Reinforcement learning with DQN

Rotaru Liviu-Iulian

Faculty of Mathematics and Computer science, University of Bucharest

Introduction

Reinforcement learning is an area of machine learning concerned with the study of agents interacting with an environment in order to maximize a reward signal. The introduction of deep neural networks caused by the advancement of technology in the last decade has brought us Deep RL which meant the policy and the q-functions previously used were to be replaced by neural networks reeking their benefits: advanced feature extraction and generalization. Lately this field of work has seen the attention of more and more giants in the field such as OpenAI who have created dedicated testbeds for this specific research area.

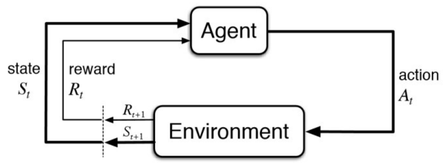

Fig. 1:

The Method

Let **E** be an environment with **S** a set of discrete states and A a set of discrete actions over the said sates. Q-learning is the RL method that uses a q-function **Q(s,a,r,s')** to approximate the value of a given state **V(s)** where **s** is the last state, **a** is the action taken in that state, **r** is the reward given and **s'** is the state that we ended up in. In order to choose the action we may use a greedy or eps-greedy policy.

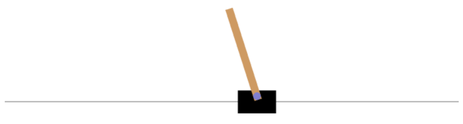

Fig. 2:

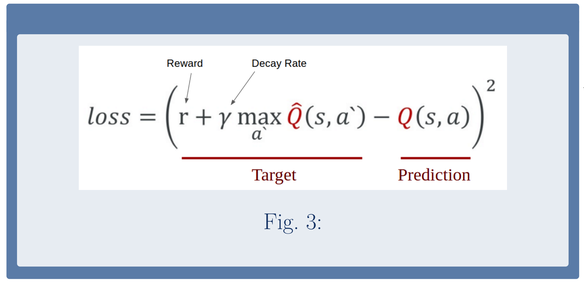

Fig. 3:

In deep RL the Q-function will be a Neural network that maps the observation of the environment to the Values of the next states we could end-up in. In practice, training deep RL agents can take a lot of time and is very hard to refine. For this work I chose to implement DQN[1] as it is one of the most popular method in the literature and has seen a lot of improvements over the years [2]. The DQN algorithm comes with 2 notable improvements: 1. It splits the Q network into 2 separate networks, the first being the on-line network that is actively improved when optimizing and another that is used as a target network that is updated every n steps from the on line network. 2. Experience replay which is a buffer that contains **(s, a, r, s')** tuples, that samples from the last C experiences that the agent had.

Result and discussion

I will be using openaigym's Cartpole environment to test the DQN algorithm and illustrate the training process and results. There are 2 ways to get the observations from the environment: the exact state observations regarding speed, position etc. and taking the exact environment render information, the full image. I will be discussing both of them here. First of all taking the observations of the given environment gives the network a great advantage as there are only a few numbers to work with as opposed to taking a full image. For the observations I used a network of 2 fully connected layers with a batch normalization between them.

For the images I used a 3 layer convolutional network with a fully connected head at the end (Figure 4).

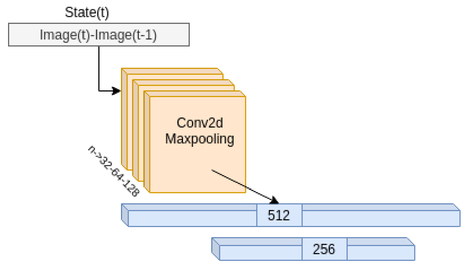

Fig. 4:

The training process and what i've learned

orange is the validation value over 50 episodes
blue is the reward for every episode
green is the mean of the last 100 training rewards

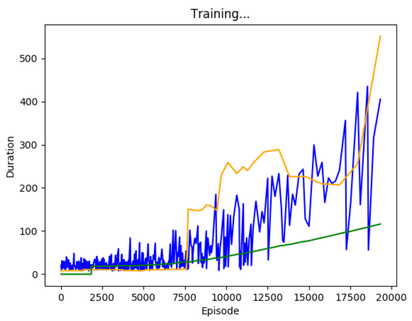

Fig. 5:

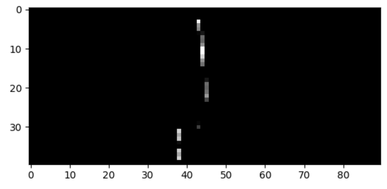

Fig. 6:The RGB or Greyscale subtraction of 2 consecutive frames

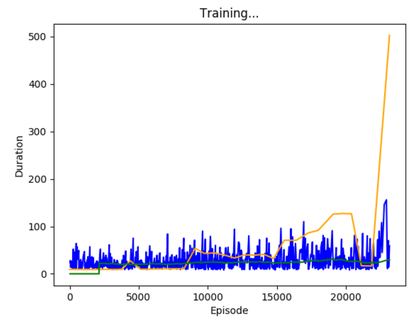

Fig. 7:

It is no secret that training reinforcement learning agents and especially when using DQN is really unstable and susceptible to hyperparameters changes. This has also been the case for this project. Figure 5 illustrates a successful training process for the low dimensional observations case. The validation, average reward and reward values all have a clear upward trajectory. However this is not always the case, as we can see in Figure 7. The sudden uptick in the validation value can be the result of the combination of a lot of different factors such as the Q function receiving a sequence of good batches, the learning rate being a bit too big. I will outline a few of the conclusions I have drawn: Having a very big replay memory doesn't mean better performance. Tweaking the replay memory is very important but overestimating how much you need will lead to overfitting on strategies that your agent already learned. It takes a lot of time: Training an agent can take a lot of time depending on the batch size used. Changes in the learning rate may seem not to bring anything to the training process initially but could mean a big deference in the final result.

References

[1] Human-level control through deep reinforcement learning.

[2] Rainbow: Combining improvements in deep reinforcement learning.