(10)

Reinforcement Learning II

★ How to Explore?

⇒ Several Schemes for forcing exploration

Simplest : random action (ϵ-greedy)

- → Every time step, flip a coin
- → With (small) probability ϵ, act randomly
- → With (large) probability $1-\epsilon$, act on current policy

⇒ Problem with random actions?

- → You do eventually explore the space, but keep thrashing around once learning is done.
- → One Solution : lower ϵ over time
- → Another Solution : Exploration functions

★ Exploration Function

- Random actions : Explore a fixed amount
- Better idea : Explore areas whose badness is not (yet) established, eventually stop exploring

⇒ Takes a value estimate u and a visit count n & return an optimistic utility :

$$\boxed{f(u,n) = u + k/n}$$

Modified Q-update : $$Q(s,a) \xleftarrow[\alpha]{} R(s,a,s') + \gamma \max_{a'} f\big(Q(s',a'), N(s',a')\big)$$

★ Regret

⇒ Measure of your total mistake cost

↓

Difference between your rewards & optimal rewards

regret = u(optimal action) − u(action taken)

* Approximate Q-Learning

⇒ Basic Q-Learning keeps a table of all q-values.

⇒ In realistic situation's we cannot possibly learn about every single state.
- → Too many states to visit them all in training.
- → Too many states to hold the q-tables in memory.

→ Instead, we want to generalize:
- → Learn about some small number of training states from experience.
- → Generalize that experience to new, similar situations.
- → This is a fundamental idea in machine learning, and we will see it over & over again.

(#) Feature-Based Representations

"Describe state using a vector of features"

⇒ Features are functions from states to real numbers (often 0/1) that capture important properties of the state.

→ Example:
- → Distance to closest ghost
- → Distance to closest dot
- → Number of ghosts
- → $1/(\text{dist to dot})^2$
- → Is pacman in a tunnel? (0/1)

Linear Value Function

⇒ Using feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = \omega_1 f_1(s) + \omega_2 f_2(s) + \cdots + \omega_n f_n(s)$$

$$Q(s) = \omega_1 f_1(s,a) + \omega_2 f_2(s,a) + \cdots + \omega_n f_n(s,a)$$

Advantage: Our experience is summed up in a few powerful numbers.

Disadvantage: States may share features but actually be very different in value.

⇒ Q-learning with linear Q functions:

transition $= (s, a, r, s')$

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

Exact Q: $Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$

Approximate Q: $\omega_i \leftarrow \omega_i + \alpha\,[\text{difference}]\, f_i(s,a)$

Q-Learning & Least Square

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

Error $= y - Q(s,a)$

$\hookrightarrow \omega_1 f_1 + \omega_2 f_2 \cdots \omega_n f_n$

$$\nabla_\omega \tfrac{1}{2} e^2 = \nabla_\omega (y - \omega^T f)^2 = -\tfrac{2}{2}(y - \omega^T f)\, f$$

$$\omega \leftarrow \omega + \alpha\,[\text{difference}]\, f(s,a)$$

* Policy Search

⇒ Learn policies that maximize rewards, not the value that predict them.

⇒ Simplest policy search:

- ↳ Start with an initial value function or Q function
- ↳ Nudge each feature weight up & down & see if your policy is better than before