muc
Faculty of Computer and Information Science
Ljubljana, Vecna pot 113
Email: mm1706@student.uni-lj.si

***Abstract*—In this project we attempt to train an agent to play Pong using an off policy model free method, known as Deep Q Learning with a few adjustments. We manage to develop an agent that beats the best human player with just a few hours of training using a personal computer.**

I. Introduction

The core of this project is based on paper[1], where Deep Q Learning Algorithm was presented. The novelty of this approach was developing a model free agent, which learns using high dimension observations of the environment, such as pixels and scores to train a reinforcement agent. Method outperformed the best existing reinforcement learning methods on 43 out of 49 Atari games without incorporating any additional prior knowledge about Atari games. The second paper[2] introduced a modification which prevents over-estimations of action value function. Here we will evaluate both algorithms on the Pong game and compare the results.

March 3, 2022

II. MATERIALS AND METHODS

A. Background

Before we proceed to methods and results, we should provide an explanation of main concepts. The Deep Q Learning Algorithm builds on top of classic Q-learning which is impractical for environments with a large state space. A neural network is used to learn from a high dimensional sensory data (large state space). We use a neural network to approximate the optimal Q-function: $Q(s,a,\theta) \approx Q^*(s,a)$. Standard Q-learning is known to diverge when the state space becomes large, the following two concepts were introduced to deal with this issue.

First concept known as experience replay is introduced to remove correlations in the observation sequence, it also prevents the agent to get stuck in a poor local minimum. The concept is based on storing experiences at every time step. Experience is defined as $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, where s_t is the state of the environment at time t, a_t is the action performed by the agent at time t, r_t is the reward obtained by performing a_t in s_t, s_{t+1} is the resulting state in which the agent finds its self after performing a_t at s_t. The agent stores experiences in a data set $D_t = \{e_1, e_2, e_3, ..., e_t\}$. 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 called a mini batch is sampled at random from D at every time step t and used to update the online network Q.

Second modification is the use of a separate network (target network) for generating the targets $y_j = r_j + \gamma max_a Q^{'}(\phi_{j+1}, a^{'}, \theta^{-})$ in the Q-learning update. Target network gets updated using online network every C steps. Using two networks makes the algorithm more stable. 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 much less likely.

The following is the algorithm introduced in paper[1] which we implemented and later trained on Pong.

Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
     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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Paper[2] showed that Deep Q Learning tends to suffer from overestimation of action values. They pointed out that since Q-learning selects optimal function by picking the maximal Q value for a given state, it tends to prefer overestimated Q values. In case over-estimation effects all action values equally such problem will not occur. If, however the over estimations are not uniform across states, the policy gets skewed. In fact they showed that noise is always present, it may come from using function approximators (neural networks) or from the environment (randomness in environment). 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 proposed the idea of decoupling the selection of actions from the evaluation. They suggest we use one network Q to select the max action and the second network Q^- to evaluate the value of that action. We change the definition of target as follows: We no longer use $y_i = r_i + \gamma max_q Q(s_{i+1}, a_{i+1})$, instead we define target as $y_i = r_i + \gamma Q^-(s_{i+1}, max_q Q(s_{i+1}, a_{i+1}))$.

B. Methods

The pong actions space is composed of two actions, the agent can either move up or down. The size of the state space is $256^{84*84*4}$, for comparison 10^{82} is the number of atoms in universe. The state space is the set of screen pixels:

- Image size: 84 x 84
- Grayscale with 256 gray levels
- Consecutive 4 images

Before we can start learning from pixels we have preprocess the images. Images are downscaled and converted to grayscale, flickering of pixels was removed, the last 4 observations of environment are stacked. Structure of the neural network used is similar to the one presented in paper[1]. The network has 3 convolutional layers and 2 fully connected layers. Network uses RMSProb optimizer and a MSE Loss function. We trained the agent using a replay memory buffer of size $N = 50000$, which ended up using around 13GB of ram. The agent was trained with epsilon-greedy approach, the value of ε decreased from $\varepsilon_{max} = 1$ to $\varepsilon_{min} = 0.1$, using steps size of $\varepsilon_d = 0.00001$. Epsilon decrees as the function of steps and with it the percentage of random actions.

We let the agent train until its score stabilizes. This took around 1000000 steps. No additional tuning of network parameters was performed.

Average score as a function of number of steps is displayed on Fig.1. The figure clearly shows that agent learns over time and that learning is stable, unlike the regular q-learning. Most of the learning occurs when the epsilon drops to around 0.01. The max score is reached in around 600000 steps. The best average score was 16.10.

With no change of network and agent parameters we evaluated the performance of Double Q Learning agent. Resulting graph is displayed on Fig. 2. Again most of the learning occurs when the epsilon drops to around 0.01. The max score is reached at around 500000 steps. The max average score appears to oscillate a bit less when compared to Deep Q Learning agent. The best average score was 16.37.

Even though final score is roughly the same for both agents. The double Deep Q learning agent achieved the max score of 21 in 5 episodes, where as Deep Q learning agent achieved it 3 times.

In the results section comparison of both models as well as comparison to what they achieved in paper[1] is displayed. *DQN* denotes our Deep Q Learning agent, *DDQN* denotes our Double Deep Q Learning agent, *DQN$_P$* denotes Deep Q Learning agent developed in the paper[1], *HUMAN* denotes a professional Pong player (The human performance is the average reward achieved from around 20 episodes of each game lasting a maximum of 5 min each, following around 2h of practice playing). *RANDOM* denotes an agent that only takes random actions.

We can see that our model beats the best human and achieves the best score that is about 1.76 times greater than best human score. The best score achieved by *DQN$_P$* is 18.9, which is quit higher than what we managed to get. Main reason for the difference may be the hardware limitation (in paper they used a memory buffer of size 1000000, where as we could only afford a size of 50000) or the time limitation (their agent trained for much longer).

C. Results

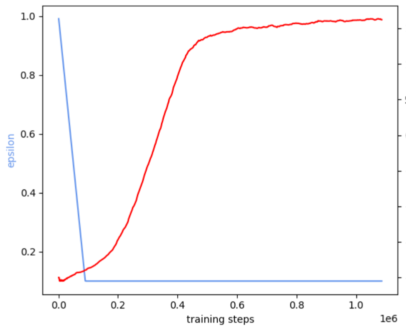

Fig. 1. Deep Q Learning agent performance

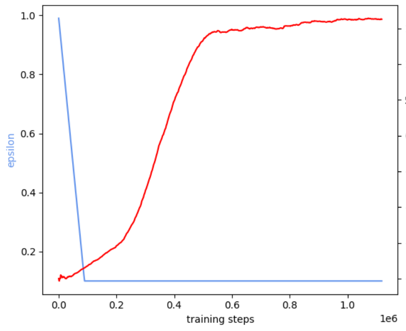

Fig. 2. Double Deep Q Learning agent performance

Agent: DQN	DDQN	DQN$_P$	HUMAN	RANDOM
Score: 16.1	16.4	18.9	9.3	-20.7

TABLE I
PONG GAME BEST SCORES

III. Conclusion

We showed that using Deep Q Learning we were able to exceed the best human performance by training a model strictly from game pixels and scores. Double Q learning did not effect our performance substantially, probably due to the fact that Pong game is relatively simple, since it has a small action space. Attempting to train the agent on a more complex game would require more hardware as well as time resources, but may give us insight into limitations of this method.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529–533.

[2] Van Hasselt, H., Guez, A. Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning.

[3] https://www.youtube.com/watch?v=zR11FLZ-O9Mt=594sab$_channel = LexFridmanhttps://github.com/openai/gym$