Deep Reinforcement Learning Agent with Prioritized Replay

Daniel Szemerey

Udacity

Author Note

This project is an obligatory delivery for Project I in

Udacity's Deep Reinforcement Learning Nanodegree

Abstract

This report summarizes the work done on Project I of the Deep Reinforcement Learning Nanodegree. Two agents were developed:

Two DQN Agents were developed with PyTorch:

- a **base Agent**, with a Replay Buffer, a separate target Q-Network, with a 2 hidden layer deep network
- an Agent built on top of the base Agent, which utilizes **Prioritized Replay**.

Agents are tested on the Banana environment packaged and provided by Udacity.

Prioritized Replay shows inefficiency in both computational resources and convergence. Further work needs to be done to improve the performance.

Keywords: Artificial Intelligence, Reinforcement Learning, DQN, Prioritized Replay

Table of Contents

Environment

Goal and Reward: Collect yellow bananas (+1.0 point), avoid purple bananas (-1.0 point).

Interaction is episodic (but could be continuous), with a hard step limit of 300.

state space:	**37**
action space:	**4** (forward, backward, left, right)
agents (brains):	**1**

considered solved:	**> +13** avg. over 100 episodes
termination criteria:	**300** time steps
reward:	**+1.0** when collecting yellow bananas **-1.0** when collecting a purple banana

source of the environment: Udacity - Deep Reinforcement Learning engine: unityagents

Agent I – Base DQN Agent

Description and Hyperparameters

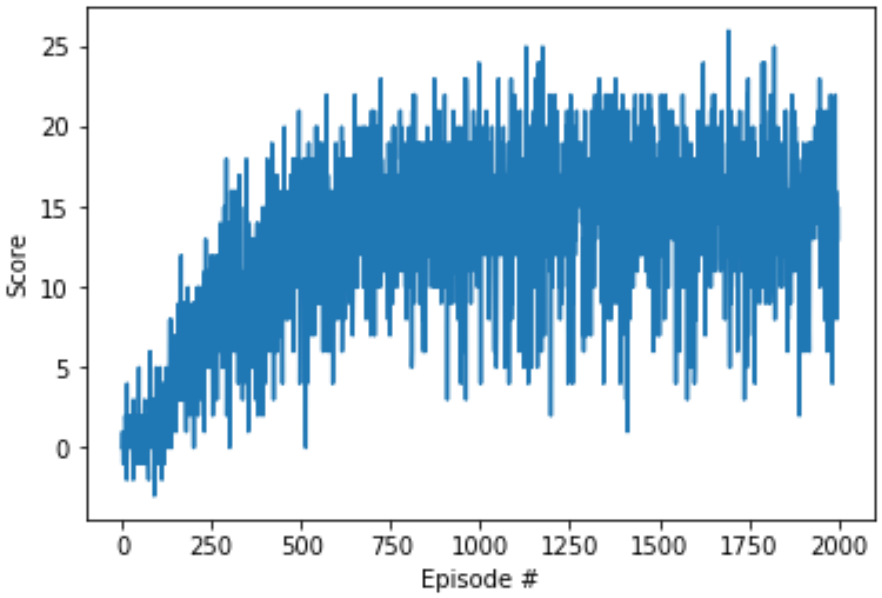
A base agent with a Deep Neural Network of hidden layer size, $N_L(hidden) = 2$ with both size 74 and ReLU activation function.

The agent adds and randomly samples experiences at every fourth step, $t_{update} = 4$, from a Replay Buffer of sample batch size $N_{Batch} = 64$, and buffer size $N_{Buffer} = 100.000$. The epsilon rate of the agent is slowly decreased, resulting in less search by the end of the training.

GAMMA (discount factor)	0.99
TAU (soft update of target network parameter)	1e-3
Learning Rate	5e-4
Target network is updated every [] episode:	4
EPSILON (EXPLORATION - EXPLOITATION)	
Epsilon start	1.0
Epsilon min	0.01
Epsilon decay	0.995

Result

The agent converges steeply (at around 500 episodes) to an average of 13.5 points in 100 consecutive trials, and slowly increases to a maximum of 15.2 points.



Agent II – DQN Agent with Prioritized Replay

Description

To improve the convergence of the algorithm a prioritized replay algorithm was added to the base DQN agent. When the agent samples from the experiences, it does so through a probability distribution proportional to the the normalized TD error that experiences had when added to the Replay Buffer.

Probability of an experiences is calculated as follows:

$$P(i) = \frac{p(i)^a}{\sum_i^{N_{sample}} p(i)^a}$$

Where,

$p(i)$, is the priority of each experience

N_{sample}, is the available number of experiences

a, a hyperparameter that sets creates a balance between using randomness and priority

The priority is the absolute value of the TD error further adjusted with a small constant e , which makes sure that even experiences with zero error get sampled.

The algorithm also adjusts for the non-random sampling with a Importance Sample element. Each loss is multiplied by the corresponding importance sample. A constant Beta is used when calculating the importance samples.

Results

Unfortunately, Prioritized Replay both underperforms in terms of converging to a solution and computational efficiency. Everytime the Replay Buffer samples according to priorities it has to go through each sample and pick a subset according to their probabilities. This can be costly as the size of Replay Buffer growth with each episode. One can set a hard upper limit of the size of the buffer (in this case it is 100.000 experiences), but which still results in a large computational expensive step.

Future Ideas

The DQN Agent with Prioritized Replay could front load some of the calculations, eg.: keeping track of the sum of priorities in a separate variable, which is incremented as the experience get added. The hyperparameters of the Prioritized Replay could also be further tweaked, eg.: in this implementation Beta is constant, but could increase with the passing of episodes. Other improvement could be implemented from the Rainbow improvements.

References

Grokking, Deep Reinforcement Learning, Miguel Morales, 2020

Dataset

Entire Project Folder with code can be found here (folder): deep-reinforcement-learning/p1_navigation at master · szemyd/deep-reinforcement-learning (github.com)