(1)

Reinforcement Learning.

Note Title 5/18/2008

Learn how to take actions in an environment. Reward is given only at the final solution.

Example: learn to play backgammon.

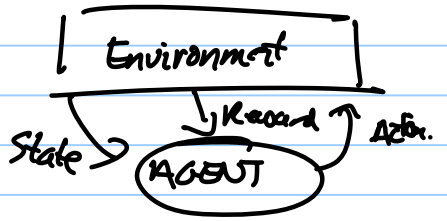
Final result → win game / lose game.

learn to fly a helicopter.

learn to move a robot to find the exit of a maze.

Decision maker – agent

Enviroment.



Problem: reward only occurs at the final solution. How to assign credit to all the actions.

credit assignment problem.

A reinforcement learner must generate an internal value for the intermediate states or actions as how good they are for leading us to the goal.

Solution requires a sequence of actions. A Markov decision process is used to model the agent.

(2) # Single State Case: K-Armed Bandit.

K-armed bandit – a slot machine with K levers.
Action – choose and pull one of the levers and win money.
Classification Problem – choose one of K.

$Q(a)$ is value of action a.
$Q(a) = 0$, for all a.
When we try action a, we get $r_a \geq 0$

Suppose rewards are deterministic (i.e. same reward for each action – i.e. each time we pull the lever).

Explore/exploit.
choose action a^* if $Q(a^*) = \max_a Q(a)$

If rewards are stochastic, we get different rewards for same action
Amount of reward $p(r|a)$.
Define $Q_t(a)$ as estimate of value of action a at time t.
The average of all rewards received when action a was chosen.
Online update:

$$Q_{t+1}(a) = Q_t(a) + \eta\{r_{t+1}(a) - Q_t(a)\}$$

(delta rule)

$r_{t+1}(a)$ reward received after taking action a at time $t+1$.
$Q_{t+1}(a)$ converges to $p(r|a)$ as time increases.

(3)

To extend to full reinforcement learning

(1) Several states. Several slot machines with different reward probabilities $P(r|s_i, a_j)$ and need to learn $Q(s_i, a_j)$ value of taking action a_j when in state s_i

(2) Actions affect not only the reward but also the next state, and we move from one state to another

(3) the rewards are delayed and we must estimate immediate values from delayed rewards

(4)

Discrete time, $t = 0, 1 \ldots$

state $s_t \in S$

action $a_t \in A(s_t)$

reward $r_{t+1} \in \mathcal{R}$

Markov Decision Process (MDP)

reward and next state are sampled from $p(r_{t+1} | s_t, a_t)$ and $P(s_{t+1} | s_t, a_t)$.

Initial state / often a terminal state (goal)

all actions in terminal state transition to it.

Episode, Trial — sequence of actions from initial state to final state

Policy π, defines the agents behavior

$\pi : S \to A$

policy defines action $a_t = \pi(s_t)$

Value of policy π $V^\pi(s_t)$ is the expected cumulative reward for following the policy starting at s_t.

Finite-horizon or episodic model, maximize the expected reward for next T steps

$$V^\pi(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{i=1}^{T} r_{t+i}\right]$$

Infinite-horizon $V^\pi(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots]$

$$= E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

$0 \le \gamma < 1$ discount rate

(5)

For each policy π, we can compute $V^{\pi}(s_t)$ and we want to find the optimal policy π^* s.t.

$$V^{*}(s_t) = \max_{\pi} V^{\pi}(s_t), \quad \forall s_t$$

In other applications, instead of working with the values of state $V(s_t)$. — we work with value of state-action pairs $Q(s_t, a_t)$. → how good is it to perform action a_t in state s_t & obeying optimal policy afterwards.

$Q^*(s_t, a_t)$ is the value, expected cumulative reward, of action a_t taken in state s_t.

Value of state is equal to the value of the best possible action.

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$

$$= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

$$= \max_{a_t} E[r_{t+1} + \gamma V^*(s_{t+1})]$$

$$V^*(s_t) = \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1})\right)$$

This is called Bellman's equation.

Similarly,

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

once we have $Q^*(s_t, a_t)$, we define the policy π as

$\pi^*(s_t)$: Choose a_t^* where $Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t)$.

Model-Based Learning

If we completely know the environment model parameters $p(r_{t+1} | s_t, a_t)$ & $P(s_{t+1} | s_t, a_t)$

Then we can solve for the optimal value function and policy by dynamic programming.

The optimal value function is the solution to Bellman's equation.

Once the optimal value function, the optimal policy is to choose the action that maximizes the value in the next state:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1} | s_t, a_t] + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) V^*(s_t) \right)$$

This is a standard control / decision theory problem - reinforcement learning was invented by neural network researchers for the case when $p(s_{t+1} | s_t, a_t)$ and $p(r_{t+1} | s_t, a_t)$ are unknown

The standard control/decision theory problem can be solved by the Value Iteration Algorithm.

(7)

Value Iteration.

To find the optimal policy, use the optimal value function.

Iterative algorithm – value iteration – converges to the correct V^* value.

Initialize $V(s)$

repeat. { For all $s \in S$,

all $a \in A$.

$Q(s,a) \rightarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$

$V(s) \rightarrow \max_a Q(s,a)$

Until convergence

Converge if $\max_{s \in S} |V^{(t+1)}(s) - V^{(t)}(s)| < \delta$.

Convergence Time $O(|S|^2|A|)$ $k < |S|$ no. of possible next states.

but often $O(k|S||A|)$

Note: it is possible to compute $E[r|s,a]$ easily because $P(r_{t+1}|s_t,a_t)$ is known, similarly we can compute $\sum_{s' \in S} P(s'|s,a) V(s')$ because $P(s_{t+1}|s_t,a_t)$ is known.

If $P(r_{t+1}|s_t,a_t)$ & $P(s_{t+1}|s_t,a_t)$ are unknown then this is harder – see temporal difference learning later.

Explore then Exploit.

Temporal Difference Learning.

Most interesting/realistic applications of reinforcement learning occur when we do not know the model.

This requires exploring the environment to query the model.

Temporal Difference (TD) algorithm.

Use ϵ-greedy search,

prob $1-\epsilon$ choose best action — exploit

prob ϵ chose one of all possible actions — explore

Choose probabilistically

$$P(a|s) = \frac{e^{Q(s,a)/T}}{\sum_b e^{Q(s,b)/T}}$$

large T exploration, small T exploit..

where $Q(s,a)$ is your current estimate of the Q-function.

Initially, $Q(s,a)$ is a bad estimate of the true $\hat{Q}(s,a)$ so start with large T (exploration strategy) and gradually reduce T as we improve our estimate of $Q(s,a)$ and move to an exploitation strategy.

(9)

Deterministic Rewards & Actions.

Special case: only one reward and one action possible.

Then Bellman's equation reduces to.

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Algorithm: Explore by strategy above

when in state s_t,

choose action a_t by one of the stochastic strategies

gives reward r_{t+1} and takes us to state s_{t+1}

Update $\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$

store $\hat{Q}(s,a)$ in table.

Initially all $\hat{Q}(s,a)$ are 0.

Suppose we only get a non-zero reward at the end of a sequence (i.e. play backgammon — only get a reward from the action that wins the game).

In intermediate states, the rewards are 0 and so the Q's stay at 0.

At goal state (e.g. win game), we get a reward for the previous state and previous action.

Next time we reach this 'previous state', we will assign a non-zero Q to the state before this.

Eventually this will propagate backwards so that we get non-zero Q's for almost all states and actions.

Note: this takes time and many sequences. Gerry Tesauro trained a computer to play backgammon by making the machine play a very large number of games against itself.

Nondeterministic Rewards and Actions

$$Q(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

This is the general case. We can't use the previous strategy because we can receive different rewards and go to different states.

We could explore the space and learn $P(s_{t+1} | s_t, a_t)$ and $p(r_{t+1} | s_t, a_t)$ — then solve the control/decision theory problem. But this is not the best strategy. Instead learning $\hat{Q}(s_t, a_t)$ will be explicitly learning the Ps

Q-learning

TD algorithm

$$\hat{Q}(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$$

think of $r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$

as a sample of instances for each (s_t, a_t).

This algorithm is guaranteed to converge to the optimal $\hat{Q}$ (Proof: Watkins & Dayan).

Extensions:

- many variants:
- Apprenticeship learning (Abeel et al. Stanford) get an expert to give you a good policy, then improve this by reinforcement learning.
- learning sequences of moves/actions

e.g. start with actions which move to a neighboring square only, then learn actions that move several squares at the same time.