Reinforcement Learning

Arnav Gupta

November 27, 2024

Contents

1 Reinforcement Learning

What an agent should do given some:

- prior knowledge: possible states of the worlds and possible actions
- observations: current world state and immediate reward/punishment

- goal: act to maximize accumulated reward

Assume some sequence of **experiences** with a state, action, and reward. For the agent to decide what to do next, it must decide whether to explore to gain more knowledge or exploit the knowledge it has already discovered.

Reinforcement learning is difficult due to the credit assignment problem, which is what actions are responsible for the reward but may have occurred a long time before the reward was received (long-term effect of an action). This leads to the explore-exploit dilemma: when the agent should be greedy vs inquisitive.

1.1 Main Approaches

Can search through a space of policies.

Model Based RL: learn a model consisting of state transition function $P(s' \mid a, s)$ and reward function $R(s, a, s')$ and solve this as an MDP

Model-Free RL: learn $Q^*(s, a)$ and use to guide action

2 Q-Learning

2.1 Temporal Differences

Consider some sequence $v_1, v_2, v_3, \ldots$ and take a running estimate of the first k values:

$$A_k = \frac{v_1 + \cdots + v_k}{k}$$

when a new value arrives

$$A_k = \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k$$

The **TD formula** is that, for $\alpha = \frac{1}{k}$

$$A_k = A_{k-1} + \alpha(v_k - A_{k-1})$$

2.2 Q-Learning

Store Q[State, Action] and update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).

Suppose the agent has an experience $\langle s, a, r, s' \rangle$, which provides a datum to update $Q[s,a]$. $\langle s, a, r, s' \rangle$ provides the data point

$$r + \gamma \max_{a'} Q[s', a']$$

which can be used in the TD formula giving

$$Q[s,a] \leftarrow Q[s,a] + \alpha \left(r + \gamma \max_{a'} Q[s', a'] - Q[s,a] \right)$$

Algorithm 1 Q-learning Algorithm

```
1: Initialize $Q[S, A]$ arbitrarily
2: Observe current state $s$
3: repeat
4:     Select and carry out an action $a$
5:     Observe reward $r$ and state $s'$
6:     $Q[s,a] \leftarrow Q[s,a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s,a])$
7:     $s \leftarrow s'$
8: until forever
```

2.3 Properties

Q-learning converges to the optimal policy, no matter what the agent does, as long as it tries each action in each state enough (infinitely often).

The agent can either exploit (select the action that maximizes $Q[s,a]$) or explore (select another action).

3 Strategies for Reinforcement Learning

3.1 Exploration Strategies

The ϵ **greedy strategy** is to choose a random action with probability ϵ and choose a best action with probability $1 - \epsilon$.

In **softmax action selection**, starting from state s, choose an action a with probability

$$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]}/\tau}$$

where $\tau > 0$ is the temperature. Good actions are chosen more often than bad actions and for $\tau \to \infty$ all actions are equally probably and for $\tau \to 0$, only the best action is chosen.

For optimism in the face of uncertainty, initialize Q to values that encourage exploration.

The **upper confidence bound (UCB)**, store $N[s,a]$, which is th number of times that the state-action pair has been tried) and use

$$\arg\max_a []$$

where $N[s] = \sum_a = N[s,a]$.

3.2 Off/On-Policy Learning

Q-learning does **off-policy** learning, where it learns the value of the optimal policy, no matter what it does, but this could be bad if the exploration policy is dangerous.

On-policy learning learns the value of the policy being followed.

If the agent will explore, it may be better to optimize the actual policy it is going to do.

3.3 SARSA

Uses the experience $\langle s, a, r, s', a' \rangle$ to update $Q[s,a]$. **On-policy** since it uses empirical values for s' and a'.

Algorithm 2 SARSA Algorithm

```
 1: Initialize $Q[s,a]$ arbitrarily
 2: Observe current state $s$
 3: Select action $a$ using a policy based on $Q$ (includes exploration)
 4: repeat
 5:     Carry out action $a$
 6:     Observe reward $r$ and state $s'$
 7:     Select action $a'$ using a policy based on $Q$
 8:     $Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a])$
 9:     $s \leftarrow s'$
10:     $a \leftarrow a'$
11: until forever
```

3.4 Model-Based

Uses the experiences more effectively. Used when collecting experiences is expensive and can do lots of computation between each experience.

Idea is to learn the MDP and interleave acting and planning.

After each experience, update probabilities and the reward, then do some steps of asynchronous value iteration.

Algorithm 3 Model-based Learner

```
 1: Data Structures:
 2:     $Q[S, A]$                                ▷ Action-value function
 3:     $T[S, A, S]$                             ▷ State transition counts
 4:     $R[S, A]$                                ▷ Reward function
 5: Assign $Q$, $R$ arbitrarily, $T \leftarrow$ prior counts
 6: $\alpha \leftarrow$ learning rate
 7: Observe current state $s$
 8: repeat
 9:     Select and carry out action $a$
10:     Observe reward $r$ and state $s'$
11:     $T[s, a, s'] \leftarrow T[s, a, s'] + 1$
12:     $R[s, a] \leftarrow \alpha \times r + (1 - \alpha) \times R[s, a]$
13: until done
14: repeat for a while (asynchronous VI):
15: Select state $s_1$, action $a_1$
16: Let $P = \sum_{s_2} T[s_1, a_1, s_2]$
17: $Q[s_1, a_1] \leftarrow \frac{\sum_{s_2} T[s_1,a_1,s_2]}{P} \times (R[s_1, a_1] + \gamma \max_{a_2} Q[s_2, a_2])$
18: $s \leftarrow s'$
19: end repeat
```

Let $s = (x_1, \ldots, x_N)^T$ where x are features, the Q function can be approximated linearly as

$$Q_w(s, a) \approx \sum_i w_{ai} x_i$$

and non-linearly as

$$Q_w(s, a) \approx g(x; w)$$

For experience $\langle s, a, r, s', a' \rangle$, the target Q-function is

$$R(s) + \gamma \max_a Q_w(s', a) = R(s) + \gamma Q_w(s', a')$$

and the current Q-function is

$$Q_{\bar{w}}(s, a)$$

The squared error is

$$\text{Err}(w) = \frac{1}{2}\left[Q_w(s,a) - R(s) - \gamma \max_{a'} Q_{\bar{w}}(s',a')\right]^2$$

The gradient is

$$\frac{\partial \text{Err}}{\partial w} = \left[Q_w(s,a) - R(s) - \gamma \max_{a'} Q_{\bar{w}}(s',a')\right] \frac{\partial Q_{w(s,a)}}{\partial w}$$

For SARSA with linear function approximation

Algorithm 4 SARSA with Linear Function Approximation

```
1: **Input:** Discount factor $\gamma$, Learning rate $\alpha$
2: Assign weights $w = \langle w_0, \ldots, w_n \rangle$ arbitrarily
3: begin
4:    Observe current state $s$
5:    Select action $a$
6: repeat
7:          Carry out action $a$
8:          Observe reward $r$ and state $s'$
9:          Select action $a'$ (using a policy based on $Q_w$)
10:         Let $\delta = r + \gamma Q_w(s',a') - Q_w(s,a)$
11:      for $i = 0$ to $n$ do
12:            $w_i \leftarrow w_i + \alpha \times \delta \times \frac{\partial Q_w(s,a)}{\partial w_i}$
13:      end for
14:         $s \leftarrow s'$; $a \leftarrow a'$
15: until forever
16: end
```

4 Convergence and Divergence

Linear Q-learning converges under the same conditions as Q-learning

$$w_i \leftarrow w_i + \alpha[Q_w(s,a) - R(s) - \gamma Q_w(s',a')]x_i$$

Nonlinear Q-learning may diverge, as adjusting w to increase Q at (s,a) might introduce errors at nearby state-action pairs.

To mitigate divergence, can use **experience relay** or **two Q functions** (Q network or target network).

4.1 Experience Relay

Store previous experiences (s, a, r, s', a') in a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning.

This breaks correlations between successive updates, so learning is more stable.

A few interactions with the environment are needed to converge (greater data efficiency).

4.2 Target Network

Use a separate target network that is updated only periodically. The target network has weights $\bar{w}$ and computes $Q_{\bar{w}}(s, a)$.

This should be repeated for each (s, a, r, s', a') in a mini-batch:

$$w \leftarrow w + \alpha[Q_w(s, a) - R(s) - \gamma Q_{\bar{w}}(s', a')] \frac{\partial Q_w(s, a)}{\partial w}$$

Then $\bar{w} = w$.

4.3 Deep Q Network

4.4 Bayesian Reinforcement Learning

Include the parameters (transition function and observation function) in the state space.

Model-based learning is done through inference (belief state).

The state space becomes continuous but the belief space is a space of continuous functions.

Can mitigate complexity by modeling reachable beliefs.

This gives optimal exploration-exploitation tradeoff.

Algorithm 5 Q-learning with Experience Replay

```
1: Assign weights $w = \langle w_0, \ldots, w_n \rangle$ at random in $[-1, 1]$
2: **begin**
3:     Observe current state $s$
4:     Select action $a$
5: **repeat**
6:         Carry out action $a$
7:         Observe reward $r$ and state $s'$
8:         Select action $a'$ using a policy based on $Q_w$
9:         Add $(s, a, r, s', a')$ to experience buffer
10:        Sample mini-batch of experiences from buffer
11:     **for** each experience $(\hat{s}, \hat{a}, \hat{r}, \hat{s'}, \hat{a'})$ in mini-batch **do**
12:             Let $\delta = \hat{r} + \gamma Q_w(\hat{s'}, \hat{a'}) - Q_w(\hat{s}, \hat{a})$
13:             $w \leftarrow w + \alpha \times \delta \times \frac{\partial Q_w(\hat{s}, \hat{a})}{\partial w}$
14:     **end for**
15:         $s \leftarrow s'$; $a \leftarrow a'$
16:     **if** every $c$ steps **then**
17:             Update target: $w \leftarrow w$
18:     **end if**
19: **until** forever
20: **end**
```