Machine Learning Engineer Nanodegree

Capstone Proposal

Sachin Gupta

April 9, 2018

PROPOSAL

Domain Background

Table based Q-Learning (note: Not Deep Q-Learning) is a type of reinforcement machine learning. Q-Learning was first detailed in a Cambridge PhD thesis by Christopher Watkins in 1989. It is inspired by how you would normally train an animal or child. You positively reward desirable behavior and punish (negative reward) undesirable behavior.

The next problem is in the rate at which out Q-Table grows with complexity.

This is what prompted the use of an artificial neural network to approximate the behavior of a Q-Table. In 2013 DeepMind released 'Playing Atari With Deep Reinforcement Learning' and 2015 'Human-Level Control Through Deep Reinforcement Learning'.

Proposal Statement

My project proposes a comparative study of the performance of table-based Q-Learning and Deep D Learning algorithms on various simulations available in 'Gym' (Atari games, Box2d, etc.).

The agent will be coded using both the (Q-table based) Q-Learning Algorithm and the Deep Q Learning algorithm and the performance will be compared in terms of training time and also the ability to handle non precedented states.

Datasets and Inputs

The environment and simulator have been taken from the ‘Gym’ library.

The environment’s **‘step’** function returns four values. These are:

1. Observation (object): an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.
2. Reward (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
3. Done (boolean): whether it’s time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes and ‘done’ being ‘True’ indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)
4. Info (dict): diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment’s last state change). However, official evaluations of your agent are not allowed to use this for learning.

Solution Statement

This ideology of this project is to tackle the problem of the rate of growth of the Q-table which has to contain policies for all possible states hence, several models based on deep-reinforcement learning algorithms will be trained and compared.

Benchmark Model

Since the main objective of this project is to improve the performance if the Table based Q-learning model. It will involve a comparative study of various and models and hence the performance will be evaluated relative to the original q-learning agent and the new deep-reinforcement learning models.

Evaluation Metrics

Along with this some important factors, under consideration will be the training time and training efficiency in terms of arriving at an optimal policy for maximum number of states possible and the ability to handle non precedented states.

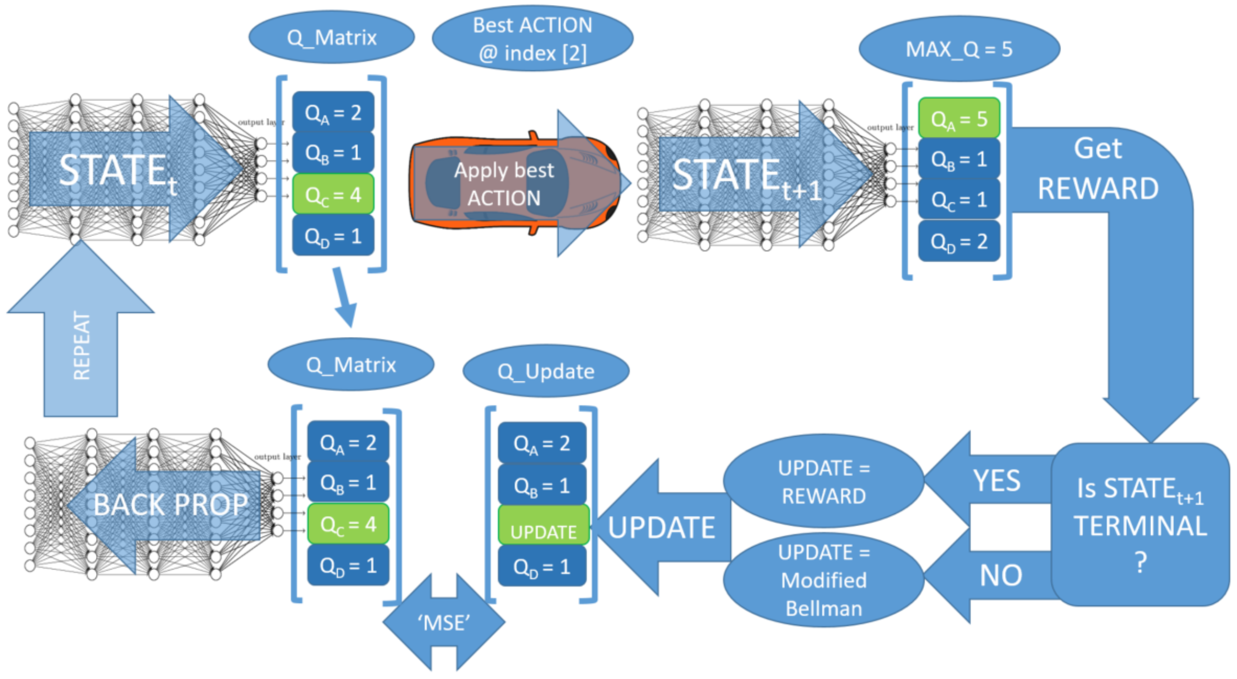
Also, there is a score attached to each episode that depicts the maximum cumulative reward earned by the agent and the slope and shape of the curve formed by plotting the various scores will also be used to make appropriate inferences.

Project Design

The project involves:

1. An environment in which the agent operates.
2. A simulator that simulates that environment.
3. The agent originally trained on Table based Q-learning algorithm.
4. Agents trained on one or more deep-reinforcement learning methods.

The flowchart for a deep-reinforcement learning (here, deep q-learning) agent:



Algorithm:

```
initialize state, epsilon, alpha, epsilon
while training:
  observe state
  feed forward state through NN to get q_matrix
  if (epsilon > random_float_between_0_and_1):
    select random action
  else:
    action_idx = index of the element of q_matrix with largest value
  apply action at action
  observe next_state
  observe reward
  if next_state is terminal:
    q_update = reward
  else:
    feed forward next_state through NN to get q_matrix_next
    observe the largest q value (q_max) in q_matrix_next
    q_update = reward + gamma*q_max

  make copy of q_matrix (q_target)
  q_target[action_idx] = q_update
  loss = MSE(q_matrix, q_update)
  optimize NN using loss funciton
  state = next_state
  epsilon -= decay_rate
```