Intelligent Autonomous Game Bots using Deep Q Learning

Karan Saxena, Amit Asish Bhadra, Adarsh Trivedi, Aditya Roy Choudhary
Guide: Prof. Dilip K Sen [Batch 06 ; D: 23rd June, 2017]
{1MV13CS047, 1MV13CS014, 1MV13CS004, 1MV13CS005, dilipksen} @sirmvit.edu

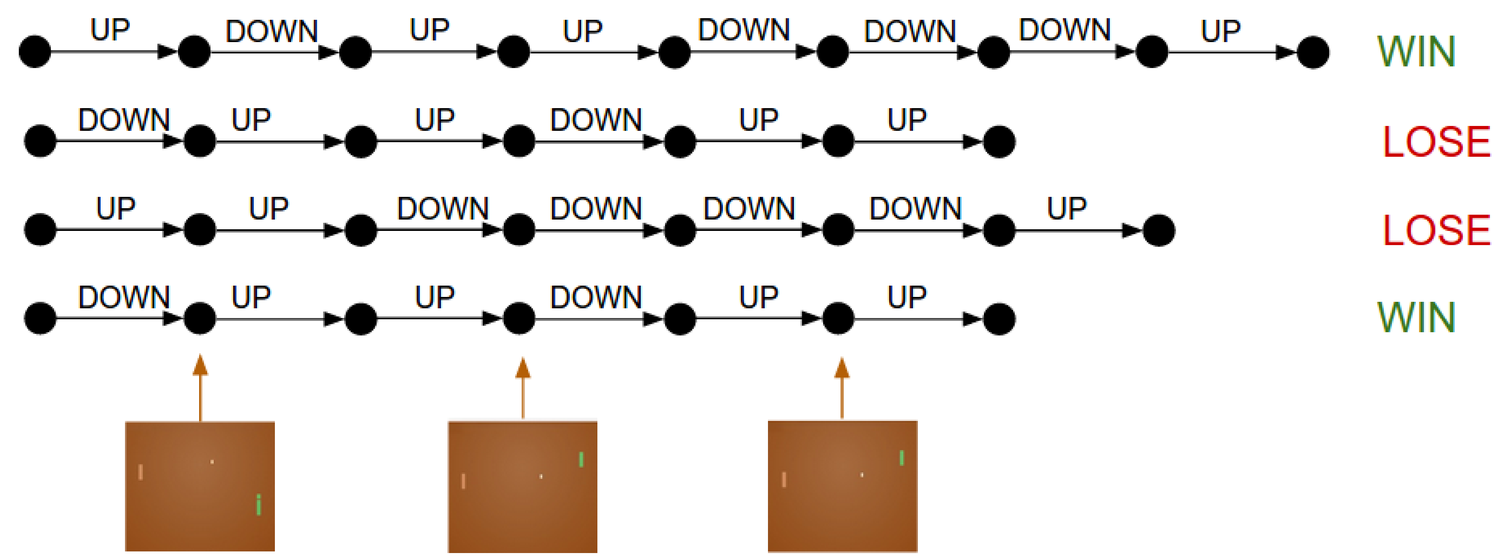
Problem

Unlike policy gradient methods, which attempt to learn functions which directly map an observation to an action, Q-Learning attempts to learn the value of being in a given state, and taking a specific action there.

This is a difficult problem due to four aspects.

1. Exploration vs Exploitation
2. Stochasticity of taking Actions
3. Conditional Rewards
4. Delayed Rewards

Representation



Contributions

We try to mitigate the problem of exploration vs exploitation, and solve it efficiently with a modification of Deep Q Network algorithm.

The method is based on [1]. Our main modifications are:

1. Using OpenAI Gym to get the environment.
2. Using Gym helped making the code environment agnostic.
3. Parallelizing DQN.

Method

In it's simplest implementation, Q-Learning is a table of values for every state (row) and action (column) possible in the environment.

$$Eq\ 1.\ Q(s,a) = r + \gamma(max(Q(s',a'))$$

This says that the Q-value for a given state (s) and action (a) should represent the current reward (r) plus the maximum discounted (γ) future reward expected according to our own table for the next state (s') we would end up in.

We are going to be using a method called policy gradients, where our simple neural network learns a policy for picking actions by adjusting it's weights through gradient descent using feedback from the environment.

References

1. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

2. Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.

3. Crites, Robert H., and Andrew G. Barto. "An Actor/Critic Algorithm that is Equivalent to Q-Learning." Advances in Neural Information Processing Systems (1995): 401-410.

Policy Gradient

To update our network, we will simply try an arm with an e-greedy policy. This means that most of the time our agent will choose the action that corresponds to the largest expected value, but occasionally, with e probability, it will choose randomly. In this way, the agent can try out each of the different arms to continue to learn more about them. Once our agent has taken an action, it then receives a reward of either 1 or -1. With this reward, we can then make an update to our network using the policy loss equation:

```
Loss = Log(π)*A
```

A is advantage, and is an essential aspect of all reinforcement learning algorithms. Intuitively it corresponds to how much better an action was than some baseline. In future algorithms, we will develop more complex baselines to compare our rewards to, but for now we will assume that the baseline is 0, and it can be thought of as simply the reward we received for each action.

π is the policy. In this case, it corresponds to the chosen action's weight.

A Typical RL Problem

Typical aspects of a task that make it an RL problem are the following:

1) Different actions yield different rewards. For example, when looking for treasure in a maze, going left may lead to the treasure, whereas going right may lead to a pit of snakes.

2) Rewards are delayed over time. This just means that even if going left in the above example is the right things to do, we may not know it till later in the maze.

3) Reward for an action is conditional on the state of the environment. Continuing the maze example, going left may be ideal at a certain fork in the path, but not at others.

All we need to focus on is learning which rewards we get for each of the possible actions, and ensuring we chose the optimal ones. In the context of RL lingo, this is called learning a policy.

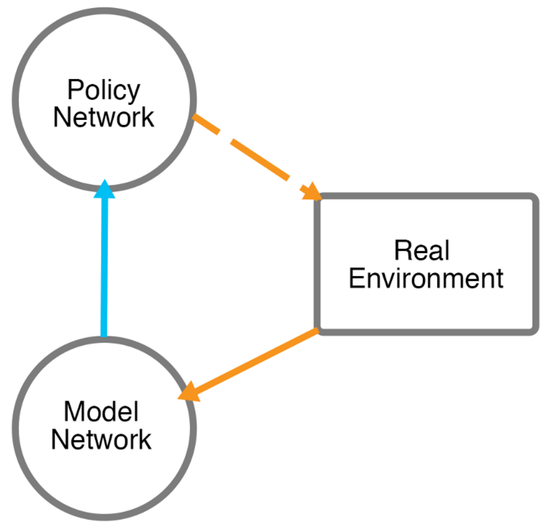
Markov Decision Process

We can define a Markov Decision Process as follows.

An MDP consists of a set of all possible states S from which our agent at any time will experience s. A set of all possible actions A from which our agent at any time will take action a. Given a state action pair (s, a), the transition probability to a new state s' is defined by $T(s, a)$, and the reward r is given by $R(s, a)$. As such, at any time in an MDP, an agent is given a state s, takes action a, and receives new state s' and reward r.

While it may seem relatively simple, we can pose almost any task we could think of as an MDP. For example, imagine opening a door. The state is the vision of the door that we have, as well as the position of our body and door in the world. The actions are our every movement our body could make. The reward in this case is the door successfully opening. Certain actions, like walking toward the door are essential to solving the problem, but aren't themselves reward-giving, since only actually opening the door will provide the reward.

Model Based RL



A model is going to be a neural network that attempts to learn the dynamics of the real environment.

For example, in Pong we would like a model to be able to predict the next position of the bat given the previous position and an action. By learning an accurate model, we can train our agent using the model rather than requiring to use the real environment every time.