Machine Learning Engineer Nanodegree

Capstone Proposal

Abdul Mdee
June 30, 2018

Proposal: Application of Reinforcement Learning Methods.

Domain Background

Reinforcement learning is a branch of machine learning whereby an agent learns by interacting with the environment, see figure 1. It is inspired by behaviorist psychology. Reinforcement learning is very different from other types of machine learning (supervised learning and unsupervised learning) because, in a reinforcement learning, the learning occurs through interactive experience while the other types of machine learning are through learning from the dataset.

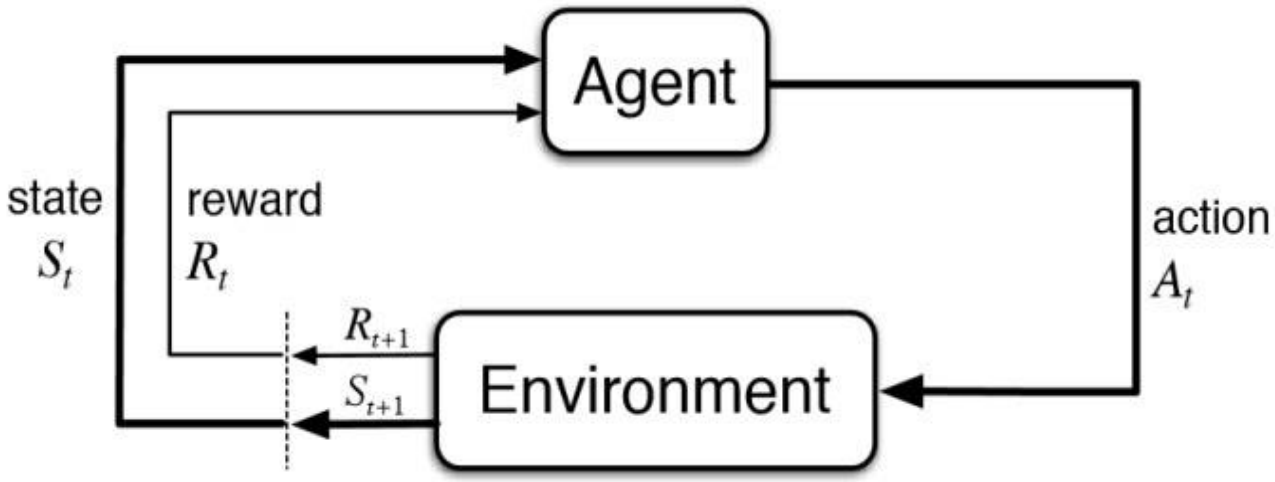


Figure 1

Reinforce learning is very important because it is the closest science has come to build a machine with almost human intelligent. I am attracted to reinforcement learning because of the possibility of understanding how human intelligent works. Also using the reinforcement learning techniques to solve and improve classical problems such as teaching a robot to walk. The field of reinforcement learn shows so much promise in improving / building an actual artificial

intelligent (AI). As we can see example from projects like Deepmind; where AlphaGo used deep reinforcement learning to defeat a Go world champion, and arguably the strongest Go player in history.

Problem Statement

(approx. 1 paragraph)

In order to actually understand the field of reinforcement learning and it applications, we have to understand how reinforcement learn works in a simple problems. I am going to apply reinforcement learning methods to a simple card game called Easy21. Easy21 is similar to the Blackjack but the rules are different and non-standard. The project problem statement is inspired by David Silver Class at UCL

Datasets and Inputs

It is a reinforcement learning approach no specific dataset needed but the following are the rules of the game.

1. An infinite deck of cards (i.e. cards are sampled with replacement).
2. Each draw from the deck results in a value between 1 and 10 (uniformly distributed) with a color of red (probability 1/3) or black (probability 2/3).
3. There are no aces or picture (face) cards in this game.
4. At the start of the game both the player and the dealer draw one black card (fully observed).
5. Each turn the player may either stick or hit.
6. If the player hits then she draws another card from the deck.
7. If the player sticks she receives no further cards.
8. The values of the player's cards are added (black cards) or subtracted (red cards).
9. If the player's sum exceeds 21, or becomes less than 1, then she "goes bust" and loses the game (reward -1).
10. If the player sticks then the dealer starts taking turns. The dealer always sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome – win (reward +1), lose (reward -1), or draw (reward 0) – is the player with the largest sum.

Solution Statement

Problem solution approach:

- Creating an environment that implement Easy21. The environment contains a function named step, which takes state s (dealer's first card 1–10 and the player's sum 1–21) and an action a (hit or stick) as an input and returns a sample of the next state s' (which can be a terminal state if the game is finished)

- The environment will be a model-free reinforcement learning and the transition matrix will not be explicitly represented to the MDP.
- No discounting ($\gamma = 1$), the dealer's moves are treated as a part of the environment , i.e. calling *step* with a *stick* action will play out the dealer's cards and return the final reward and terminal state.

Benchmark Model

I propose to use maximization of the return by the reinforcement learning agent as the benchmark for Easy21 through Monte-Carlo control and TD learning.

Evaluation Metrics

I will use an optimal value function as an evaluation metric.

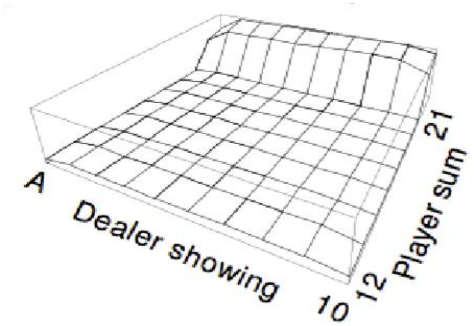
$$V^*(s) = \max_a Q^*(s,a)$$

Project Design

I will apply the following methods to game of Easy21 per recommendation from David Silver class. Please note the project design is per Easy21 approach in David Silver class.

1 Monte-Carlo Control in Easy21:

Apply Monte-Carlo control to Easy21. Initialize the value function to zero. Use a time-varying scalar step-size of $\alpha_t = 1/N(s_t,a_t)$ and an ε-greedy exploration strategy with $\epsilon_t = N_0/(N_0 + N(s_t))$, where $N_0 = 100$ is a constant, $N(s)$ is the number of times that state s has been visited, and $N(s,a)$ is the number of times that action a has been selected from state s. Feel free to choose an alternative value for N_0, if it helps producing better results. Plot the optimal value function $V^*(s) = \max_a Q^*(s,a)$ using similar axes to the following figure taken from Sutton and Barto's Blackjack example.



2 TD Learning in Easy21:

Implement Sarsa(λ) in 21s. Initialize the value function to zero. Use the same step-size and exploration schedules as in the previous section. Run the algorithm with parameter values $\lambda \in \{0, 0.1, 0.2, \ldots, 1\}$. Stop each run after 1000 episodes and report the mean-squared error $\sum_{s,a} (Q(s,a) - Q*(s,a))^2$ over all states s and actions a, comparing the true values $Q^*(s,a)$ computed in the previous section with the estimated values $Q(s,a)$ computed by Sarsa. Plot the meansquared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

3 Linear Function Approximation in Easy21:

We now consider a simple value function approximator using coarse coding. Use a binary feature vector $\phi(s, a)$ with $3 * 6 * 2 = 36$ features. Each binary feature has a value of 1 iff (s,a) lies within the cuboid of state-space corresponding to that feature, and the action corresponding to that feature. The cuboids have the following overlapping intervals:

$dealer(s) = \{[1,4], [4,7], [7,10]\}$

$player(s) = \{[1,6] [4,9], [7,12], [10,15], [13,18], [16,21]\}$

$a = \{hit, stick\}$

Where:

- *dealer*(*s*) is the value of the dealer's first card (1–10)
- *sum*(*s*) is the sum of the player's cards (1–21)

Repeat the Sarsa (λ) experiment from the previous section, but using linear value function approximation $Q(s,a) = \phi(s,a)^T \theta$. Use a constant exploration $\varepsilon = 0.05$ and a constant step-size of 0.01. Plot the mean-squared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

References:

1. http://www.incompleteideas.net/book/ebook/
2. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html