Machine learning: Part 4

- Decision-theoretic planning
- Reinforcement learning

*Slides based on those of D. Poole and A. Mackworth

What is reinforcement learning?

- RL is learning what to do so as to maximize a numerical reward (or reinforcement) signal
- Learner is not told what actions to take, but must discover them by trying them out and seeing what the reward is
- Examples
  - Game - reward winning, punish losing
  - Dog - reward obedience, punish destructive behavior
  - Robot - reward task completion, punish dangerous behavior

Applications of Reinforcement Learning

- 游戏：电子游戏，棋牌游戏
  - Google DeepMind playing Atari Games
  - Alpha Go
- 机器人：
  - 机器人抓取
  - 机器人行走
  - 机器人控制
- 无人机：
  - 无人机树林中导航
- 自动驾驶：
  - 端到端控制，车道保持
  - 动态环境中决策
- 其它应用：库存管理、动态定价、广告投放

Agents as Processes

Agents carry out actions:

- forever: infinite horizon
- until some stopping criteria is met: indefinite horizon
- finite and fixed number of steps: finite horizon

Decision-theoretic Planning

What should an agent do when

- it gets rewards (and punishments) and tries to maximize its rewards received
- actions can be stochastic; the outcome of an action can't be fully predicted
- there is a model that specifies the (probabilistic) outcome of actions and the rewards
- the world is fully observable (the agent knows the state of the world from the observations)

Markov Decision Processes

We only consider stationary models where the state transitions and the rewards do not depend on the time.

An MDP consists of:

- set S of states.
- set A of actions.
- $P(s'|s, a)$ specifies the probability of transitioning to state s' given that the agent is in state s and does action a.
- $R(s, a, s')$ is the expected reward received when the agent is in state s, does action a and ends up in state s'.
- $0 \leq \gamma \leq 1$ is discount factor.

Example: to exercise or not?

Each week *Sam* has to decide whether to exercise or not:

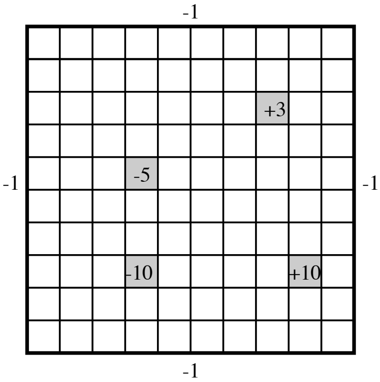
- States: $\{fit, unfit\}$
- Actions: $\{exercise, relax\}$
- Dynamics:

State	Action	$P(fit \mid State, Action)$
fit	exercise	0.99
fit	relax	0.7
unfit	exercise	0.2
unfit	relax	0.0

- Reward (does not depend on resulting state):

State	Action	Reward
fit	exercise	8
fit	relax	10
unfit	exercise	0
unfit	relax	5

Grid World Model



Grid World Model

- Actions: up, down, left, right.
- 100 states corresponding to the positions of the robot.
- Robot goes in the desired direction with probability 0.7, and one of the other 3 directions with probability 0.1.
- If it crashes into an outside wall, it remains in its current position and has a reward of -1.
- Four special rewarding states: the agent gets the reward when doing an action in that state
- In state (9,8), no matter what it does, it is flung, at random, to one of the four corners

Rewards and Values

Suppose the agent receives a sequence of rewards $r_1, r_2, r_3, r_4, \ldots$ in time. What utility should be assigned?

- total reward $V = \sum_{i=1}^{\infty} r_i$
  but if the sum is infinite, unable to compare such sequences
- average reward $V = \lim_{n\to\infty} (r_1 + \cdots + r_n)/n$
  However, whenever the total reward is finite, the average reward is zero, hence unable to compare such sequences
- discounted return $V = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$
  Under this criterion, future rewards are worth less than the current reward.

Properties of the Discounted Rewards

- The discounted return for rewards $r_1, r_2, r_3, r_4, \ldots$ is

$$\begin{aligned} V &= r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots \\ &= r_1 + \gamma(r_2 + \gamma(r_3 + \gamma(r_4 + \ldots))) \end{aligned}$$

- If V_t is the value obtained from time step t

$$V_t = r_t + \gamma V_{t+1}$$

- $1 + \gamma + \gamma^2 + \gamma^3 + \cdots = 1/(1-\gamma)$
  Therefore $\dfrac{\text{minimum reward}}{1-\gamma} \leq V_t \leq \dfrac{\text{maximum reward}}{1-\gamma}$

- We can approximate V with the first k terms, with error:

$$V - (r_1 + \gamma r_2 + \cdots + \gamma^{k-1} r_k) = \gamma^k V_{k+1}$$

Policies (策略)

- A stationary policy is a function:

$$\pi : S \to A$$

Given a state s, $\pi(s)$ specifies what action the agent who is following π will do.

- An optimal policy is one with maximum expected discounted reward.
- For a fully-observable MDP with stationary dynamics and rewards with infinite or indefinite horizon, there is always an optimal stationary policy.

How many stationary policies are there?

- Each week *Sam* has to decide whether to exercise or not:
  - States: $\{fit, unfit\}$
  - Actions: $\{exercise, relax\}$
- the grid world with 100 states and 4 actions

Value of a Policy

Given a policy π:

- $Q^{\pi}(s,a)$: the expected value of doing action a in state s, then following policy π.
- $V^{\pi}(s)$: the expected value of following policy π in state s.
- Q^{π} and V^{π} can be defined mutually recursively:

$$
\begin{aligned}
Q^{\pi}(s,a) &= \sum_{s'} P(s'|a,s)\left(R(s,a,s') + \gamma V^{\pi}(s')\right) \\
V^{\pi}(s) &= Q(s,\pi(s))
\end{aligned}
$$

Value of the Optimal Policy

- $Q^*(s,a)$: the expected value of doing action a in state s, then following the optimal policy.
- $V^*(s)$: the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

$$
\begin{aligned}
Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(R(s,a,s') + \gamma V^*(s')\right) \\
V^*(s) &= \max_a Q^*(s,a) \\
\pi^*(s) &= argmax_a Q^*(s,a)
\end{aligned}
$$

Value Iteration

A method of computing an optimal policy and its value.

- Let V_k and Q_k be k-step lookahead value and Q functions.
- Set V_0 arbitrarily.
- Compute Q_{k+1}, V_{k+1} from V_k.
- This converges exponentially fast (in k) to the optimal value function.

  The error reduces proportionally to $\dfrac{\gamma^k}{1-\gamma}$

Asynchronous Value Iteration

- Do not sweep through all the states, but update the value functions for each state individually.
- This converges to the optimal value functions, if each state and action is visited infinitely often in the limit.
- It can either store $V[s]$ or $Q[s, a]$.
- Repeat forever:
  - Select state s
  - $V[s] \leftarrow \max_a \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma V[s']\right)$
- Repeat forever:
  - Select state s, action a
  - $Q[s, a] \leftarrow \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q[s', a']\right)$

Example: to exercise or not?

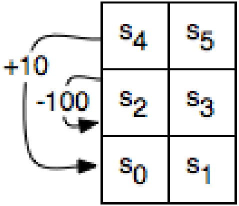
Let $\gamma = 0.9$

- Iteration 0: $\bar{V} = (0, 0)$
- Iteration 1: $\bar{V} = (10, 5)$
  - $(f, e) : 8, \;\; (f, r) : 10$
  - $(u, e) : 0, \;\; (u, r) : 5$
- Iteration 2: $\bar{V} = (17.65, 9.5)$
  - $(f, e) : 0.99(8 + 0.9 \cdot 10) + 0.01(8 + 0.9 \cdot 5) = 16.955$
  - $(f, r) : 0.7(10 + 0.9 \cdot 10) + 0.3(10 + 0.9 \cdot 5) = 17.65$
  - $(u, e) : 0.2(0.9 \cdot 10) + 0.8(0.9 \cdot 5) = 5.4$
  - $(u, r) : (5 + 0.9 \cdot 5) = 9.5$
- Iteration 3: $\bar{V} = (23.812, 13.55)$
  - $(f, e) : 0.99(8 + 0.9 \cdot 17.65) + 0.01(8 + 0.9 \cdot 9.5) = 23.812$
  - $(f, r) : 0.7(10 + 0.9 \cdot 17.65) + 0.3(10 + 0.9 \cdot 9.5) = 23.685$
  - $(u, e) : 0.2(0.9 \cdot 17.65) + 0.8(0.9 \cdot 9.5) = 10.017$
  - $(u, r) : (5 + 0.9 \cdot 9.5) = 13.55$

Reinforcement learning

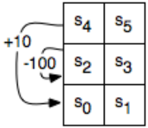
Like decision-theoretic planning, except model of dynamics and model of reward not given.

A tiny example



- There are 6 states $s_0, \ldots, s_5$.
- The agent has 4 actions: UpC, Up, Left, Right.
- upC ("up carefully"): goes up, except in states s4 and s5, where the agent stays still, and has a reward of -1.

The tiny example



- right: moves to the right in states s0,s2,s4 with a reward of 0 and stays still in the other states, with reward -1.
- left: moves to the left in s1,s3,s5. In s0, it stays with reward -1. In s2, it stays with reward -100. In s4, it moves to s0 with reward 10.
- up: With probability 0.8 it acts like upC, except the reward is 0. With probability 0.1 it acts as a left, and with probability 0.1 it acts as right.

How should the agent act?

Reinforcement learning: main approaches:

- search through a space of policies to find the best policy, *e.g.*, using evolutionary algorithms
- learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a, s')$; solve this as an MDP.
- learn $Q^*(s, a)$, use this to guide action.

Experiential Asynchronous Value Iteration

initialize $Q[S, A]$ arbitrarily
observe current state s
repeat forever:
    select and carry out an action a
    observe reward r and state s'
    $Q[s, a] \leftarrow r + \gamma \max_{a'} Q[s', a']$
    $s \leftarrow s'$

Temporal Differences (时序差分)

- Suppose we have a sequence of values: $v_1, v_2, v_3, \ldots,$ and the goal is to predict the next value, given all of the previous values
- One way to do this is to have a running estimate of the average of the first k values:

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

- *e.g.*, given a sequence of students' grades and the aim of predicting the next grade, a reasonable prediction is to predict the average grade

Temporal Differences (cont)

- Suppose we know A_{k-1} and a new value v_k arrives:
$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k} = \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k$$
- Let $\alpha_k = \frac{1}{k}$, then
$$A_k = (1 - \alpha_k) A_{k-1} + \alpha_k v_k = A_{k-1} + \alpha_k (v_k - A_{k-1})$$
- The difference $v_k - A_{k-1}$ is called the temporal difference error or TD error
- it specifies how different the new value v_k is from the old prediction A_{k-1}

TD formula

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

- To get the new estimate, the old estimate is updated by α_k times the TD error
- The idea: if the new value is higher than the old prediction, increase the predicted value;
- if the new value is less than the old prediction, decrease the predicted value.

The choice of α_k

- Setting $\alpha_k = \frac{1}{k}$ assumes that all values have an equal weight
- In RL, the latter values of v_i are more accurate than the earlier values and should be weighted more
- One way to weight later examples more is to set α as a constant $(0 < \alpha \leq 1)$.
- Unfortunately, this does not converge to the average
- You can guarantee convergence if

$$\sum_{k=1}^{\infty} \alpha_k = \infty \text{ and } \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

Q-learning

- Idea: store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- An experience $\langle s, a, r, s' \rangle$ provides a new estimate for the value of $Q^*(s, a)$:

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

which can be used in the TD formula giving:

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

Q-learning

initialize $Q[S, A]$ arbitrarily
observe current state s
repeat forever:
  select and carry out an action a
  observe reward r and state s'
  $Q[s, a] \leftarrow Q[s, a] + \alpha\,(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s \leftarrow s'$

The tiny example

Let $\gamma = 0.9$, $\alpha = 0.2$; all Q values are initialized to 0.
Here is a sequence of experiences and the update:

s	a	r	s'	Update
s0	upC	-1	s2	Q[s0,upC]=-0.2
s2	up	0	s4	Q[s2,up]=0
s4	left	10	s0	Q[s4,left]=2.0
s0	upC	-1	s2	Q[s0,upC]=-0.36
s2	up	0	s4	Q[s2,up]=0.36
s4	left	10	s0	Q[s4,left]=3.6
s0	up	0	s2	Q[s0,upC]=0.06
s2	up	-100	s2	Q[s2,up]=-19.65
s2	up	0	s4	Q[s2,up]=-15.07
s4	left	10	s0	Q[s4,left]=4.89

$0.8 \times 0.36 + 0.2 \times (-100 + 0.9 \times 0.36) = -19.65$
$0.8 \times -19.65 + 0.2 \times (0 + 0.9 \times 3.6) = -15.07$

Properties of Q-learning

- Q-learning converges to an optimal policy, no matter what the agent does, as long as it tries each action in each state enough.
- But what should the agent do?
  - exploit: when in state s, select an action that maximizes $Q[s, a]$
  - explore: select another action

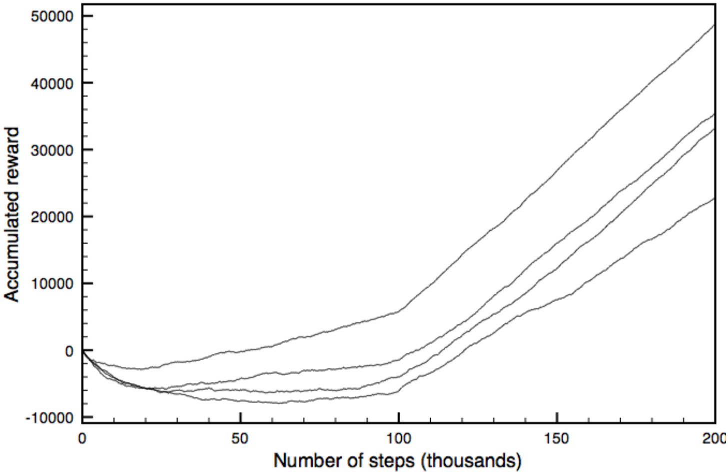
Exploration Strategies

- The ϵ-greedy strategy:
  choose a random action with probability ϵ and a best action with probability $1 - \epsilon$.
- Softmax action selection: in state s, choose 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.
  τ defines how much a difference in Q-values maps to a difference in probability.

Evaluating Reinforcement Learning Algorithms



One algorithm dominates another if its plot is consistently above the other.

On-policy Learning

- Q-learning does off-policy learning: it learns the value of an optimal policy, no matter what it does.
- This could be bad if the exploration policy is dangerous.
- On-policy learning learns the value of the policy being followed. e.g., act greedily 80% of the time and act randomly 20% of the time
- Why? If the agent is actually going to explore, it may be better to optimize the actual policy it is going to do.
- SARSA uses the experience $\langle s, a, r, s', a' \rangle$ to update $Q[s, a]$, here a' is what the agent decides to do in s'

SARSA (state-action-reward-state-action)

initialize $Q[S, A]$ arbitrarily
observe current state s
select action a using a policy based on Q
repeat forever:
    carry out action a
    observe reward r and state s'
    select action a' using a policy based on Q
    $Q[s, a] \leftarrow Q[s, a] + \alpha\,(r + \gamma Q[s', a'] - Q[s, a])$
    $s \leftarrow s'$
    $a \leftarrow a'$