Navigation Project

Tiago Montalvão

June 17, 2020

1 Introduction

This is a report of the first project of Udacity Deep Reinforcement Learning Nanodegree. It describes the model architecture with chosen hyperparameters used in the agent, the experiments, and potential improvements upon the current results.

2 Implementation

2.1 Agent

The agent was implemented using a Deep Q-Network. To read the paper describing the full details of DQN and from where the following algorithm was taken, see [1].

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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Basically, the DQN consists of a combination of Q-Learning (Sarsamax), which is an off-policy model-free reinforcement learning algorithm, with neural networks, used for approximating the states, since the state space can be huge or sometimes infinite, as it is in this case, since the state gathers some continuous information (e.g. the agent's velocity)

The DQN was then implemented with:

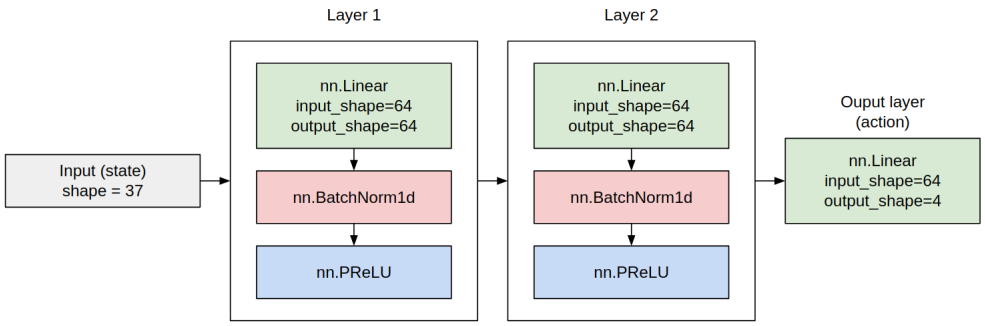
- **Experience replay buffer** to handle data correlation between consecutive samples.
- **Two neural networks** with identical architecture (local and target networks), as described in the following section, with **soft update** to the target network.
- ε**-greedy exploration** strategy, with ε starting at 1.0 and exponentally decreasing up to 0.1.

The chosen hyperparametes are described above:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 1.00            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

2.2 Model architecture

The model architecture is shown in the figure below:



There are two hidden layers, each consisting of:

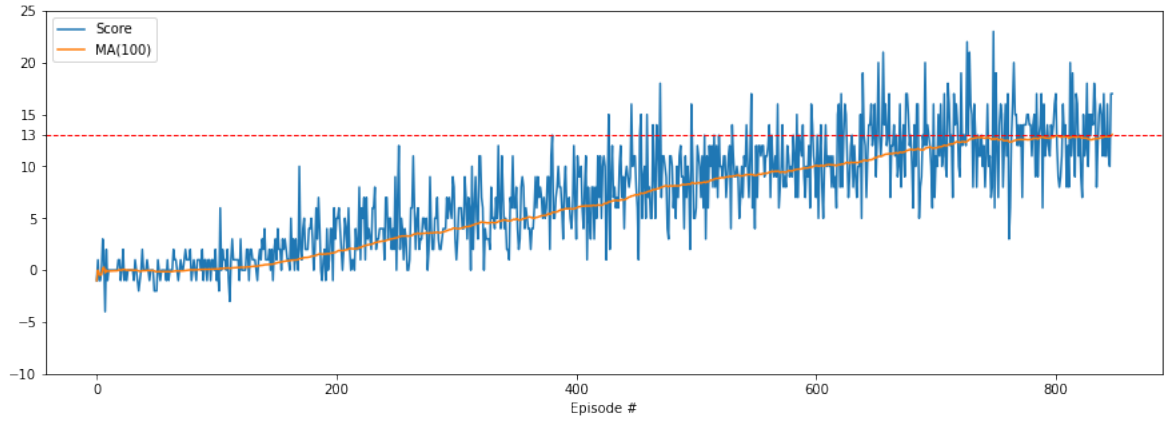
1. Linear activation, with 64 neurons
2. 1d Batch Normalization
3. Parametric ReLU [2]

In the project notebook, there is a cell which executes a command that gives the model summary (the same info above with some numbers). The most interesting here is the number of trainable parameters: 7110, which is not large.

3 Results

The agent was able to solve the environment in only 848 episodes, and it took about 32 minutes to train. All the details can be checked in the project notebook.

The following image shows both individual episode score (in blue) as well as the moving average of the last 100 episodes (or all of the episodes, if the episode number is less than 100).



4 Ideas for Future Work

DQN is one of the most classic value-based algorithms for DRL, but a couple of improvements can be implemented. Some of them are:

- **Double DQN** [3]: tackles the problem of overestimating Q-values.
- **Prioritized Experience Replay** [4]: gives more weight in experience sampling from replay buffer for tuples that have larger TD errors.
- **Dueling DQN** [5]: learns a state-value function, along with a function called advantage for each action in a particular state.

On top of these improvements and a few more, a new algorithm was developed, called Rainbow [6]. In the time this algorithm was developed, it was state-of-the-art.

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, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[3] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

[6] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.