Project 1. Navigation

Zhamila Issimova

November 25, 2018

Abstract

In this report implementation of DQN algorithm for navigation problem is described. It contains brief information about this deep reinforcement learning technique as well as design choices which were made to train machine learning agent in Unity environment.

1 Introduction

The goal of the project is to teach an agent to navigate in a large square world, while collecting as many bananas as possible. This is reinforcement learning task, in which agent starts in some state, makes an action (based on its policy), and environment responds to agent by change of state and returning a reward. This reward compliments or returns penalty to agent based on its actions. Since neural network with hidden layers is used as function approximation, it becomes Deep Reinforcement Learning task. Famous DQN algorithm [1] is used to solve this navigation problem.

2 Methodology

2.1 Environment

Navigation problem is represented in "banana" environment, which is provided by The Unity Machine Learning Agents Toolkit (ML-Agents). Since the goal is to maximize the score, a reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. State space has 37 dimensions which includes agent's velocity, along with ray-based perception of objects around the agent's forward direction. Action space is discrete and consists of moving forward or backwards, left or right.

2.2 DQN algorithm

Since action space is discrete and the goal is to maximize output, DQN perfectly fits to solve this problem. DQN algorithm was taught earlier on the program and its solution implementation files ($dqn_agent.py$ and $model.py$) were taken without any further changes. In this version of DQN implementation vanilla algorithm without double networks, prioritized experience replay, dueling network architecture. In this simpler version of DQN only target network and replay buffer were used. Target network is just a copy of Q network to make

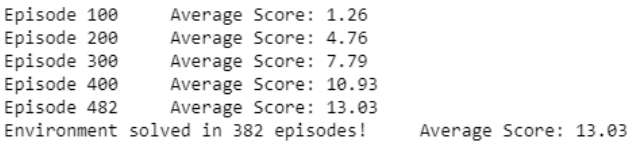

Figure 1: Environment solved in 382 episodes

learning more stable without oscillations. Instead of once-in-nth step update, soft (or sliding) update method was chosen with $\tau = 0.001$

$$\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target} \tag{1}$$

Replay buffer or experience replay is used to break the temporal correlations by mixing more and less recent experience. Buffer size was chosen to be 100,000, and batches of size of 64 were sampled uniformly random. Learning was done with discount factor 0.99 and learning rate was set as 0.0005. Learning is done by backpropagation in the neural net and Adam optimizer. So loss was calculated as Mean Squared Error between Q values of target and local networks.

2.3 Deep Neural Network

As it was stated above, neural net file was taken from DQN module solutions untouched. Hence, in this project, DQN local and target Q networks used the same Qnetwork wich consisted of 3 fully connected layers. As usual the first layer had the same number of input channels as number of states and output channel number was set to 64. Then, this first hidden layer was passed through rectifier linear unit RelU function to get rid of negative values. Second hidden (fully connected) layer had the same number of inputs and outputs and also passed through RelU function. Finally, third fully connected had the same number of output channels as the number of actions. Thus, this deep neural net created an approximation of Q values mapping all states to actions.

3 Results

The number of epochs were set as 2000, but the environment was solved in 382 episodes as it can be seen from Figure 1. The problem was considered as solved if average score over 100 episodes is higher than 13. The average score during whole training can be seen on Figure . As the goal was achieved, training stopped before going through whole 2000 episodes. However, if training was not aborted It could be seen that training curve stabilized after 500 episodes.

4 Conclusion

In this project extensive experience with DQN algorithm was obtained. DQN was adapted to Unity "banana" environment. The goal of navigation project was solved and agent were trained to have average score more than 13. The output was also visualized and score of 22 was obtained. In future, it is possible to extend this work and make agent to learn directly from visualized input using

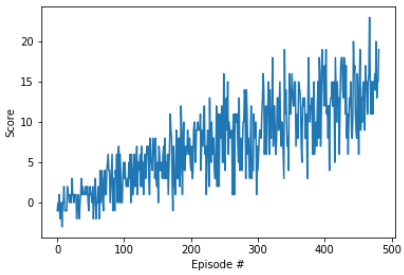

Figure 2: Learning curve

only raw pixels. Also training time can be reduced if recent tricks such as double networks, prioritized experience replay, dueling network architecture are used.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature* vol. 518, no. 7540, pp. 529-533, 2015.