Quick Intro into Reinforcement Learning

Artem Sokolov

Institute for Computational Linguistics, Heidelberg University

10 October 2018

- this is not a replacement to the RL course (summer semester)
- we will only introduce basic concepts and notation
- slides from the David Silver's course
  - ➡ `www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`
  - ➡ + videos!
- which in turn is based on the RL book by Sutton and Barto
  - ➡ `http://incompleteideas.net/book/the-book-2nd.html`

RL is a branch of science that studies decision making

How is RL different from other ML learning paradigms?

- No direct supervision, only a reward signal
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives
- feedback is delayed, not instantaneous

- A reward R_t is a **scalar** feedback signal
  - ➡ being a scalar is quite a natural requirement, because at some point the agent needs to rank actions and pick one
  - ➡ but it's also one of the sources of problems which IL aims to solve
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward

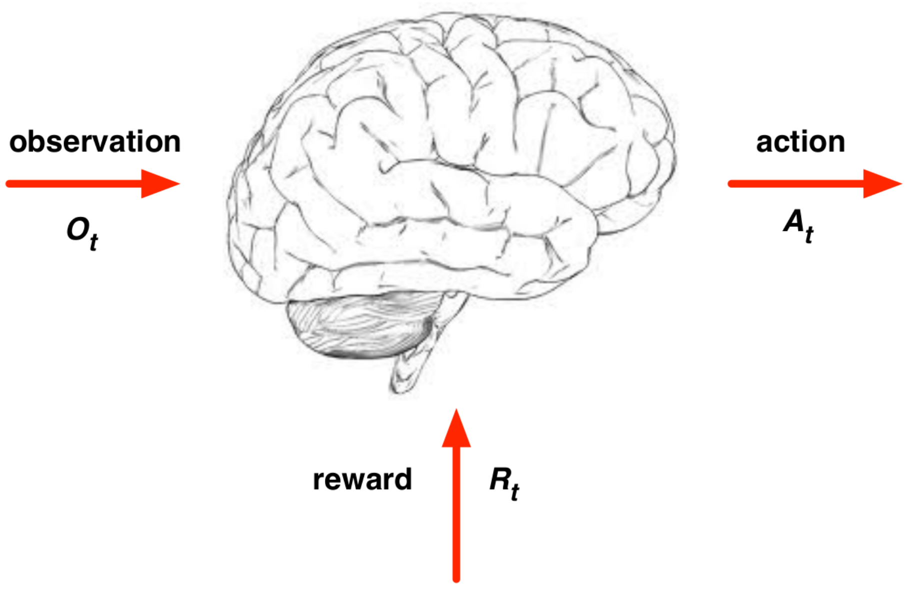
RL is based on:

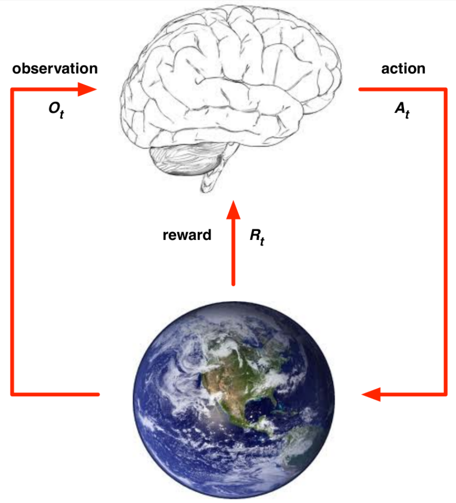
Reward Hypothesis

All goals can be described by the maximisation of expected cumulative reward

- drone flying
  - negative for crashing
  - positive for finishing
- driving
  - negative for crashing another car, running over a pedestrian, etc.
  - positive for following the speed limits, keeping distance, etc.
- games
  - negative for dying and/or for each time unit passed
  - positive for eating ‘food’, collecting ‘treasures’, finishing a level
- investment
  - positive for each $ of revenue
  - negative for each $ of loss
- humanoid robot walk
  - negative for falling
  - positive for the height of head, for forward motion

- Goal: select actions to maximise total future reward
- Actions may have long term consequences
- Reward may be delayed
- May be better to sacrifice immediate reward to gain more in long-term
- Examples:
  - ➡ Financial investment (may take months to mature)
  - ➡ Refuelling a helicopter (might prevent a crash in several hours)
  - ➡ Sacrificing a figure in chess (might help winning many moves from now)


observation
O_t
action
A_t
reward
R_t



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at environment's step

- The history is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$$

  - ➡ i.e. all observable variables up to time t
  - ➡ i.e. the sensorimotor stream of a robot
- State is the information used to determine what happens next
- Formally, state is a function of the history:

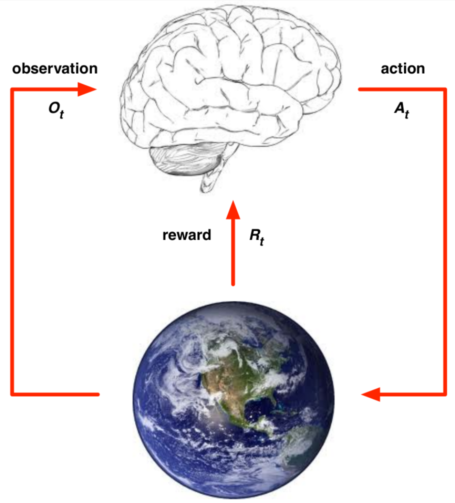
$$S_t = f(H_t)$$

- distinction between a state of the environment and the agent's state:
  - ➡ environment state is usually unobservable
  - ➡ even if it partially is, may be irrelevant
- agent's state
  - ➡ this is the basis for agent's decision taking and RL algorithms
  - ➡ may partially include the environment state through sensors
  - ➡ and this is usually the function $S_t = function\_of\_our\_choice(H_t)$

- a state is Markov if contains all useful info for decision taking
- definition a state S_t is Markov iff

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \ldots, S_t]$$

- in other words S_t is a sufficient statistics
- H_t is trivially Markov
- the state of the environment is also Markov (by definition)



Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment

Major Components of an RL Agent

- Policy: agent's behaviour function
- Value function: how good is each state and/or action
- Model: agent's representation of the environment

- A policy is the agent's behaviour
- It is a map from state to action, e.g.
  - ➡ Deterministic policy: $a = \pi(s)$
  - ➡ Stochastic policy: $\pi(a|s) = P[A_t = a|S_t = s]$

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

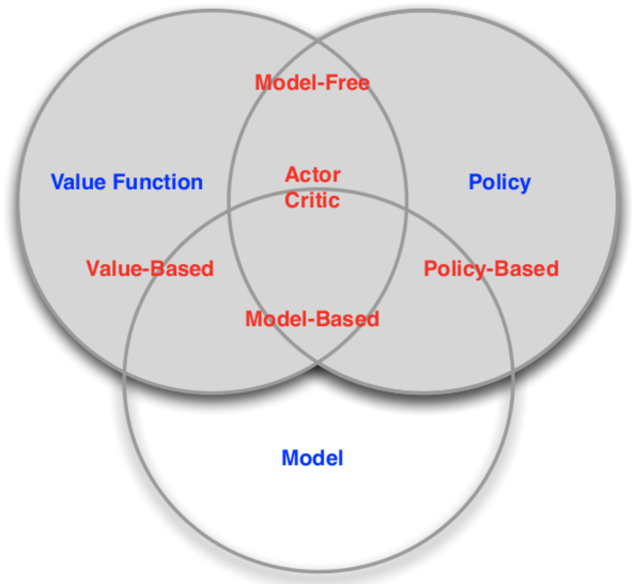
$$V(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$$

- A model predicts what the environment will do next
- P predicts the next state
- R predicts the next (immediate) reward, e.g.

$$P^a_{ss'} = P[S_{t+1} = s | S_t = s, A_t = a]$$
$$R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

- Value based
  (no explicit policy, value function)
- Policy based
  (explicit policy, no value function)
- Actor-Critic based
  (policy, value function)

- Model-based
  (policy and/or value function + model)
- Model-free
  (policy and/or value function, no model)


Model-Free
Value Function
Actor
Critic
Policy
Value-Based
Policy-Based
Model-Based
Model

- Learning
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. search

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

- exploration finds more information about the environment
- exploitation exploits known information to maximise reward
- both are important

- Restaurant Selection
  - ➡ Exploitation - Go to your favourite restaurant
  - ➡ Exploration - Try a new restaurant
- Online Banner Advertisements
  - ➡ Exploitation - Show the most successful advert
  - ➡ Exploration - Show a different advert
- Oil Drilling
  - ➡ Exploitation - Drill at the best known location
  - ➡ Exploration - Drill at a new location
- Game Playing
  - ➡ Exploitation - Play the move you believe is best
  - ➡ Exploration - Play an experimental move

- Prediction: evaluate a policy (find $V(s)$)
- Control: find the best policy

Markov Decision Process

Markov decision processes formalize an environment for RL

- Where the environment is fully observable (i.e. the current state completely characterises the process)
- Almost all RL problems can be formalised as MDPs,
- Optimal control primarily deals with continuous MDPs
- Bandits are MDPs with one state

- a state is Markov if contains all useful info for decision taking
- definition: a state S_t is Markov iff

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \dots, S_t]$$

- in other words S_t is a sufficient statistics
- H_t is trivially Markov
- the state of the environment is also Markov (by definition)

For a Markov state s and successor state s', the state transition probability is defined by

$$P_{ss'} = P[S_{t+1} = s' | S_t = s]$$

State transition matrix P defines transition probabilities from all states s to all successor states s'

$$P = \begin{array}{cc} & \text{to} \\ \text{from} & \begin{bmatrix} P_{11} \dots P_{1n} \\ \dots \\ P_{n1} \dots P_{nn} \end{bmatrix} \end{array}$$

Markov Process

Definition

A Markov Process is a tuple (S, P)

- s is a (finite) set of state
- P is a transition matrix $P_{ss'} = P[S_{t+1} = s' | S_t = s]$

Markov Reward Process

Definition

A Markov Process is a tuple (S, P, R, γ)

- s is a (finite) set of state
- P is a transition matrix $P_{ss'} = P[S_{t+1} = s' | S_t = s]$
- R is a reward function, $R(s) = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

- Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  - ➡ $\gamma \simeq 0$ – short-sighted evaluation
  - ➡ $\gamma \simeq 1$ – far-sighted evaluation
  - ➡ why: math convenience, evidence in nature, less emphasis on future
- Value function:

$$V(s) = \mathbb{E}[G_t | S_t = s]$$

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots |S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]
\end{aligned}
$$

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \end{aligned}$$

In the matrix form:

$$V(s) = R(s) + \gamma \sum_{s' \in S} P_{ss'} V(s')$$

$$V = R + \gamma P V$$

where V, R are column-vectors, and P is a matrix

$$
\begin{aligned}
V =& R + \gamma P V \\
V(I - \gamma P) =& R \\
V =& (I - \gamma P)^{-1} R
\end{aligned}
$$

- expensive: $O(n^3)$
- matrix inversion feasible only for small MDPs
- iterative methods for large MDPs

- all episodes start in the center state, C
- proceed either left or right by one state on each step, with equal probability.
- episodes terminate either on the extreme left or the extreme right
- when an episode terminates on the right, a reward of $+1$ occurs; all other rewards are zero.
- because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state.
- value of the center state is $v_\pi(C) = 0.5$.
- values of all the states, A through E, are $1/6$, $2/6$, $3/6$, $4/6$, $5/6$

- all episodes start in the center state, C
- proceed either left or right by one state on each step, with equal probability.
- episodes terminate either on the extreme left or the extreme right
- when an episode terminates on the right, a reward of $+1$ occurs; all other rewards are zero.
- because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state.
- value of the center state is $v_\pi(C) = 0.5$.
- values of all the states, A through E, are 1/6, 2/6, 3/6, 4/6, 5/6

Exercise: write a Python program that calculates it

```
import numpy as np
I = np.eye(7)
R = np.array([0,0,0,0,0,0.5,0])
P = np.array([[1,0,0,0,0,0,0],
              [0.5,0,0.5,0,0,0,0],
              [0,0.5,0,0.5,0,0,0],
              [0,0,0.5,0,0.5,0,0],
              [0,0,0,0.5,0,0.5,0],
              [0,0,0,0,0.5,0,0.5],
              [0,0,0,0,0,0,1]])
V = np.dot(np.dot(np.linalg.inv(I-P), P), R)
```

Markov Reward Process

Definition

A Markov Process is a tuple (S, P, R, γ)

- S is a (finite) set of state
- P is a transition matrix $P_{ss'} = P[S_{t+1} = s' | S_t = s]$
- R is a reward function, $R(s) = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Decision Process

Definition

A Markov Process is a tuple (S, P, A, R, γ)

- S is a (finite) set of state
- A is a (finite) set of actions
- P is a transition matrix $P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$
- R is a reward function, $R^a(s) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Process

Definition

A Markov Process is a tuple (S, P, A, R, γ)

- S is a (finite) set of state
- P is a transition matrix $_{ss'} = P[S_{t+1} = s' | S_t = s]$
- R is a reward function, $(s) = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

New concept:

- Policy π is a distribution over actions given state:

$$\pi(a|s) = P[A_t = a | S_t = s]$$

- policy = behaviour of the agent
- depends on the Markov state

- MDP (S, A, P, R, γ) and policy π
- can reduce to an MRP $(S, P^{\pi}, R^{\pi}, \gamma)$

$$P^{\pi}_{ss'} = \sum_{a \in A} \pi(a|s) P^{a}_{ss'}$$

$$R^{\pi}_{ss'} = \sum_{a \in A} \pi(a|s) R^{a}_{ss'}$$

Classes:

- wrt time:
  - stationary (=time-independent)
  - non-stationary (=time-dependent)
- wrt certainty:
  - deterministic ($\forall s \exists a, \pi(a|s) = 1$)
  - stochastic (otherwise)

- was: state-value function:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

$$V_\pi(s) = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}|S_t = s]$$

- new: action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s_t, A_t = a]$$

$$q_\pi(s, a) = \mathbb{E}_\pi[R_t + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s_t, A_t = a]$$

Value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$$

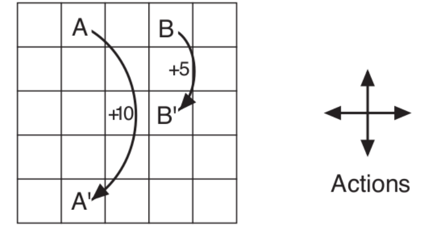
Similarly for action-value function:

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = s]$$

$$v(s) = \sum_a \pi(s,a) q(s,a)$$

$$v(s) = \sum_a \pi(s,a)[r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s')]$$

$$v(s) = \sum_a \pi(s,a)[r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a \in A} \pi(s',a') q(s',a')]$$



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Exercise: confirm that the center cell has $v(s) \simeq 0.7$ using $v(s)$ of it's neighbors

Definition

Optimal state-value function $v^*(s)$ is the max value-function over πs:

$$v^*(s) = \max_{\pi} v_{\pi}(s)$$

Similarly, for action-value function

$$q^*(s,a) = \max_{\pi} q_{\pi}(s,a)$$

- they characterize the best possible performance in the MDP
- to "solve an MDP" is to find v^* or q^*

Thm

For any MDP

- $\exists \pi^*$, s.t. $\pi^* \geq \pi, \forall \pi$ (possibly non-unique)
- For any optimal policy

$$v_{\pi^*}(s) = v^*(s)$$
$$q_{\pi^*}(s, a) = q^*(s, a)$$

An optimal policy can be found by maximising over a,

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a \in A} q^*(s,a) \\ 0, & \text{otherwise} \end{cases}$$

- There is **always** a deterministic optimal policy for any MDP
- If we know $q^*(s,a)$, we immediately have the optimal policy

$$v(s) = \sum_{a} \pi(s,a) q(s,a)$$

$$q(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v(s')$$

$$v(s) = \sum_{a} \pi(s,a) [R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s')]$$

$$q(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a \in A} \pi(s', a') q(s', a')$$

$$v^*(s) = \max_a \pi(s,a) q^*(s,a)$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v^*(s')$$

$$v^*(s) = \max_a [R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')]$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a \in A} \pi(s',a') q^*(s',a')$$

Solving MDPs

A method for solving complex problems, by breaking them down into subproblems:

- Solve the subproblems
- Combine solutions to subproblems

Requirements:

- Optimal substructure (Principle of optimality applies, Optimal solution can be decomposed into subproblems)
- Overlapping subproblems (Subproblems recur many times, Solutions can be cached and reused)
- Markov decision processes satisfy both properties (Bellman equation gives recursive decomposition, Value function stores and reuses solutions)

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP (which also assumes full knowledge)
- For prediction:
  - ➡ Input: MDP (S, A, P, R, γ) and policy π
  - ➡ Output: value function v_π
- For control:
  - ➡ Input: MDP (S, A, P, R, γ)
  - ➡ Output: optimal value function v^*
  - ➡ and: optimal policy π^*

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman **expectation** backup

$$v_1 \to v_2 \to \cdots \to v_\pi$$

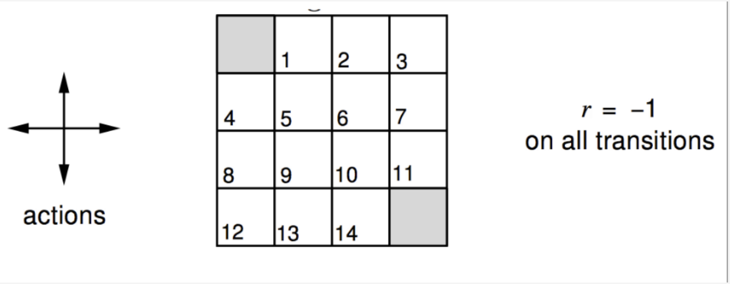
- synchronously:
  - ➡ At each iteration $k+1$
  - ➡ For all states $s \in S$
  - ➡ Update $v_{k+1}(s)$ from $v_k(s')$ (by taking $\mathbb{E}$)
  - ➡ where s' is a successor state of s

$$v_{k+1}(s) = \sum \pi(a|s)(r_a^s + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$$

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

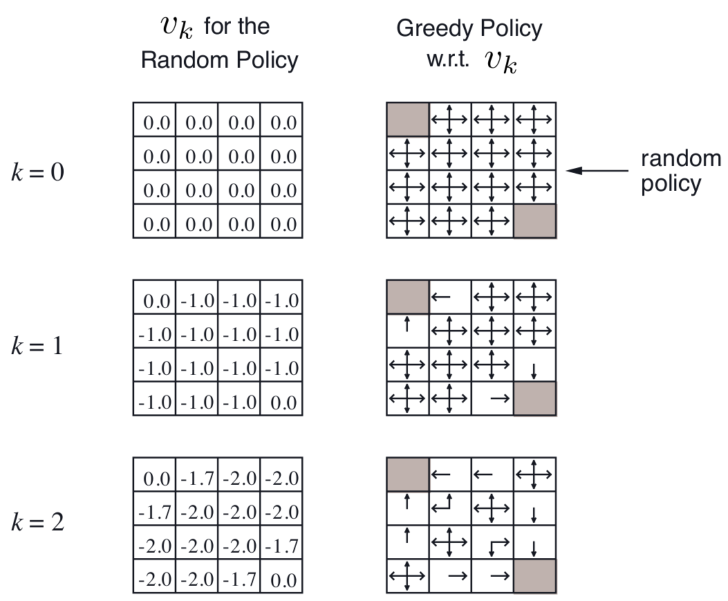
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

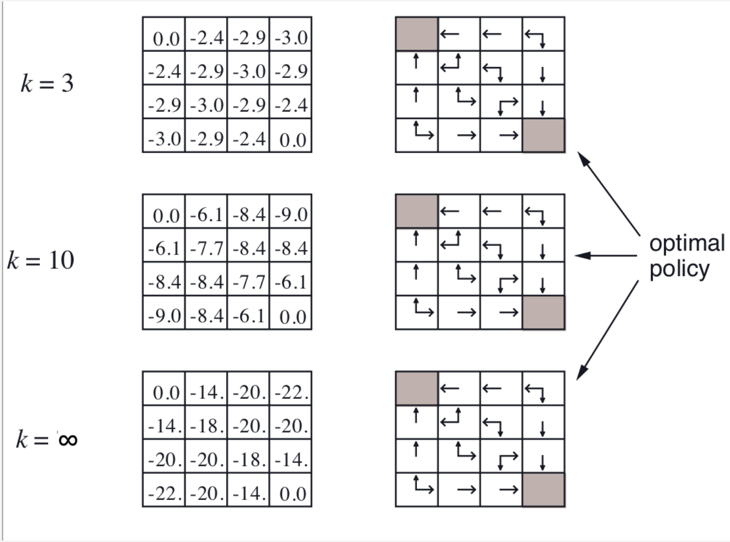
Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$


v_k for the Random Policy
Greedy Policy w.r.t. v_k
$k = 0$
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
random policy
$k = 1$
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
$k = 2$
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0


k = 3
0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0
k = 10
0.0 -6.1 -8.4 -9.0
-6.1 -7.7 -8.4 -8.4
-8.4 -8.4 -7.7 -6.1
-9.0 -8.4 -6.1 0.0
k = ∞
0.0 -14. -20. -22.
-14. -18. -20. -20.
-20. -20. -18. -14.
-22. -20. -14. 0.0
optimal policy

How to Improve a Policy?
Given an initial policy π:

- Evaluate the policy π

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$$

- Improve the policy by acting greedily with respect to v_π

$$\pi' = greedy(v_\pi)$$

- In general, many iterations of improvement / evaluation
- this process of policy iteration always converges to π^*!

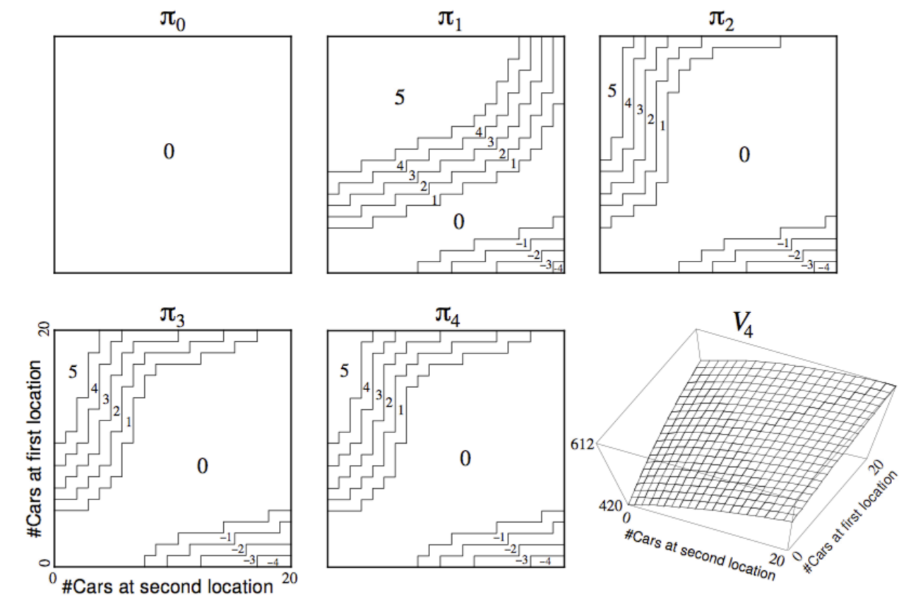
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r \,|\, s,\pi(s)) \big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \,|\, s,a) \big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: $10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly
  - ➡ Poisson distribution, n returns/requests with prob $\frac{\gamma^n}{n!}e^{-\gamma}$
  - ➡ 1st location: average requests $=$ 3, average returns $=$ 3
  - ➡ 2nd location: average requests $=$ 4, average returns $=$ 2



π_0
0
π_1
5
4
3
2
1
0
−1
−2
−3
−4
π_2
5
4
3
2
1
0
−1
−2
−3
−4
π_3
5
4
3
2
1
0
−1
−2
−3
−4
20
0
#Cars at first location
0
#Cars at second location
20
π_4
5
4
3
2
1
0
−1
−2
−3
−4
V_4
612
420
0
#Cars at second location
20
0
20
#Cars at first location

- Consider a deterministic policy, $a = \pi(s)$
- We cannot deteriorate it by acting greedily

$$\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$$

- This improves or keeps the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function,

$$v_{\pi'}(s) \geq v_\pi(s)$$

$$\begin{aligned}
v_\pi(s) \leq & q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
\leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi(S_{t+1})) | S_t = s] \\
\leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1} + \gamma^2 q_\pi(S_{t+2}) | S_{t+1} = 2) | S_t = s] \\
\leq & v_{\pi'}(s)
\end{aligned}$$

Very similar to policy evaluation, diffs are in the form of v-update:

- Problem: find optimal policy π
- Solution: iterative application of Bellman **optimality** backup

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v^*$$

- Using synchronous backups
  - ➡ At each iteration $k + 1$
  - ➡ For all states $s \in S$
  - ➡ Update $v_{k+1}(s)$ from $v_k(s')$ (by taking $\max$)
- Convergence to v^*
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

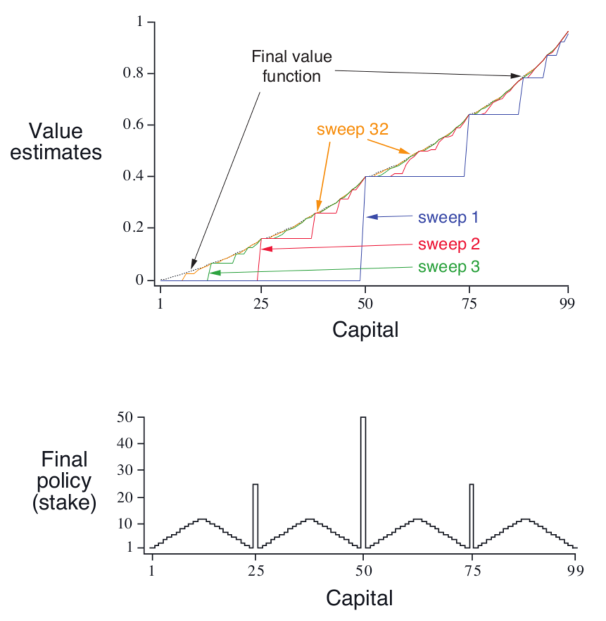
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

Question: Which algorithm does it remind for CRFs?

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- A gambler makes bets on coin flips
- If the coin comes up heads, he wins as many dollars as he has staked
- if it is tails, he loses his stake
- p_h is the probability of getting coin heads
- The game ends when the gambler wins/loses by reaching \$100 or \$0
- On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.
- The state is the gambler's capital, $s \in \{1, 2, \ldots, 99\}$
- The actions are stakes, $a \in \{0, 1, \ldots, \min(s, 100 - s)\}$
- Reward $=$ 0 on all transitions, except it's $+1$ when reaching \$100


Value estimates
Final value function
sweep 32
sweep 1
sweep 2
sweep 3
1
0.8
0.6
0.4
0.2
0
1
25
50
75
99
Capital
Final policy (stake)
50
40
30
20
10
1
1
25
50
75
99
Capital

- use a value-iteration skeleton code
- build the last plot

Model-free RL

- relied on know model P and R
- rarely the case in practice
- need model-free methods

Types of model-free prediction

- Monte-Carlo
- Temporal-Difference

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
- All episodes must terminate

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Recall that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

- On-policy learning
  - ➡ 'Learn on the job'
  - ➡ Learn about policy π from experience sampled from π
- Off-policy learning
  - ➡ 'Look over someone's shoulder'
  - ➡ Learn about policy π from experience sampled from μ

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop 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