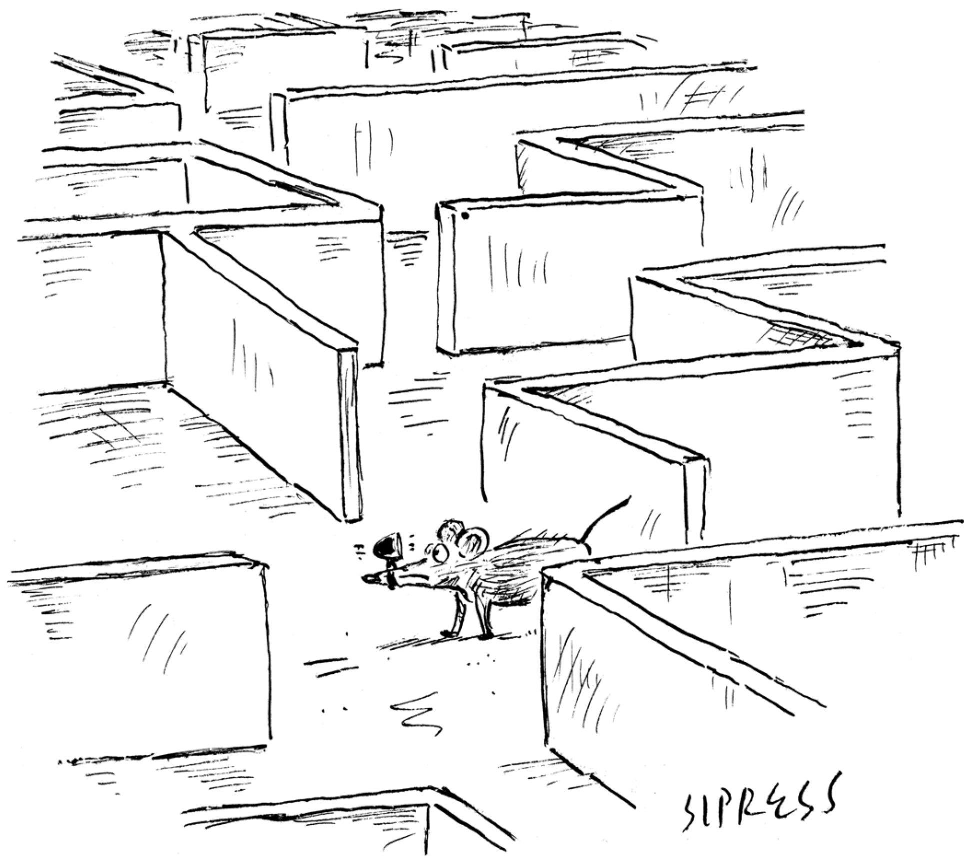
Q-learning



"Recalculating ... recalculating ..."

CSC411/2515: Machine Learning and Data Mining, Winter 2018

Michael Guerzhoy and Lisa Zhang

Recall Terminology

- State: S, Action: A, Reward: r
- Policy: $\pi_\theta(s,a)$
- Return: $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ..$
- Value Function:
$$V^{\pi_\theta}(s) = E[G_t | S_t = s]$$
- Action-Value Function:
$$q^{\pi_\theta}(a,s) = E[G_t | S_t = s, A_t = a]$$
- Relationship:
$$V^{\pi_\theta}(s) = \sum_a \pi_\theta\,(a|s)\, q^{\pi_\theta}(a,s)$$

Q-Learning

- Learn the Action-Value Function:
$$q^{\pi_\theta}(a, s) = E[G_t | S_t = s, A_t = a]$$
- Q-Learning: Learn the q function!
- Then, define policy to be
$\pi_\theta(s, a)$ = $argmax_a q^{\pi_\theta}(a, s)$ or
$\pi_\theta(s, a) \propto q^{\pi_\theta}(a, s)$
- Or use an epsilon-greedy policy:
    - Choose $\pi_\theta(s, a)$ = $argmax_a q^{\pi_\theta}(a, s)$ most of the time
    - Choose a random action some of the time

Bellman Equation (1)

- The Value Function can be decomposed:

$$V^{\pi_\theta}(s) = E[G_t|S_t = s]$$

$$\begin{aligned} V^{\pi_\theta}(\mathrm{s_t}) &= \mathrm{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ..|S_t = s] \\ &= \mathrm{E}[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + ..)|S_t = s] \\ &= \mathrm{E}[r_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathrm{E}[r_{t+1} + \gamma E[V^{\pi_\theta}(\mathrm{S_{t+1}})]|S_t = s] \end{aligned}$$

Bellman Equation (2)

- The Action-Value Function can also be decomposed:

$$q^{\pi_\theta}(a,s) = E[G_t | S_t = s, A_t = a]$$

$$\begin{aligned} q^{\pi_\theta}(a,s) & \\ &= \mathrm{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + .. | S_t = s, A_t = a] \\ &= \mathrm{E}[r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + ..) | S_t = s, A_t = a], \\ &= \mathrm{E}[r_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathrm{E}[r_{t+1} + \gamma E[V^{\pi_\theta}(\mathrm{S_{t+1}})] | S_t = s, A_t = a] \\ &= \mathrm{E}[r_{t+1} + \gamma E[q^{\pi_\theta}(A_{t+1}, S_{t+1})] | S_t = s, A_t = a] \end{aligned}$$

Optimal Policy

- Suppose we have an optimal policy π^*, then we should have the following Bellman Equations

$$V^*(\mathrm{s}) = \max_a q^*(a, s)$$

$$q^*(\mathrm{a}, \mathrm{s}) = E[r_{t+1} | S_t = s, A_t = a] + \gamma E\left[\max_{a'} q^*(a', s) \,| S_t = s, A_t = a\right]$$

For small problems where:

- There are a small number of discrete states
- We know the state transition probabilities $\mathrm{P}(\mathrm{S}_{\mathrm{t}+1} | s_t, \mathrm{a})$

We can solve this Bellman Equation directly.

Bellman equations: conditions for optimality.
Solvable with dynamic programming

Q-Learning Intuition

- Simple algorithm to find the optimal policy without knowing the state transition probabilities (known as the **model**)
- Idea: Learn a q function by training the function to satisfy the Bellman Equation

$$q^*(\mathrm{a,s}) = E[r_{t+1}|S_t = s, A_t = a] + \gamma E\left[\max_{a'} q^*(a', s_{t+1})\,|S_t = s, A_t = a\right]$$

For a sample $s_t, a_t, r_{t+1}, s_{t+1}$ from the environment. For the optimal q function we should have:

$$q^*(\mathrm{a_t, s_t}) = r_{t+1} + \gamma \max_a' q^*(a', s_{t+1})$$

Q-Learning Intuition

For a sample $s_t, a_t, r_{t+1}, s_{t+1}$ from the environment, and a q-function:

$$Loss = L\left(r_{t+1} + \gamma \max_a' q(a', s_{t+1}) - q(\mathrm{a_t}, \mathrm{s_t})\right)$$

So, train a q-function with gradient descent!

L = usually L2 loss

$$Loss = \left(r_{t+1} + \gamma \max_a' q(a', s_{t+1}) - q(\mathrm{a_t}, \mathrm{s_t})\right)^2$$

Q-Learning Algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (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

Benefits

- **Off-policy method**: samples don't have to be from the current policy (though it helps)
- Don't need to wait until episode is finished to train!
- "Learn a guess from a guess": Q-learning is just one algorithm in a family of algorithms that use this idea

Q-Learning in practise

- Sample need to be diverse enough to see everything
- **Replay buffer**: sample $(s_t, a_t, r_{t+1}, s_{t+1})$ put in replay buffer, take a batch from replay buffer to train (**Priority replay**: re-sample $(s_t, a_t, r_{t+1}, s_{t+1})$ with a large error)
- Takes longer to converge than policy gradient
- Even when the policy converged, the q-function might not

Represeting $q(s,a)$

- Lookup table
  - Learn the value of $q(s,a)$ directly
  - Works if the states and actions are discrete, and there are few of them
- $q_\theta(s,a)$
  - q_θ could be a deep (or shallow) neural network
  - Want to minimize

$$Loss = L\left(r_{t+1} + \gamma \max'_a q_\theta(a', s_{t+1}\ ) \ - q_\theta(\mathrm{a_t}, \mathrm{s_t})\right)$$

  - Tricky to do directly with gradient descent, but can try to approximate

Where to go from here?

- Reinforcement Learning (Sutton & Barto)
- David Silver's Video Lectures: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html