Deep Q-Learning for Rubik's Cube

Etienne Simon
ENS Cachan
esimon@esimon.eu

Eloi Zablocki
ENS Cachan
eloi.zablocki@gmail.com

Abstract

Deep Q-Learning is an algorithm which involves Q-Learning and Deep Learning. It has proved to be extremely succesfull for some applications [4]. The Rubik's Cube is a solved game as there exists an algorithm that solves any shuffled cube in the minimum number of moves, however we have applied a Deep Q-Learning algorithm to try to solve the Rubik's cube game. Because the reward is positive only when the cube is solved ans that the exploration grows exponentially, we have considered some tweaks for the algorithm such as curriculum learning. Mixed results have been obtain because of the slow convergence of the Q-learning algorithm.

1 Introduction

From a Reinforcement Learning (RL) perspective, the Rubik's Cube game is hard because whatever the action taken, the reward will always be zero, unless the cube is finished (same color on same face) which is a very improbable event if random moves are being taken from a suffled cube. Such situations are almost impossible to solve for classical Q-learning algorithms.

Here we consider a Deep Q-Learning algorithm that is a variant of the Q-Learning algorithm that uses a Feedforward Neural Network at the action-value function to be learned. Our hope is that the network learns patterns in the colors and the action in order to make possible the learning.

Moreover, we want to help the learning by showing easy examples first (cubes almost finished) and the increase the complexity of the examples until we reach the general problem of a completely randomly shuffled cube. Such an approach is knows as Curriculum Learning [1].

2 Deep Q-learning algorithm

In this section, we present the algorithm we have used and some theoretical considered aspects.

2.1 Q-Learning

Q-learning is a model-free reinforcement learning technique. It works by learning an action-value function that eventually gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. In our case, the action-value function is a mapping between the combination of the state and the action taken, and the utility value, where the utility value is the sum of the present reward and the discounted future rewards. When the mapping is learned, the optimal policy is to simply select the action that has the highest value in each state.

2.2 Deep Learning

The main specificity about Deep Q-Learning is that the action-value function is a feedforward neural network. The input of the network is a vector representing the state and the action taken and the

output is the utility value. Feedforward neural networks have several advantages. First, they can approximate arbitrarily well any continuous function thanks to the universal approximation theorem [2]. Moreover, the training on the parameters of feedforward neural networks has been made easy thanks to the backpropagation algorithm [5].

2.3 The Deep Q-Learning algorithm in [4]

2.3.1 Notations

For practical reasons, most of the notations used here are the same as in [4]. We note $\mathcal{D}$ the replay memory and N its capacity. The action-value mapping is noted Q. x_t denotes the state at time t, a_t the action taken at time t, r_t the observed reward and x_{t+1} the resulting state. The parametrization of the deep learning model (the feedforward neural network) is represented by θ.

2.3.2 A specificity : the Replay Memory

When interacting with the environment, the algorithm stores the quadruplet (x_t, a_t, r_t, x_{t+1}) in what is called a Replay Memory. Its purpose is to smooth the training over many past events and behaviors and not just what has just happened. This idea was first presented in [3]. At every step of the algorithm, when training in involved, a minibatch of quadruplet is taken from the replay memory. The gradient is computed on this minibatch. Thus, we see that every episode that is contained in the replay memory can be used many times for the training. Moreover, the correlations between close actions and states do not bias the algorithm anymore since the minibatch is taken from random elements of the replay memory ; our hope is that the variance of the updates will be lower with such a procedure. A small variance means that the algorithm avoid to fall in poor local minima and to oscillates between pseudo-stable states.

2.3.3 Algorithm

The algorithm presented here is the same as the one in [4]

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise a shuffled cube x_1
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(x_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and new state x_{t+1}
    Store transition (x_t, a_t, r_t, x_{t+1}) in the replay memory $\mathcal{D}$
    Sample random minibatch of transitions (x_j, a_j, r_j, x_{j+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(x_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(x_j, a_j; \theta))^2$ [1].
  end for
end for

2.4 Curriculum Learning

The basic idea to start learning easy things and to move on harder things is the core of what is called Curriculum Learning. Formalised in [1], the article shows that some models can learn up to two times faster if the learning strategy follows some curriculum that show easy examples first, in comparison with a strategy that does not take care of the order of the examples.

[1] The gradient is automaticaly computed with Theano

3 Experiments

The code is written in *Python*. We have use the library *Theano* which allows automatic differenciation and symbolic optimization. We have also used the convenient library *Blocks* which is a Theano framework. One main feature about Theano is the fact that it can be run on Graphical Processing Units (GPU) in order to speed up the training. However, in our case we only used CPU computing because what takes the most time is not the computation of the gradient but rather the action that gives the highest utility value at a given state.

3.1 Practical considerations

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[2] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24, 2001.

[3] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.