Notes

June 27, 2019

A collection of notes derived from the book, *Reinforcement Learning* by Richard Sutton and Andrew Barto. Available at `http://incompleteideas.net/book/the-book.html`

1 Action selection

Most RL methods require some form of policy or action-value based action selection method.

- **Greedy Selection**: Choosing the best action.

$$A = \text{argmax}_a Q(a)$$

- **ε-greedy Selection**: Simple exploration with ε-probability.

$$A \leftarrow \begin{cases} \text{argmax}_a Q(a) & \text{with probability 1 - } \varepsilon \text{ (breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

- **Upper Confidence Bound (UCB)**: Takes into account the proximity of the estimate to being maximal and the uncertanty in the estimates. Does not perform well on large state spaces.

$$A_t = \underset{a}{\text{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

  Where:

  - $c > 0$ is the degree of exploration
  - $N_t(a)$ is the number of times that action a has been selected prior to time t. If $N_t(a) = 0$, then a is considered to be a maximizing action.

2 Performance Measures

Methods for comparing the performance of different parameters and algorithms.

- **Optimal Action %**: Requires knowledge of the workings of the environment and whether the action was optimal. Plot % over steps.

- **Average reward**: Simply plot the average reward over steps. Good for comparing specific implementation of agent in specific implementation of environment.

- **Average reward w.r.t. parameter**: Plot the average reward over first n=1000 steps against input parameter(s) (ε, α, c, Q_0 etc) on a logarithmic scale. Good for comparing learning algorithms' general effectiveness and finding the best parameter value.

- **Mean Square Error**: Plot the mean square error (averaged over n=100 runs) of the value of a single state (error = actual-estimate) over the number of episodes run before acheiving the estimate, with the episodes on a logarithmic scale. Good for Monte Carlo method, where you can form an estimate of a single state without forming an estimate of the others.

3 Algorithms

Reinforcement learning algorithms covered by the book.

- **Dynamic Programming / Value Iteration**: Updating state values by sweeping through all states. Computationally expensive, especially on large state spaces.

```
Parameters:
  a small threshold $\theta$ > 0
  Initialize $V(s)\ \forall s \in S^+$ arbitrarily, except that
  $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in S^+$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy $\pi$, such that
  $\pi(s) = \mathrm{argmax}_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma V(s')]$
```

- **Off-policy Monte Carlo control**: Using a soft policy to explore, while estimating the optimal policy. Does not bootstrap, i.e. doesn't use estimates of previous states to estimate the current state value.

```
Parameters:
  For all $s \in S, a \in A(s)$:
    Initialize $Q(s,a)$ arbitrarily
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \mathrm{argmax}_a Q(s,a)$ (with ties broken consistently)
Loop for each episode:
```

```
    $b \leftarrow$ any soft policy
    Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
       $G \leftarrow \gamma G + R_{t+1}$
       $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
       $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
       $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken consistently)
       If $A_t \neq \pi(S_t)$, exit inner Loop (proceed to next episode)
       $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

- **n-step Sarsa**: Using the rewards from the previous n steps to update the value of the current state. Uses an ε-greedy policy to estimate Q Q*.

```
Initialize $Q(s, a)$ arbitrarily, for all $s \in S, a \in A$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$,
or to a fixed given policy.
Algorithm parameters:
  stepsize $\alpha \in$ (0, 1],
  small $\varepsilon$ > 0,
  a positive integer $n$
All store and access operations (For $S_t, A_t$, and $R_t$ can take their
index mod $n$+1
```