Iowa State University

EE 526X

Final Project: Double Q-Learning

Eliska Kloberdanz

December 11, 2019

This project is to design a reinforcement learning algorithm using double Q-learning algorithm with function approximation. Two multi-layer neural networks should be designed as the Q function approximators. The environment should be Acrobot-v1, taken from the OpenAI Gym package.

The attached python script implements the the Double Q-learning algorithm from Hasselt et al (2015) using two deep neural nets.

---

i. Problem Definition

i. Background: Reinforcement Learning & Double Q-learning

Reinforcement learning is is a type of machine learning that involves mapping situations to actions with the goal of maximizing reward. Therefore, the goal of reinforcement learning is to learn policies to guide actions in sequential decision problems by maximizing the future cumulative reward signal (Sutton and Barto 1998). Q-learning (Watkins 1989) is a popular model-free temporal difference algorithm used for evaluating actions in sequential decision making to approximate q*, the optimal action-value function. The main idea of the Q-learning algorithm is to estimate Q-value (the total reward) through iterative updates. The Q-value is defined as a sum of the immediate and maximum discounted future rewards as:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

As the above equation shows, Q-value depends on s = state, a = action, r = reward, γ = discount factor, and s' = future state. The future state s' that follows s in turn depends on s'' and so on. Therefore, the Q-value is defined recursively as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

The full Q-learning algorithm is shown below:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Source: Sutton and Andrew Barto (2014)

It has been shown that the Q-learning algorithm converges to q* with a 100% probability. However, "is known to sometimes learn unrealistically high action values because it includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values" Hasselt (2010). In response to addressing this maximization bias, Hasselt (2010) developed an improved double Q-learning algorithm that results in more reliable and stable learning.

Double q-learning involves two independent Q-value estimators that are used to update each other resulting in an unbiased Q-value estimate. In particular, we divide Q into $Q_1(a)$ and $Q_2(a)$ representing two independent estimates for the true value q(a), which allows us to use one estimate to choose the maximizing action and a different estimate to approximate the value of that maximizing action. Below is the full double Q-learning algorithm:

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Source: Sutton and Andrew Barto (2014)

Double Q-Learning uses two Q-value functions – Q1 and Q2 – which allows for separation of value estimation and action selection. Both Q1 and Q2 are approximations of the same underlying Q-function (Q*(A)), because the expected values of Q1 and Q2 converge to Q*.

This algorithm has been further improved by Hasselt et al (2015). In this project I implemented this updated version of the double Q-learning algorithm. Please see implementation details and algorithm pseudo-code below.

ii. Description of Neural Networks

The implemented double Q-learning algorithm utilizes two neural networks – primary and target neural network. The primary network is trained using back-propagation, whereas the target network is not. Instead, the target network is updated every x number of steps by setting its parameters to the model parameters of the primary network that are learned through back-propagation. The primary and target neural nets have the same architectures – they are both deep fully connected neural nets with 64 neurons in the first layer, 20 neurons in the second layer and 3 neurons in the last layer – the 3 neurons in the last layer represent the three possible actions. The input dimension on the neural nets is a 1-d array of six numbers and the non-linear activation function is relu. The last layer uses a linear action function.

iii. Algorithm Description and Pseudo-code

Double Q-learning Algorithm Pseudo-code

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

The above algorithm that I implemented is based on Hasselt et al. (2015), which is an updated version of the double Q-learning algorithm developed by the same author as the original one published in Hasselt et al. (2010). The updated algorithm in Hasselt et al. (2015) utilizes two models: Q (primary model), and Q' (target model) as opposed to two independent models. Model Q' is used for selecting actions and model Q is used for action evaluation. Model Q learns through mean squared error minimization between Q and Q*, where Q* is the optimum. The parameters that Q learns are periodically copied to Q' through Polyak averaging:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' ,$$

where θ' is the target network parameter, θ is the primary network parameter, and τ (rate of averaging) is 0.01.

The deep Q networks learn to approximate Q value functions through the Q-learning update equation below.

$$Q(s, a) = r(s, a) + \gamma \max_a Q(s', a)$$

The separate target network discussed above is another important component of training deep Q networks. Because the target network is reset only every x steps, this allows more stable learning. Finally, the algorithm implemented also utilizes experience replay. The algorithm has "memory" - it stores previous transitions, which consist of (state, action, reward, next state). The algorithm

randomly samples data from this memory to create a minibatch used for training. This technique mitigates high correlation of training samples and improves the time required for the algorithm to converge.

iv. Environment

The reinforcement learning algorithm is Acrobot-v1, taken from the OpenAI Gym package. “The acrobot system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height.” (gym.openai.com)

The input observation consist of 6 numbers, which represent the sine and cosine angles and angular velocity of each join of the two link pendulum. There are three possible discrete actions: 1, 0, -1, which represent the torque applied. Reward is −1 for every step taken, except the termination step. Please see further details about the Acrobot-v1 environment in the table below.

Goal	The goal is to swing the end of the lower link up to a given height.
Action	The action space is discrete, namely, the system is controlled by applying a torque of 0, +1 and -1 to the links.
Observation	The observation is a vector with six elements, for example, [0.9926474, 0.12104186, 0.99736744, -0.07251337, 0.47965018, -0.31494488], which describe the positions of the two links.
Reward	A reward of +1 is provided for every timestep where the lower link is at the given height or, otherwise, -1.
Termination	The episode ends when the end of the lower link is at the given height, or the maximum number of steps is reached.

Source: Ravichandiran et. al (2019)

v. Results

i. Performance

Please see below and example of 10 episodes and the reward earned per episode after successfully accomplishing the goal.

i. Cumulative reward per episode: [-500.0, -304.0, -150.0, -142.0, -192.0, -354.0, -143.0, -174.0, -500.0, -185.0]

ii. Average cumulative reward: -264.4

ii. Training Time

i. Training time per episode in seconds: [47.34665012359619, 35.378451347351074, 17.102014541625977, 16.506609201431274, 21.656212091445923, 39.10354518890381, 15.891824722290039, 19.09297275543213, 55.00230050086975, 20.35388970375061]

ii. Average training time: 28.743447017669677 seconds

vi. Conclusion

i. Insights

What I have learned from this project is that reinforcement learning is quite a different paradigm than supervised machine learning, which is the type of machine learning I was previously the most familiar with. Reinforcement learning is about trial and error and in my opinion, the exploration parameter (epsilon) is one of the most important parameters if the double Q-learning algorithm.

ii. Proposed Improvements

Acrobot-v1 is an unsolved environment, which means it does not have a specified reward threshold at which it's considered solved. The 13 best results listed on https://github.com/openai/gym/wiki/Leaderboard range from -42 to -500 reward per 100-episode best performance. The reward that I achieved on average was around -260, which is a good result in comparison with the statistics published on Leaderboard. Nevertheless, the results could be improved by training the primary neural net longer and further tuning the parameters of the neural net and the double Q-learning algorithm.