Reinforcement Learning

CE811 Game Artificial Intelligence

Joseph Walton-Rivers

12th March 2021

University of Essex

Outline

Reinforcement Learning

- Components of an RL agent
- Example: Maze

Markov Decision Processes

- Example: Rogue

Dynamic Programming

- The Small Gridworld

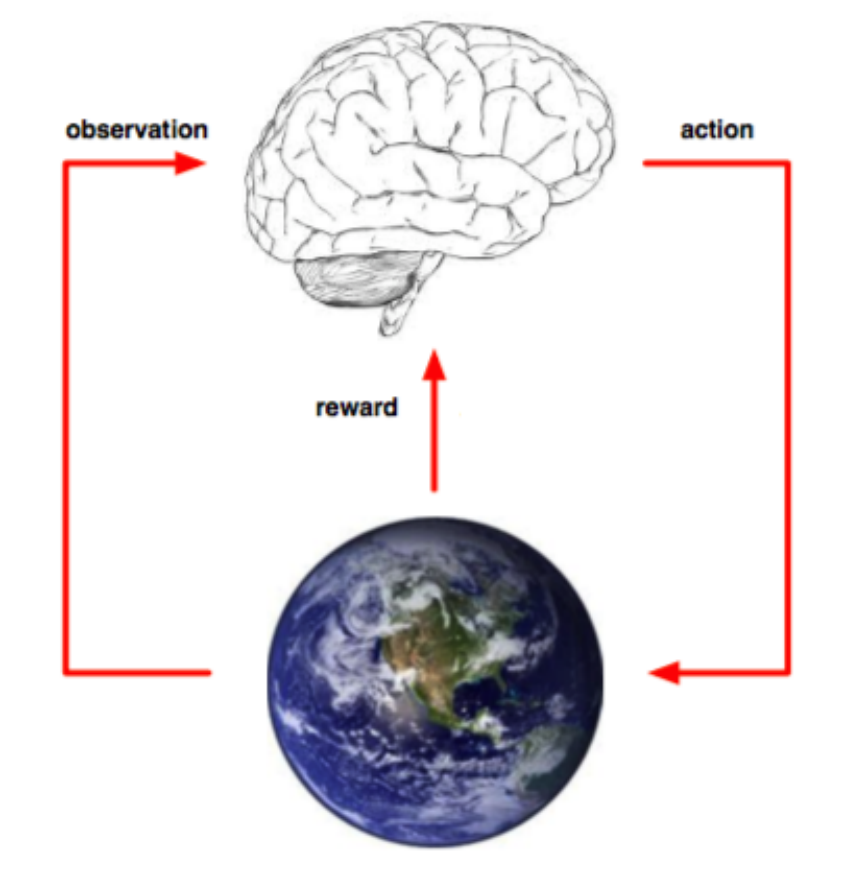
Monte Carlo Methods

Planning Vs Reinforcement Learning

- Planning is using a model to ‘think ahead’ and generate actions
  - We can apply actions in our head and see what happens
  - Our model is *perfect*
- in Reinforcement Learning, we don’t know the rules
  - We don’t know the rules (we have no model)
  - We just get plonked into the world and told to get on with it

The reinforcement Learning problem

- No supervisor, only a reward signal
- Feedback is delayed
- Sequential samples, not iid
- Actions influence future observations
- Example: games (but not only!)



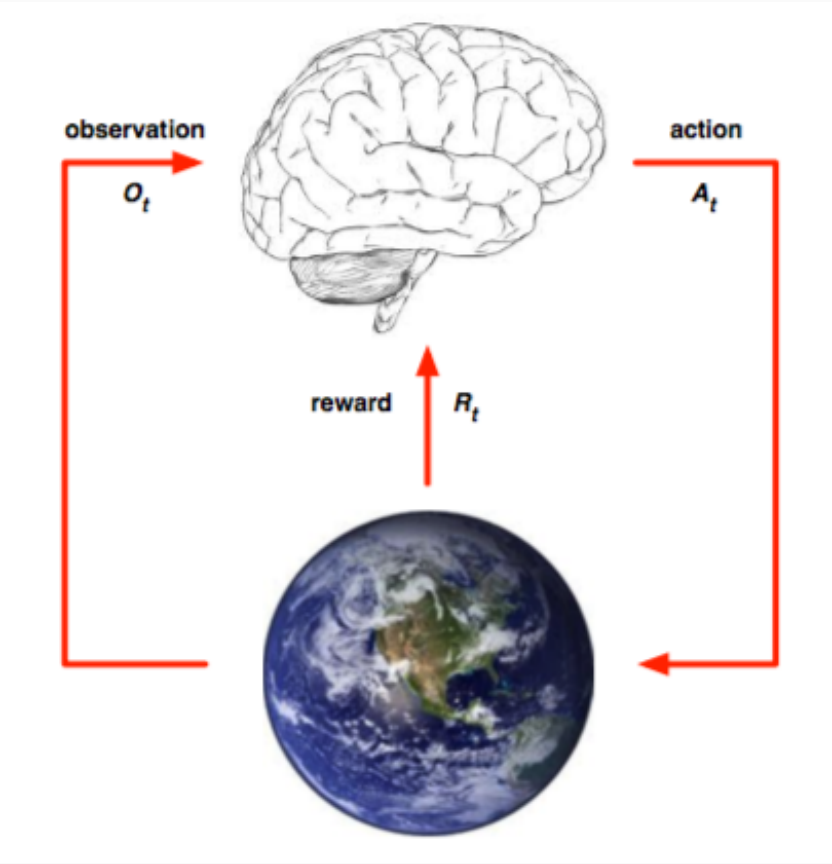
Reinforcement Learning

The Reinforcement Learning problem

- Reward
  - R_t is a scalar feedback signal
  - Indicates how well the agent is doing at step t
  - The agent's job is to maximize cumulative reward

- Sequential Decision Making
  - Goal: select actions to maximize total future reward
  - Actions may have long term consequences
  - Reward may be delayed
  - Immediate vs. long-term reward

The Reinforcement Learning problem



- 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 every step

History

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

$$H_t = O_1 R_1 A_1, \ldots, O_{t-1} R_{t-1} A_{t-1}$$

- In the general case, what happens next depends on the history.
- State is the information used to determine what happens next.
- State is a function of the history:

$$S_t = f(H_t)$$

The Markov State

- A Markov state contains all useful information from the history

Definition (Markov State)

A state S_t is **Markov** if and only if:

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, \ldots, S_t)$$

- ”The future is independent of the past given the present” (if S_t is known, H_t can be thrown away)

From Observations to States

Full observability:

- Agent observes the full environment
- This is a Markov Decision Process (MDP)
- e.g Chess

Partial observability:

- Agent observes part of the environment
- This is a Partially Observable Markov Decision Process (POMDP)
- e.g. Poker

Reinforcement Learning

Components of an RL agent

Components of an RL agent

Policy:

- It is the agent's behaviour
- A policy is a map from states (S) to actions (A)
- It can be:
  - Deterministic: $a = \pi(s)$
  - Probabilistic: $\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$

Components of an RL agent

Model:

- Agent's representation of the environment
- Predicts what the environment will do next
- P predicts the next state

$$P^a_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

- R predicts the next (immediate) reward

$$R^a_s = \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a)$$

Components of an RL agent

Value function:

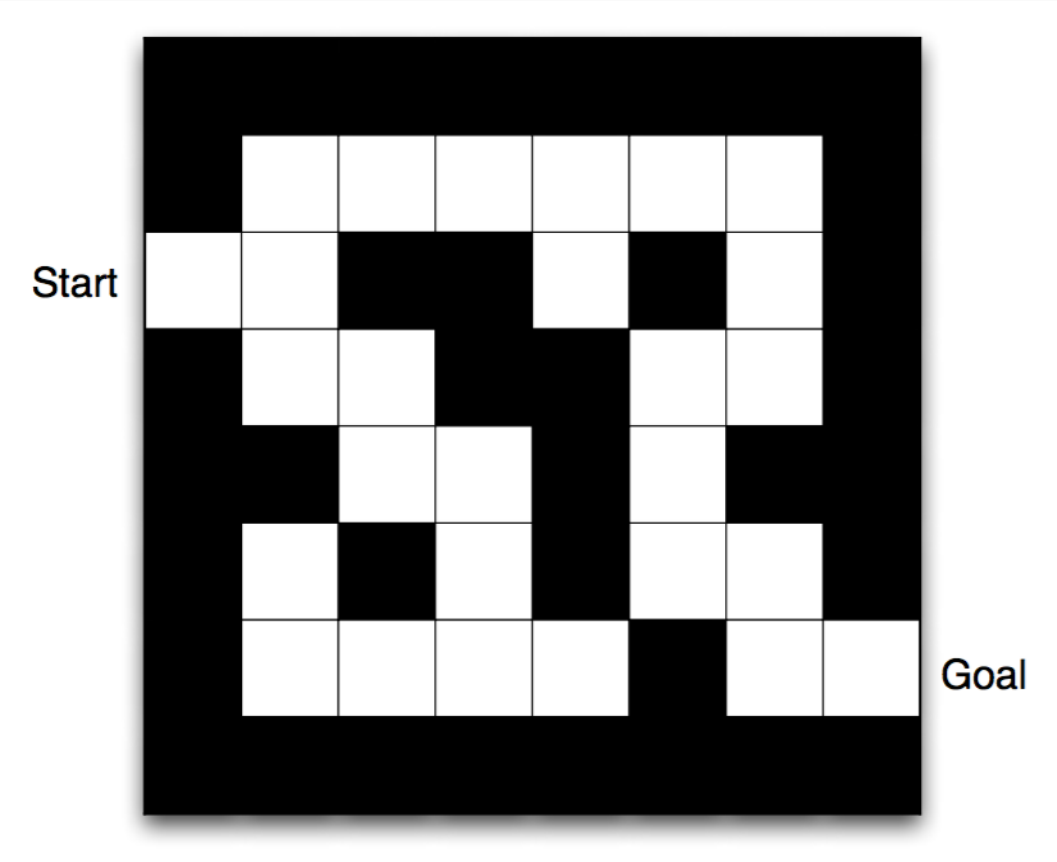
- It is a quantitative measure of how good is a state or an action
- Prediction of future reward
- Allows action selection

$$V_\pi(s) = \mathbb{E}_\pi(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s)$$

Reinforcement Learning

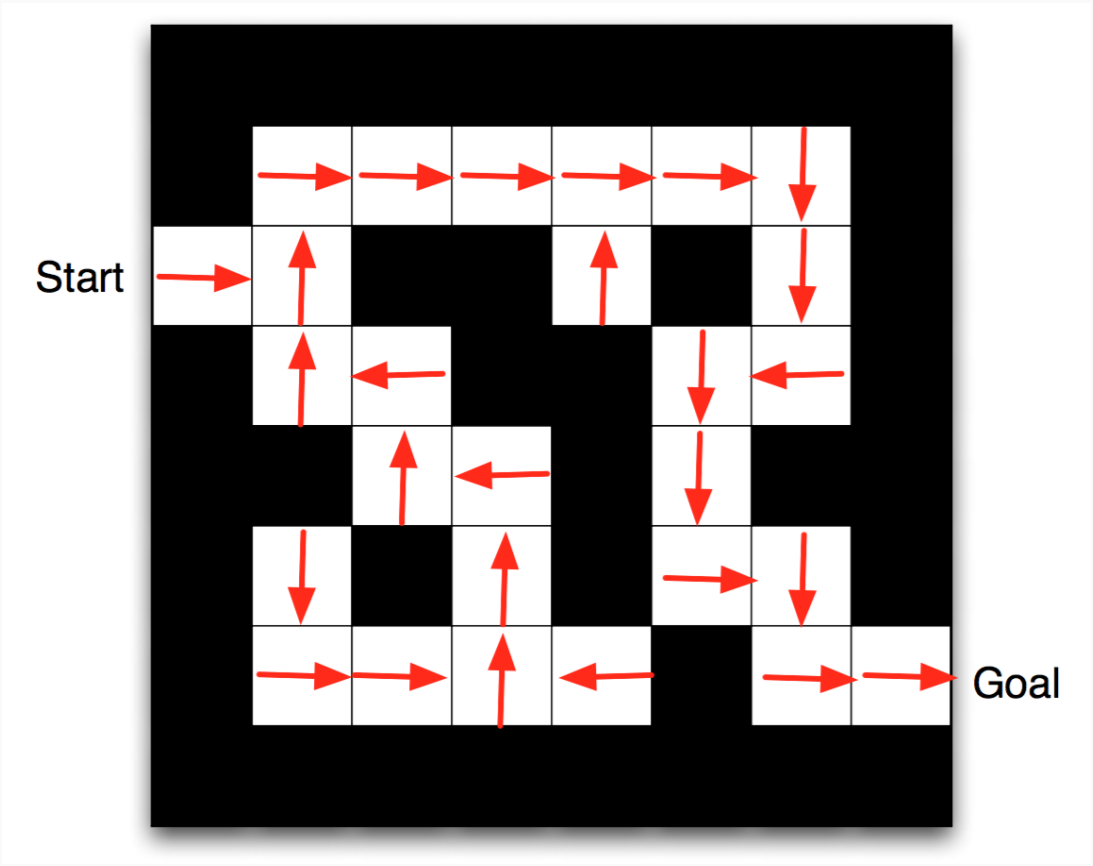
Example: Maze

Domain Definition



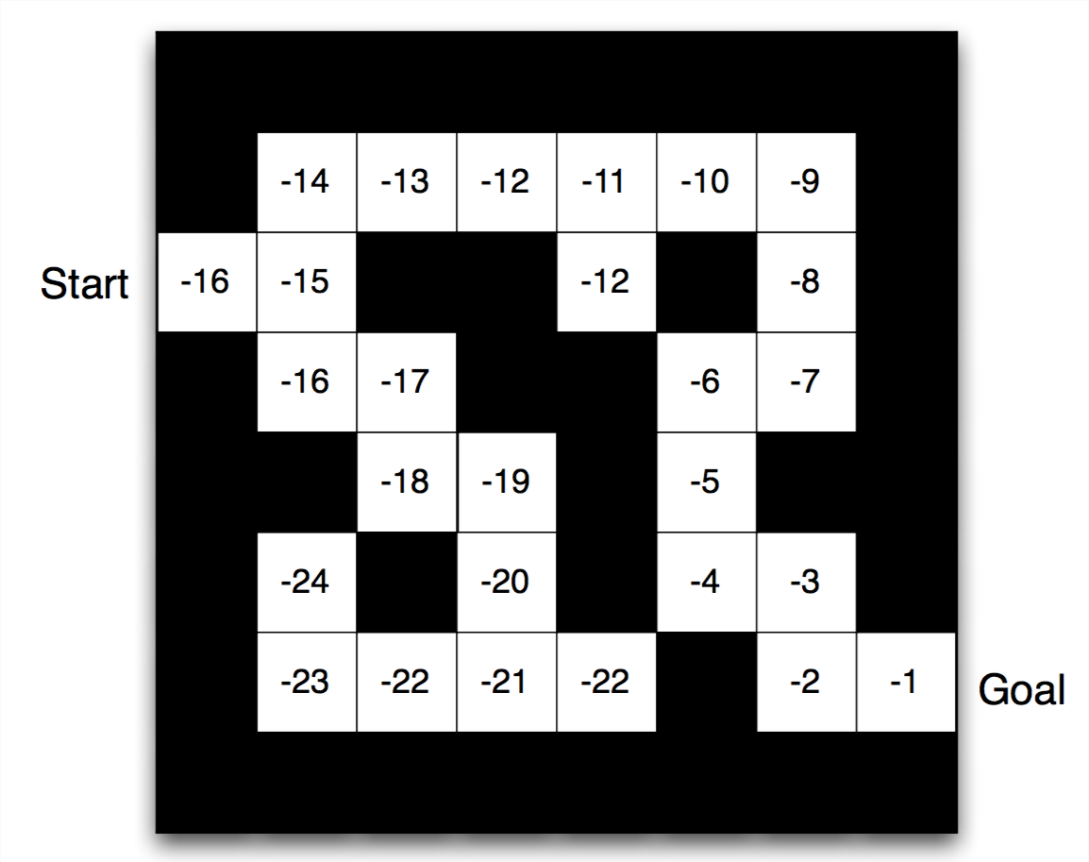
- Reward: -1 per time step (≡ state)
- Actions: N, S, W, E
- State: agent's location
- Game (*episode*) ends when the agent reaches the goal.

Policy



- Arrows represents the state transitions.
- Policy $\pi(s)$ is deterministic, for every state s

Value Function



- Numbers are the values $v(s)$ for each state s
- For each state (position) in the grid, $v(s)$ indicates the expected reward from s, **following** policy π (see last slide)
- Tells the number of moves until the goal (because of the reward choice)

Categorizing RL

Policy and/or Value Function:

Policy	Value Function	RL Category
No	Yes	Value Based
Yes	No	Policy Based
Yes	Yes	Actor Critic

With or without Model:

Policy	Value Function	Model	RL Category	Problem
Policy and/or Value Function		Yes	Model Based	Planning
Policy and/or Value Function		No	Model Free	Learning

Markov Decision Processes

Markov Decision Process

- A Markov Decision Process (MDP) formally describes an environment for RL
- The environment is fully observable
- Almost all RL problems can be formalized as MDP

Remember the Markov property:

Definition (Markov State)

A state S_t is **Markov** if and only if:

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, \ldots, S_t)$$

We are going to build up until the MDP definition.

Markov Process (or Markov Chain)

A Markov Process (or Markov Chain) is a memoryless random process (a sequence of random states $S_1, S_2, \ldots$ with the Markov property).

Definition (Markov Process)

A Markov Process (or Markov Chain) is a tuple $< S, P >$:

- S is a finite set of states
- P is a state transition probability matrix

$$P_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s)$$

Markov Process (or Markov Chain)

A Markov Process (or Markov Chain) is a memoryless random process (a sequence of random states $S_1, S_2, \ldots$ with the Markov property).

Definition (Markov Process)

A Markov Process (or Markov Chain) is a tuple $< S, P >$:

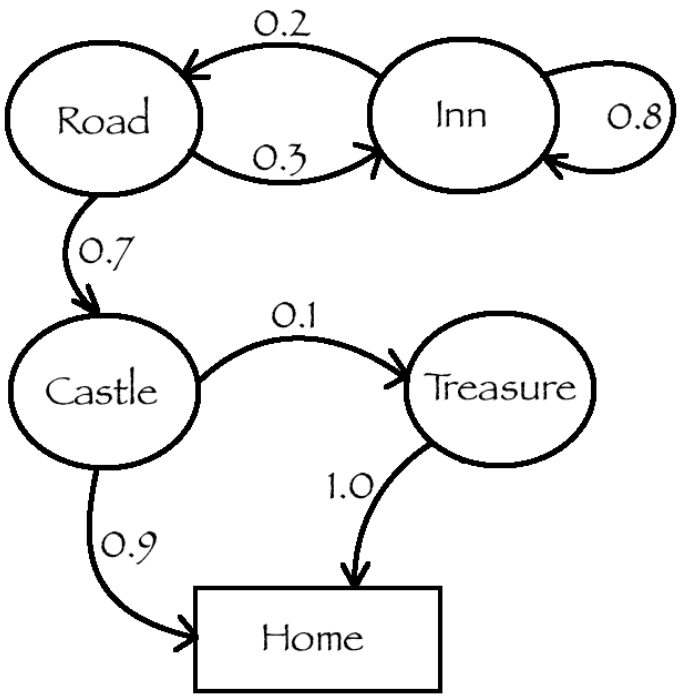
- S is a finite set of states
- P is a state transition probability matrix

$$P_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s)$$

Markov Decision Processes

Example: Rogue

Rogue Markov Chain



Probability Transition Matrix $P_{ss'}$:

To (s'):	R	I	C	T	H
From $s = R$		0.3	0.7		
From $s = I$	0.2	0.8			
From $s = C$					
From $s = T$				0.1	0.9
From $s = H$					1.0

Valid Markov Chains (episodes) starting from R are:
RIIIRCH, *RCTH*, *RIRCTH*, ...

Markov Reward Process

A Markov Reward Process is a Markov Chain with values.

Definition (Markov Reward Process)

A Markov Reward Process is a tuple $< S, P, R, \gamma >$:

- S is a finite set of states
- P is a state transition probability matrix

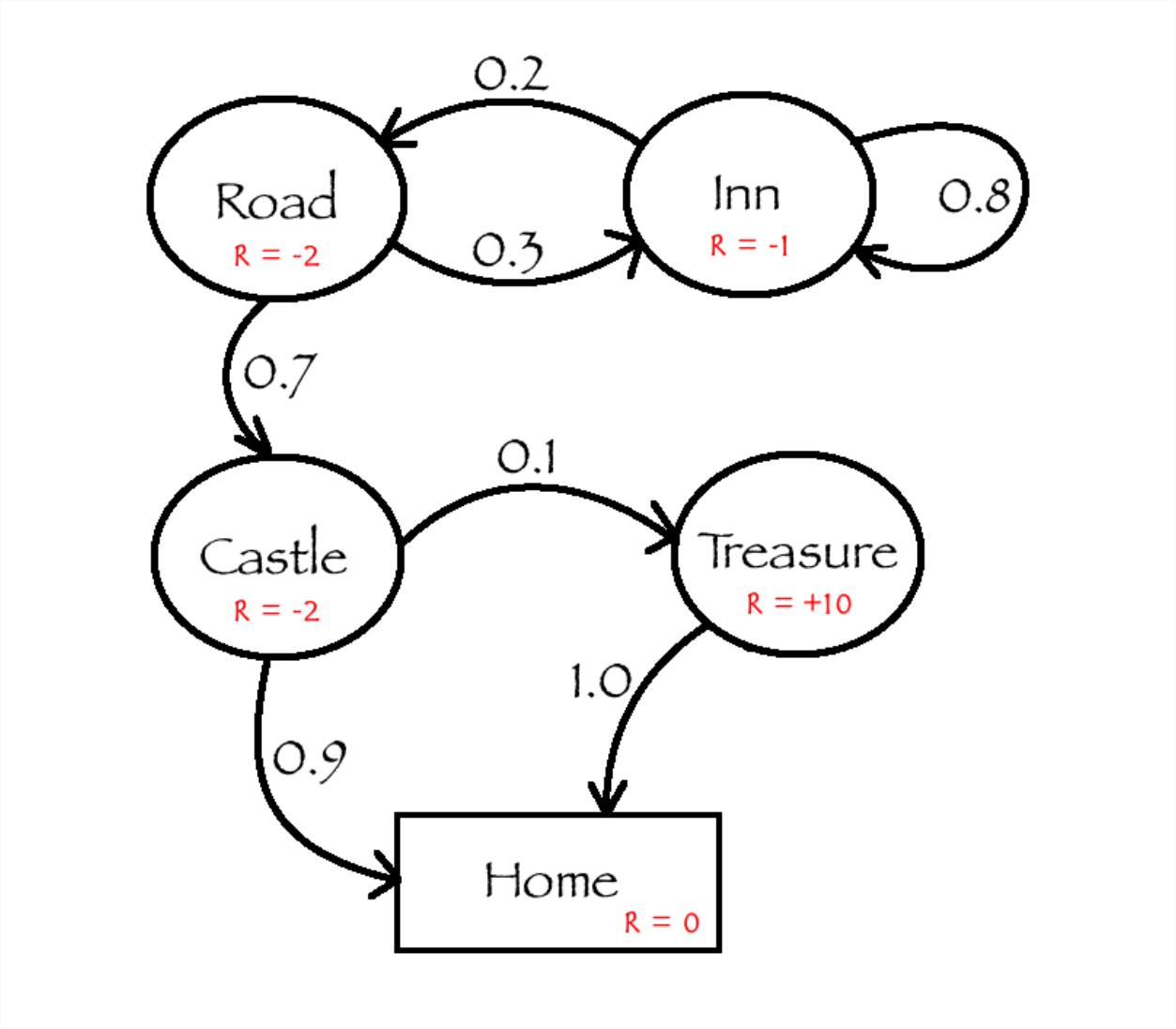
$$P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

- R is a reward function

$$R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$

- γ is a discount factor, $\gamma \in [0, 1]$

Rogue Markov Reward Process



Return

Now that we have rewards, we can define *return*:

Definition (Return)

The return G_t is the total (discounted) reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The discount $\gamma \in [0, 1]$ represents the value of future rewards:

- $\gamma = 0$: ”myopic” evaluation, only considering immediate rewards.
- $\gamma = 1$: ”far-sighted” evaluation, considering all rewards in the future.

Q? Why discount?

Why Discount?

- Uncertainty about the future.
- To put at end at G_t (mathematically convenient: no infinite returns).
- Humans/animals prefer immediate reward

Value Function

The value function $v(s)$ gives the long-term value of the state s.

Definition (Value Function)

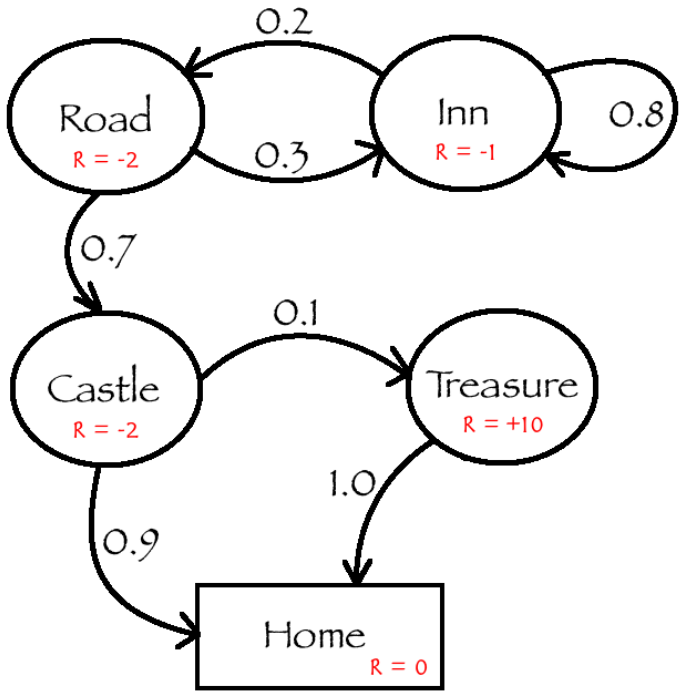
The *state-value* function $v(s)$ of an MRP is the expected return starting from state s:

$$v(s) = \mathbb{E}(G_t \mid S_t = s)$$

It is the expected return (**average** of discounted rewards) an agent would obtain following nature (the rules that govern the environment, P).

Value Function in the Rogue Example

What is the value of $v(R)$ considering only the Markov Chains *RIIIRCH*, *RCTH* and *RIRCTH*, with $\gamma = \frac{1}{2}$?



The return of each chain would be, using the definition of G_t:

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$$RCTH : v_1 = -2 - 2 \times (\tfrac{1}{2}) + 10 \times (\tfrac{1}{2})^2 + 0 \times (\tfrac{1}{2})^3$$
$$= -2 - 2 \times 0.5 + 10 \times 0.25 + 0 \times 0.125 = -0.5$$
$$RIRCTH : v_1 = -2 - 1 \times (\tfrac{1}{2}) - 2 \times (\tfrac{1}{2})^2 - 2 \times (\tfrac{1}{2})^3 + 10 \times (\tfrac{1}{2})^4 + 0 \times (\tfrac{1}{2})^5$$
$$= -2 - 1 \times 0.5 - 2 \times 0.25 - 2 \times 0.125 + 10 \times 0.0625 + 0 \times 0.03125 = -2.625$$

$$RIIIRCH : v_1 = -2 - 1 \times (\tfrac{1}{2}) - 1 \times (\tfrac{1}{2})^2 - 1 \times (\tfrac{1}{2})^3 - 2 \times (\tfrac{1}{2})^4 - 2 \times (\tfrac{1}{2})^5 + 0 \times (\tfrac{1}{2})^6$$
$$= -2 - 0.5 - 0.25 - 0.125 - 2 \times 0.0625 - 2 \times 0.03125 + 0 \times 0.0156 = -3.0625$$

Value Function in the Rogue Example

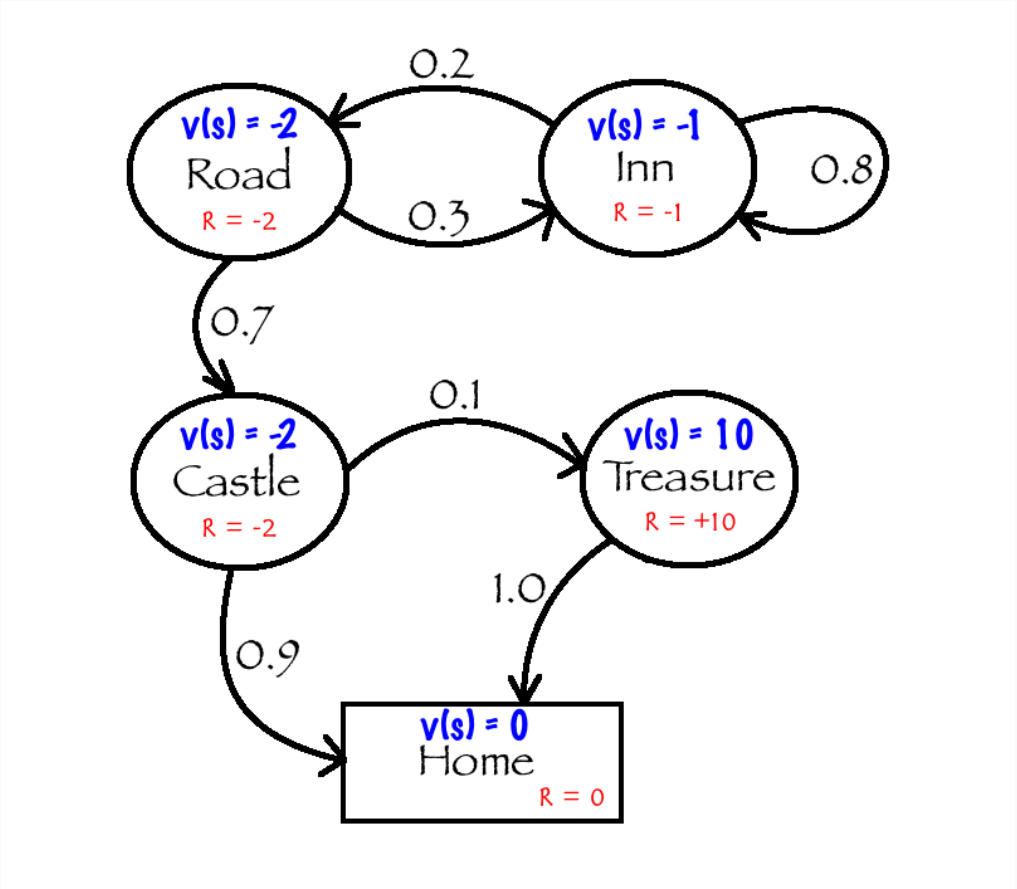


Figure 1: $v(s)$ with $\gamma = 0.0$, averaged over 10^6 returns

Value Function in the Rogue Example

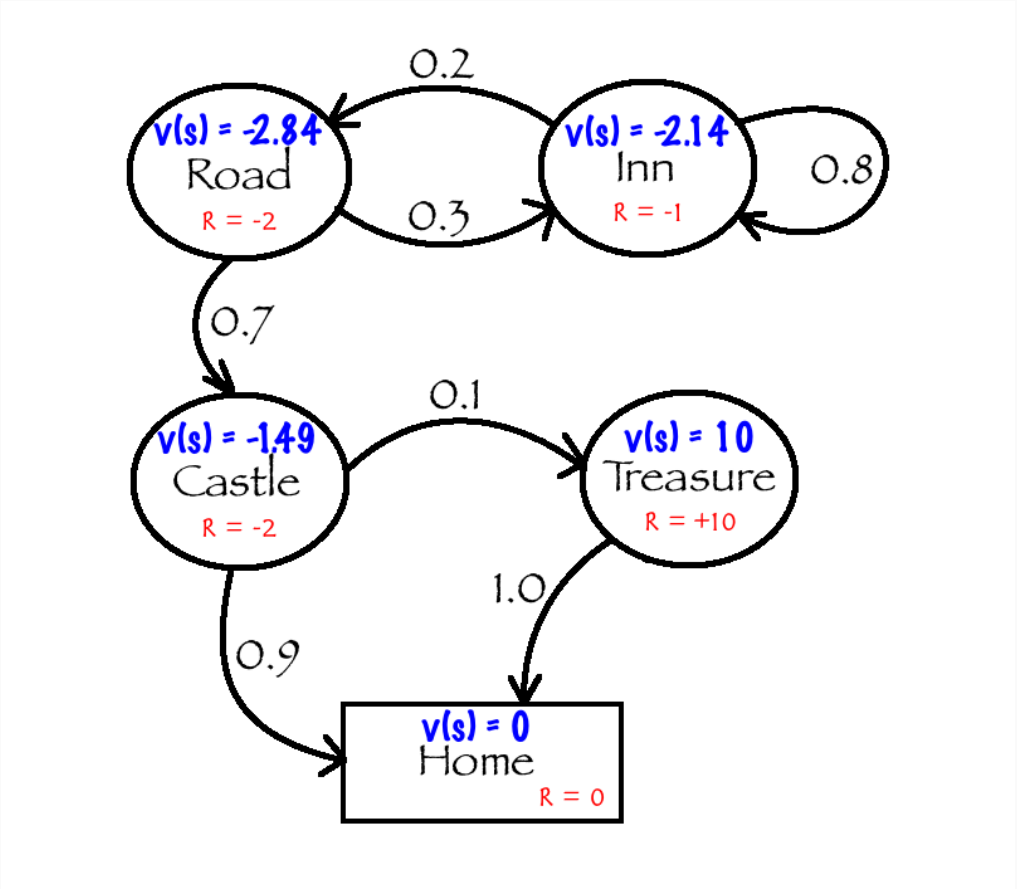


Figure 2: $v(s)$ with $\gamma = 0.5$, averaged over 10^6 returns

Value Function in the Rogue Example

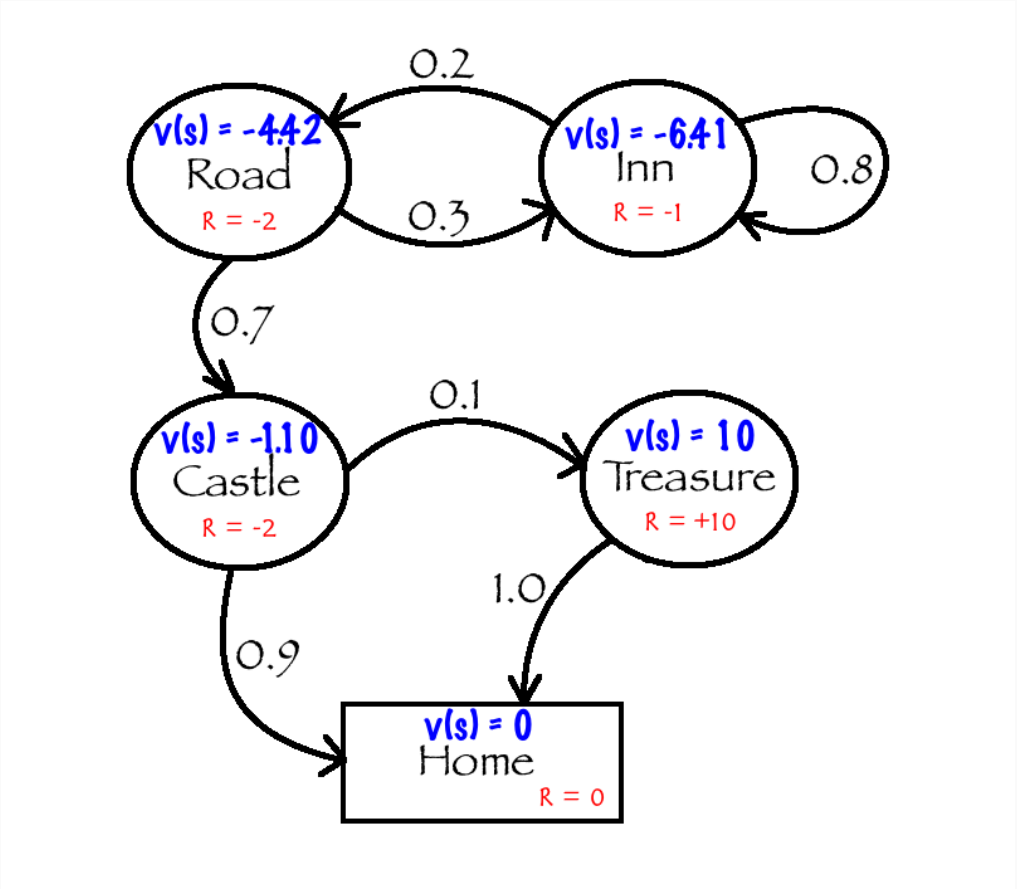


Figure 3: $v(s)$ with $\gamma = 0.9$, averaged over 10^6 returns

Value Function in the Rogue Example

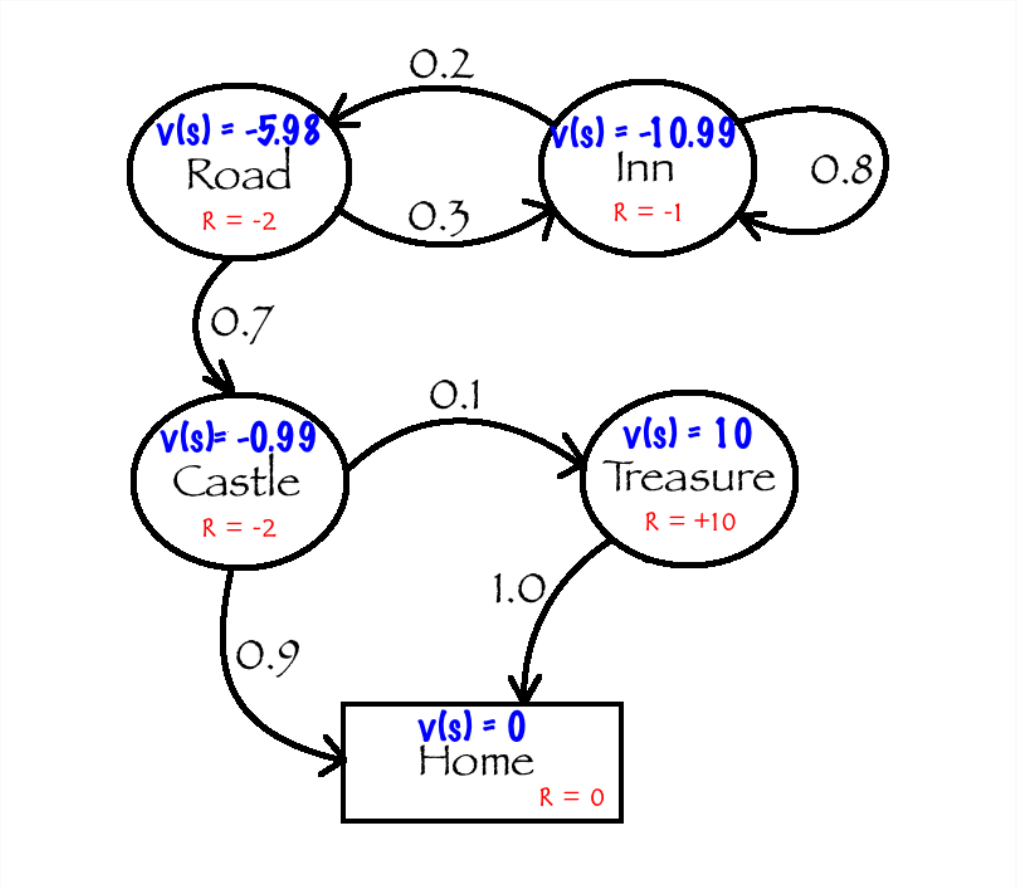


Figure 4: $v(s)$ with $\gamma = 1$, averaged over 10^6 returns

Bellman Equation for MRP

The value function can be decomposed into two parts:

- immediate reward $R_{t=1}$
- discounted value of the successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

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

Q? Do we always have to do sampling?

Bellman Equation for MRP

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

Q? Do we always have to do sampling?

- No, we can solve it analytically with Bellman Equation.
- Problem is, the MRP can be large, too large: iterative methods:
  - Dynamic Programming
  - Temporal Difference Learning
  - Monte Carlo evaluation

Markov Decision Process

A Markov Decision Process is a Markov Reward Process where the agent makes decisions according to a policy.

Definition (Markov Decision Process)

A Markov Decision Process is a tuple $< S, A, P, R, \gamma >$:

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

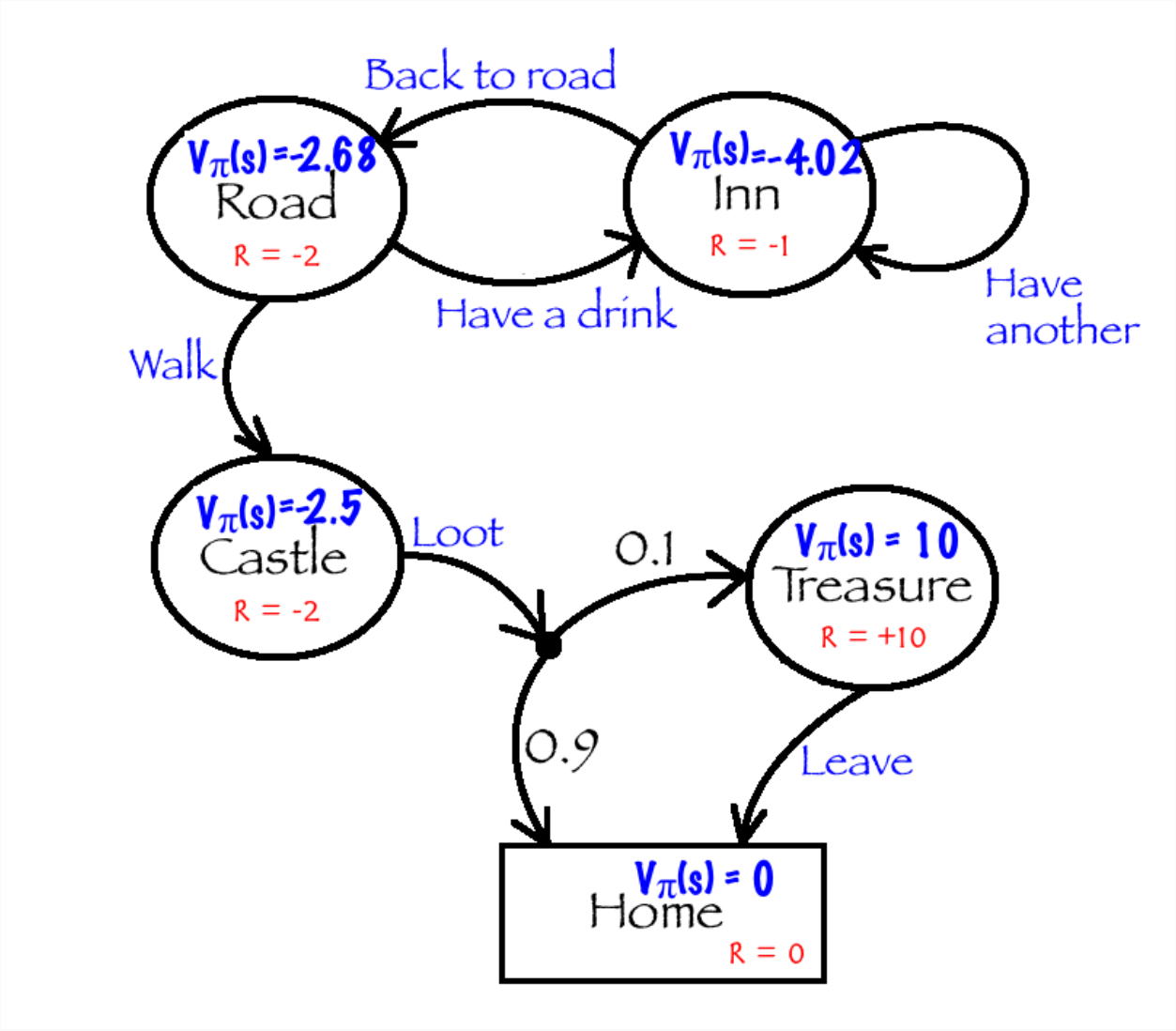
- R is a reward function

$$R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- γ is a discount factor, $\gamma \in [0, 1]$

MDP for the Rogue Example

(Note that we have two decision points: agent (via $\pi(s \mid a)$) and the environment (via $P^a_{ss'}$) at *every* transition).



Policy and Value Function

Definition (Policy)

A policy is a mapping between states and actions:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Definition (State-Value Function)

The **state-value** function $v_\pi(s)$ of an MDP is the expected return starting from state s, following policy π

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

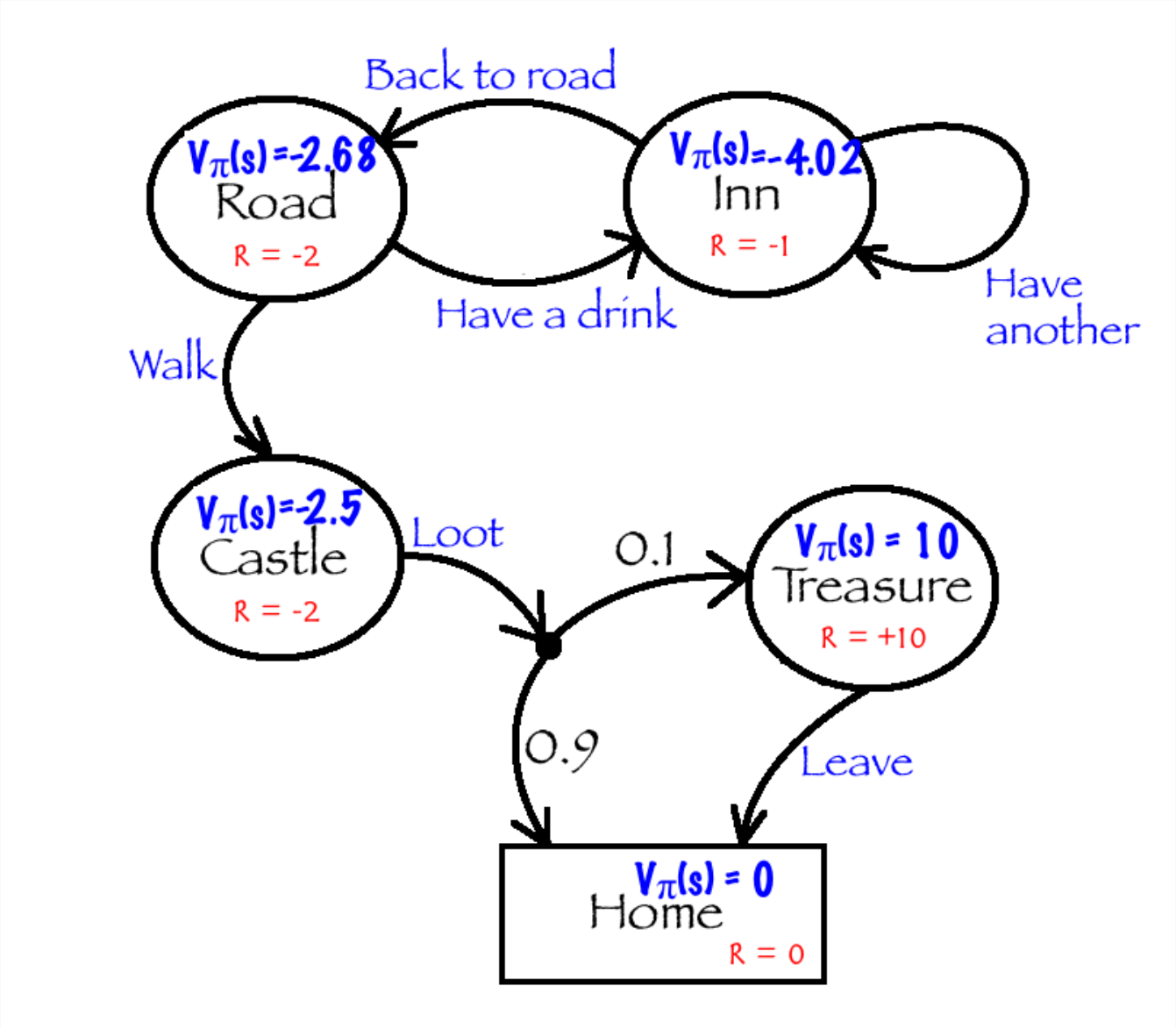
Definition (Action-Value Function)

The **action-value** function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

State-Value function in the Rogue Example

$v_\pi(s)$ with $\gamma = 0.9$, $\pi(a \mid s) = 0.5$, averaged over 10^6 returns:



Bellman Expectation Equation

The state-value function can be decomposed between immediate reward and the discounted reward from the next state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s)\Big(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_\pi(s')\Big) \end{aligned}$$

Similarly, we can do the same with the action-value function:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s'} p(s' \mid s, a) \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \end{aligned}$$

Q? We know how to estimate the value of a state (and of an action) following a policy. Now what?

Optimal Value Function

Definition (Optimal Value Function)

The **optimal** state-value function $v_*(s)$ is the maximum value over all policies.

$$v_*(s) = \max_\pi v_\pi(s)$$

The **optimal** action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

If we find $v_*(s)$, the employed policy π is then optimal. If we know how to act optimally, we have **solved** the problem.

$$v_*(s) = \max_a q_*(s, a)$$

Bellman Optimality Equations

Again, there are the equivalent Bellman **Optimality** Equations:

$$\begin{aligned} v_*(s) &= \max_a R_s^a + \mathbb{E}[\gamma v_*(s') \mid S_t = s] \\ &= \max_a R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_*(s') \end{aligned}$$

$$\begin{aligned} q^*(s, a) &= \max_a R_s^a + \mathbb{E}[\gamma q_*(s', a') \mid S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s'} p(s' \mid s, a) \max_a q_*(s', a') \end{aligned}$$

Problems

Solving these equations, we solve the MDP and the problem.

- However, these equations are non-linear. There isn't a closed form solution. How do we solve this?
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Dynamic Programming

Dynamic Programming (DP)

What is Dynamic Programming?

- Dynamic: sequential or temporal component of the problem.
- Programming: optimizing a program (policy).

Dynamic Programming (DP) solves problems by decomposing them into sub-problems that can be solved separately. DP works successfully in problems that have two properties:

- Optimal substructure:
  - Principle of optimality: the optimal solution can be decomposed into sub-problems.
  - In MDPs, this is satisfied by the *Bellman Optimality Equation*.
- Overlapping sub-problems:
  - Sub-problems may occur many times, and solutions can be cached and reused.
  - In MDPs, this is satisfied by information in the *value function* $v(s)$.

Iterative Policy Evaluation

Iterative Policy Evaluation is a DP algorithm that evaluates a given policy π (calculates the value of $v_\pi(s)$ for all states s). It performs a **prediction**: estimates how good is to follow a policy π in a given MDP.

Bellman Expectation Equation

We use the Bellman Expectation Equation. Remember:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a \mid s) \left(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_\pi(s') \right)
\end{aligned}
$$

where:

- $\pi(a \mid s)$ is the probability of taking action a in state s, under policy π.
- R_s^a is the reward obtained in state s after applying action a.
- γ is the discount factor.
- $p(s' \mid s, a)$ is the probability of transiting from state s to s' when applying action a. This is determined by nature (the environment of the MDP).

Iterative Policy Evaluation

Main idea: start with a set of values $v(s)$ (initialized at random, or $= 0$) for every $s \in S$ and apply the Bellman Expectation equation iteratively.

Iterative Policy Evaluation uses *synchronous* back-ups. This is, on each iteration, all states of the MDP are considered for updating $v(s)$ ($\forall s \in S$).

The Algorithm

The algorithm works as follows:

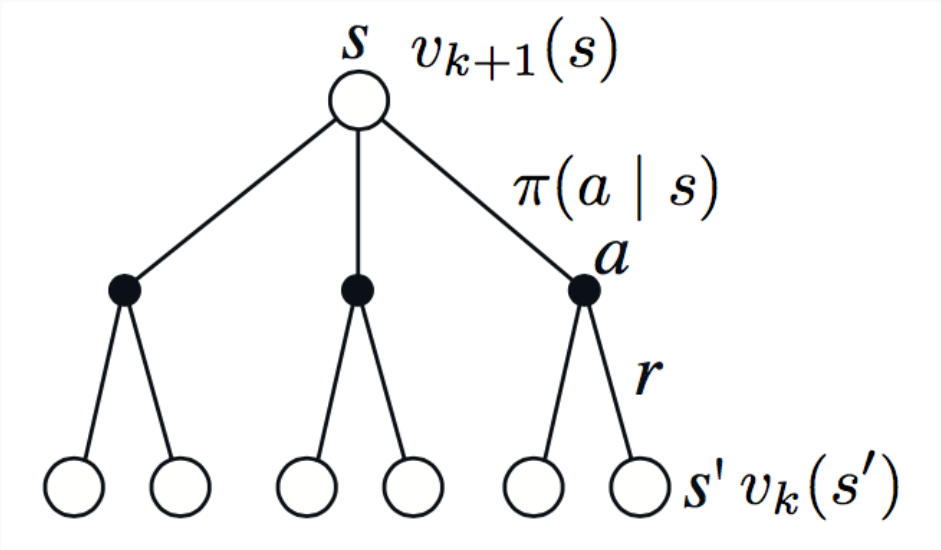
1. For all states $s \in S$, initialize $v(s)$ to a random value.
2. For each iteration k:
   2.1 For all states $s \in S$, update $v_{k+1}(s)$ from $v_k(s')$, where s' are all successors of s, using Bellman Expectation Equation:

$$v_{k+1}(s) = \sum_a \pi(a \mid s)\left(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_k(s')\right)$$

Convergence to the *true value* of v_π is guaranteed, as long as $\gamma < 1$, for all states s under policy π when $k \to \infty$.

Iterative Policy Evaluation

Iterative Policy Evaluation performs a **full backup**: replaces the old value of $v(s)$ with the new value obtained from the Bellman Expectation Equation. This is moving one step further to the future.

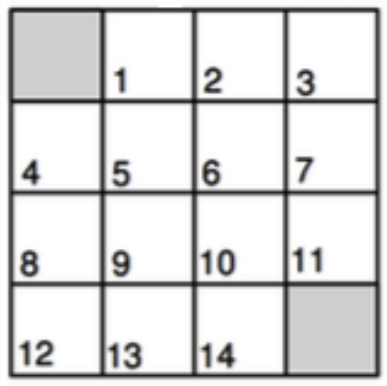


$$v_{k+1}(s) = \sum_a \pi(a \mid s) \left(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_k(s') \right)$$

Dynamic Programming

The Small Gridworld

Example: the Small Gridworld



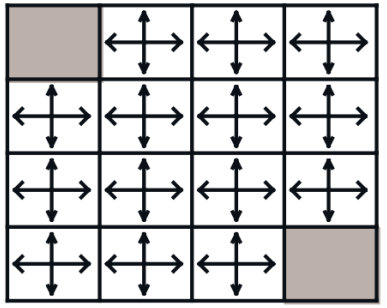
- States:
  - Non-terminal: $1, 2, \ldots, 14$.
  - Terminal: Grey cells.
- Actions:
  - Available: *Up*, *Down*, *Left*, *Right*.
  - Actions that would take the agent off grid, leave the state unchanged.
- Reward is -1 when exiting all states.
- Undiscounted task ($\gamma = 1$).
- Policy:
  - Equi-probable random policy for all $s \in S$.
  - $\pi(a \mid s) = \frac{1}{4}$
  - **We are evaluating the random policy.**
  - **Calculate** $v_\pi(s)$ for all

Iterative Policy Evaluation for the Small Gridworld

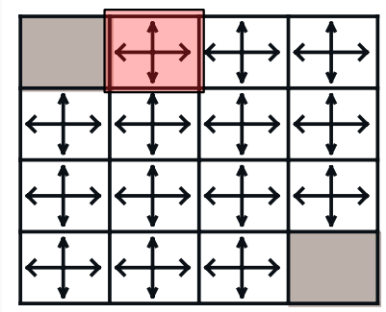
- $k = 0$; For all states $s \in S$, initialize $v(s)$ to 0 or a random value:

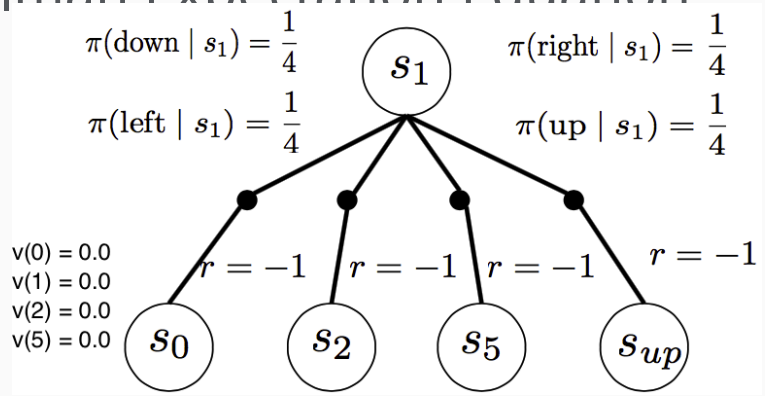
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Random policy:



- $k = 1$; For all states $s \in S$, update $v_{k+1}(s)$ from $v_k(s')$, where s' are all successors of s, using Bellman Expectation Equation





$$v_1(s_1) = \sum_a \pi(a \mid s)\left(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_k(s')\right)$$

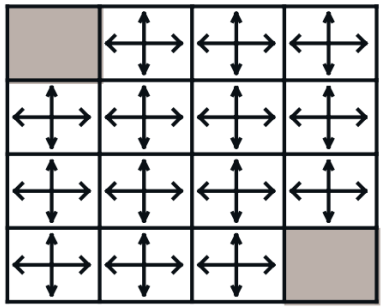
$$= \frac{1}{4}(-1+0) + \frac{1}{4}(-1+0) + \frac{1}{4}(-1+0) + \frac{1}{4}(-1+0) = -1$$

Iterative Policy Evaluation for the Small Gridworld

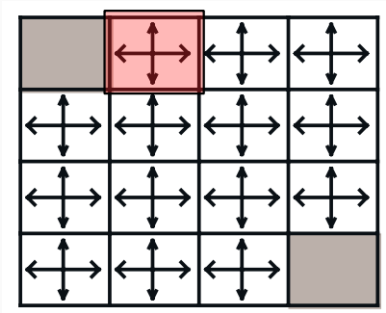
- $k = 1$:

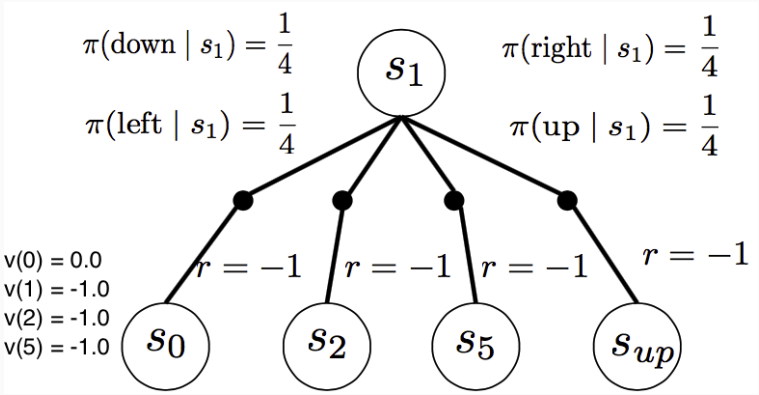
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Random policy:



- $k = 2$:





$$v_1(s_1) = \sum_a \pi(a \mid s)\left(R_s^a + \gamma \sum_{s'} p(s' \mid s, a) v_k(s')\right)$$

$$= \frac{1}{4}(-1 + 0) + \frac{1}{4}(-1 - 1) + \frac{1}{4}(-1 - 1) + \frac{1}{4}(-1 - 1) = -1.75$$

Iterative Policy Evaluation for the Small Gridworld

- For all iterations

$k = 0$

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

$k = 1$

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

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Q? When we reach $k = 10$, is it sensible to keep using a random policy? Can't we do better?

Policy Improvement

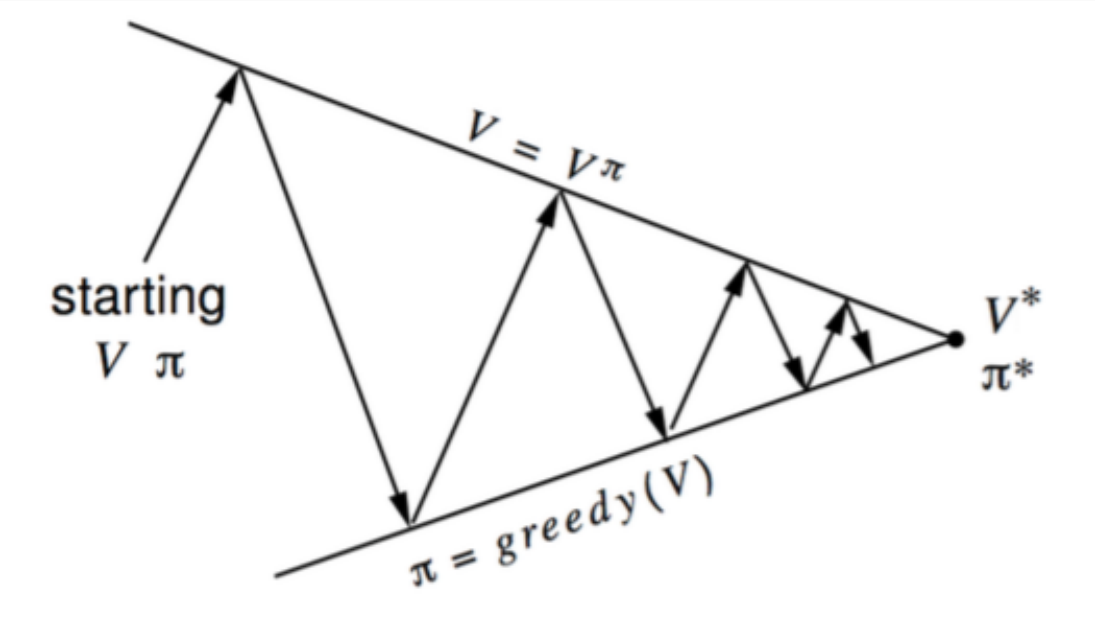
What if, instead of always using the random policy, after the first iteration we modify our policy using the values $v(s)$ we are calculating?

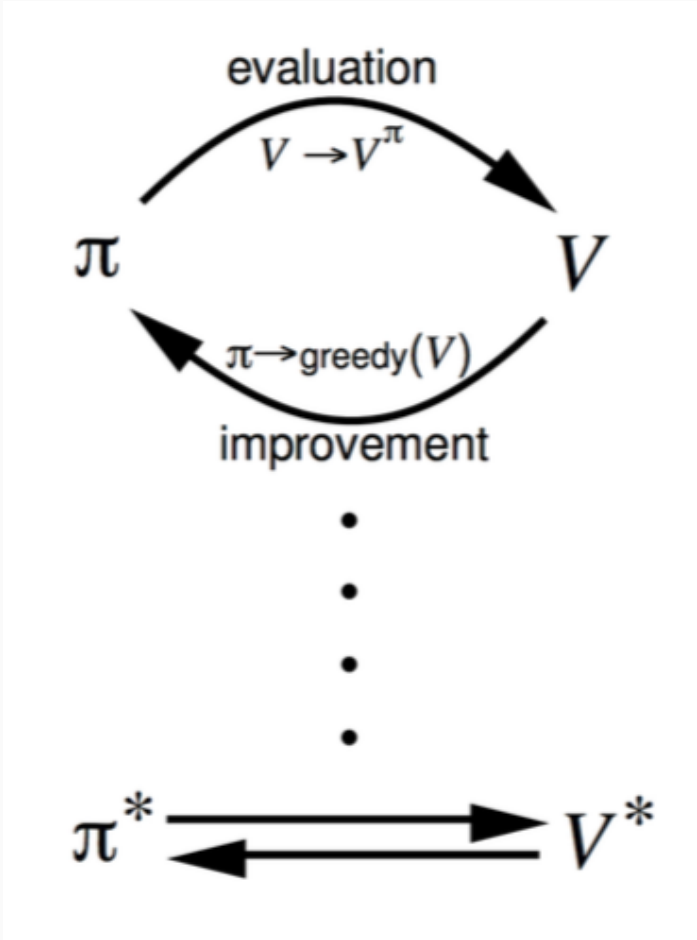
We can improve our policy by selecting the action that leads to the highest $v(s')$. This can be done with a **greedy** policy:

$$\pi'(s) = argmax_a q_\pi(s, a)$$

Policy improvement is the process of making a new policy that improves on an original policy, by acting greedily with respect to v_π.

Policy Iteration



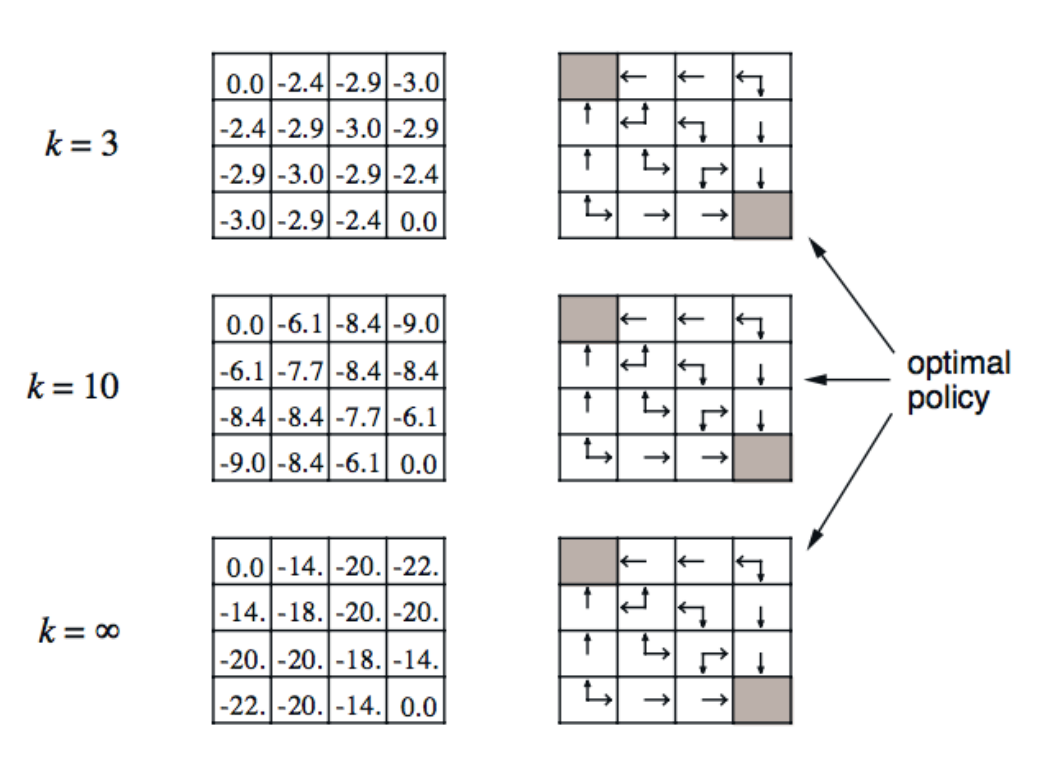


How to obtain π^*?

- **Policy evaluation:** Estimate v_π
  e.g. Iterative policy evaluation
- **Policy improvement:** Generate $\pi' \geqslant \pi$
  e.g. Greedy policy improvement

Modified Policy Iteration

Does policy evaluation need to converge to v_π?

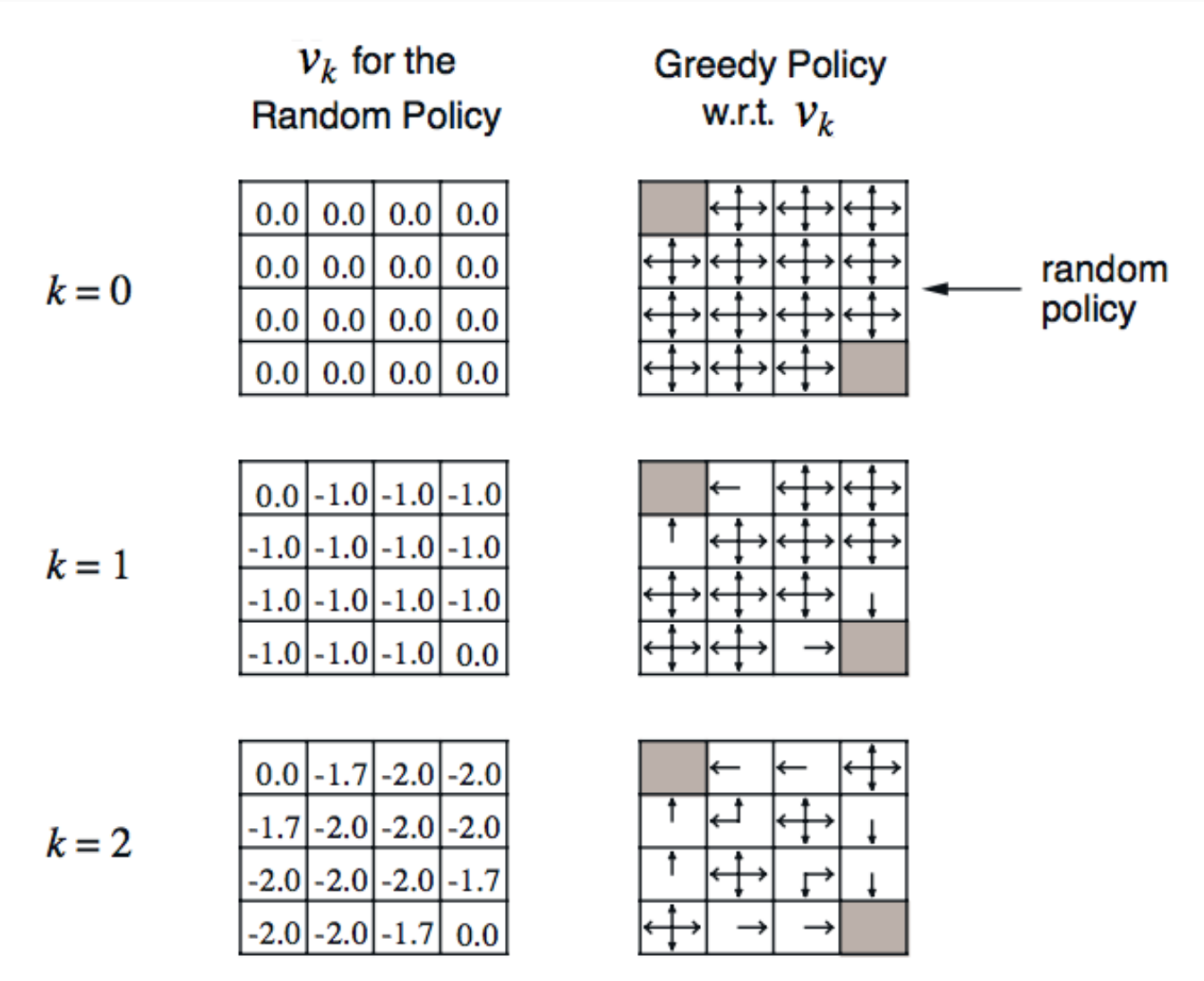


Can't we stop after k iterations of iterative policy evaluation?

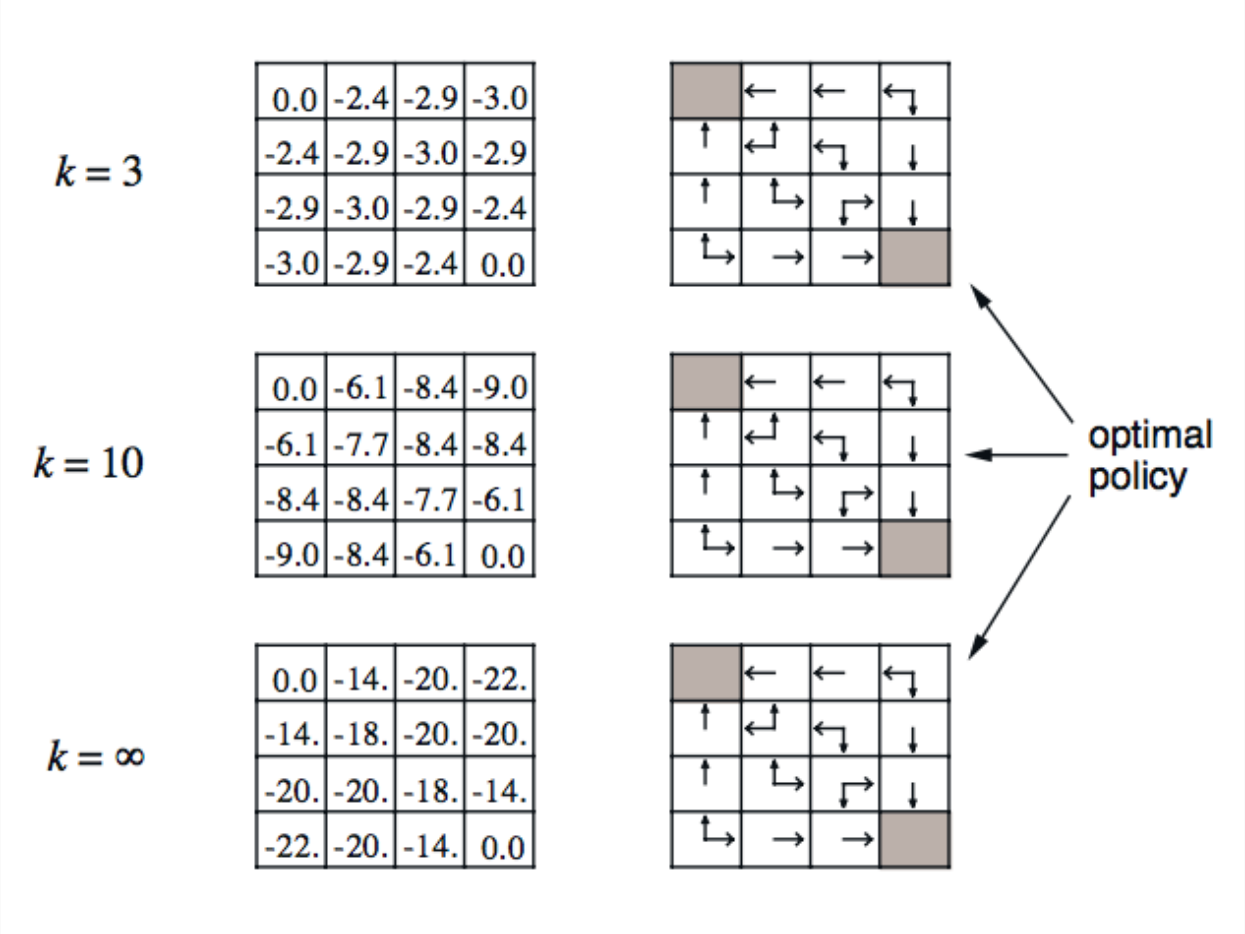
- $k = 3$ was okay for the small gridworld!

Policy Improvement for the Small Gridworld

How does **Policy Improvement** work? Let's see how does the greedy policy looks like at each step of Policy Evaluation:



Policy Improvement for the Small Gridworld



In the Small Gridworld example, $\pi' = \pi^*$ in $k = 3$, but in general many more iterations on these two steps are needed. However, policy iteration **always** converges to π^*.

Policy Improvement

Q? When do we stop? When the improvements in the policy stop.

We are improving our current policy by acting greedily:

$$\pi'(s) = argmax_a q_\pi(s, a)$$

By definition, this implies that following our new policy π' from s is never worse than following our previous policy π from s:

$$q_\pi(s, \pi'(s)) \geqslant q_\pi(s, \pi(s))$$

Applying the definition of acting greedily:

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

and knowing that these two are equivalent:

$$q_\pi(s, \pi(s)) = v_\pi(s)$$

we can conclude that:

$$max_{a \in A} q_\pi(s, a) \geqslant v_\pi(s)$$

Policy Improvement

$$max_{a \in A} q_\pi(s, a) \geqslant v_\pi(s)$$

(this means we are improving - or at least not getting worse!)

Convergence: (when do we stop) if for all states $s \in S$, there is no improvement:

- $\pi'(s) = \pi(s)$, same actions are picked from every state before and after the iteration.
- $\rightarrow q_\pi(s, \pi'(s)) = q_\pi(s, \pi(s))$
- $\rightarrow max_{a \in A} q_\pi(s, a) = v_\pi(s)$

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

- Bellman Optimality Equation is satisfied, so $v_\pi(s) = v_*(s)$
- $\rightarrow \pi$ is an optimal policy.

Value Iteration

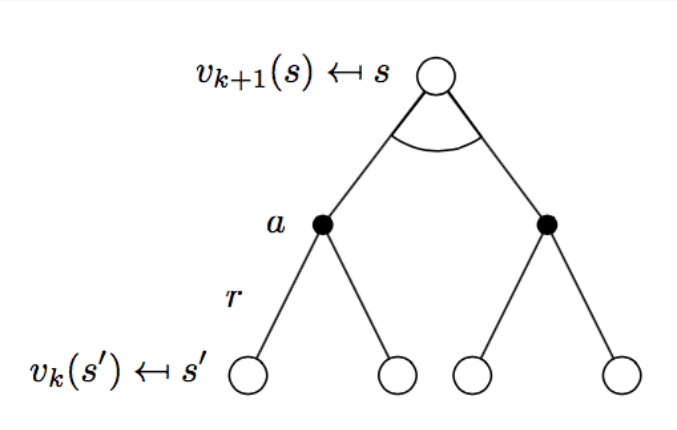
Why not update policy **every** iteration ($k = 1$)? This is **Value Iteration**.

We **always** use: $\pi(s) = argmax_a q_\pi(s, a)$

Monte Carlo Methods

Using Samples

Dynamic Programming uses full backups. This is, for each state, all successor states and actions are considered, assuming full knowledge of the MDP (this is, assuming we know $P^a_{ss'}$):



This is not effective for large sized problems, as it suffers Bellman's *curse of dimensionality*: the number of states $s \in S$ grows exponentially with the number of state parameters.

Model-Free Prediction

Instead, **Monte Carlo** (MC) methods use sampling: sample sequences of states, actions and rewards from actual or simulated interaction with an environment. MC methods require only experience, they **do not** assume complete knowledge of the MDP (we **don't** necessarily know $P^a_{ss'}$).

This is **model-free prediction**.

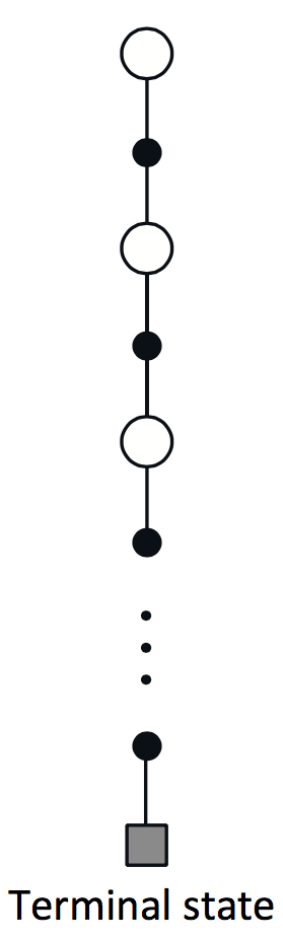
Monte Carlo Policy Evaluation

In MC, we learn the value of v_π from episodes of experience (interaction of the agent with the environment) using policy π. The return of a single sample is, as seen before:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

And the value function still is:

$$v_\pi(s) = \mathbb{E}_\pi(G_t \mid S_t = s)$$



But Note

But note:

- In DP, we **compute** the value function as the expected return, with full knowledge of the MDP.
- In MC, we **learn** the value function, as empirical mean, sampling from the MDP. MC uses the *average of returns*, which is the most obvious/simplest way. As more experience is observed (more returns are calculated), the average should converge to the expected value.

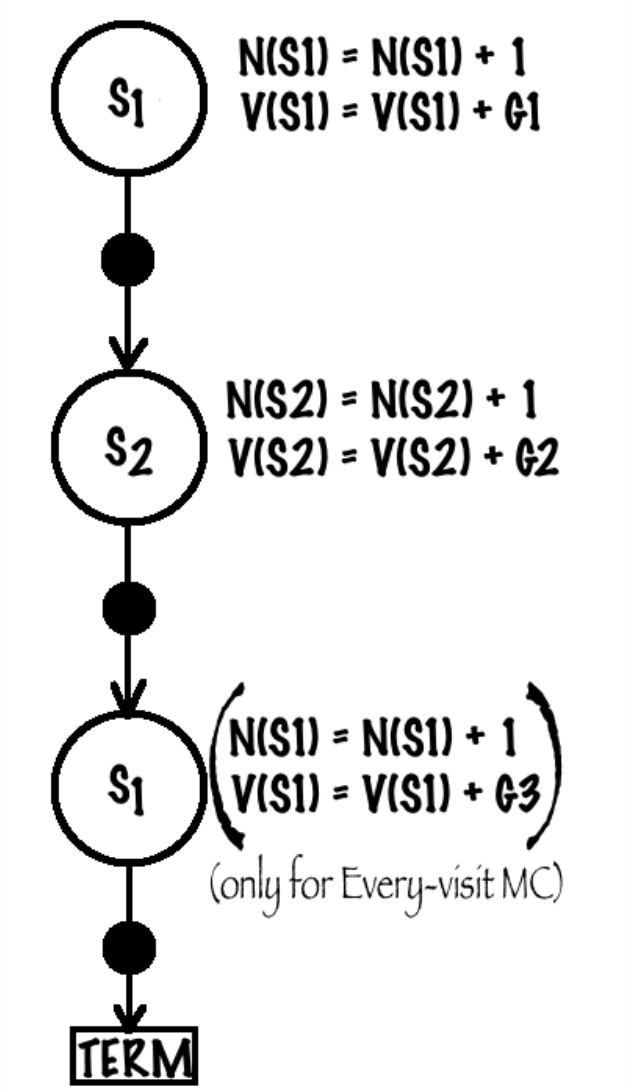
MC only works for episodic tasks (always terminate).

Monte Carlo Policy Evaluation

Note that we need to wait until the end of the episode to be able to update $N(s), v(s)$. There are two variants:

- **First-Visit** MC Policy Evaluation: Update $N(s)$ and $v(s)$ only from the **first** time s was found in an episode.
- **Every-Visit** MC Policy Evaluation: Update $N(s)$ and $v(s)$ **every** time s is found in the episode.

First versus Every-Visit MC



The value of the Return G_t is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

For this example:

$$G_1 = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}$$

$$G_2 = R_{t+2} + \gamma R_{t+3}$$

$$G_3 = R_{t+3}$$

Incremental Monte Carlo Updates

An alternative way of computing a mean, incrementally (μ_{k-1} : previous mean):

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) = \frac{1}{k}(x_k + (k-1)\mu_{k-1})$$
$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Therefore, we can update $v(s)$ incrementally after episode. For each state s_t with return G_t:

$$N(s_t) \leftarrow N(s_t) + 1$$

$$v(s_t) \leftarrow v(s_t) + \frac{1}{N(s_t)}(G_t - v(s_t))$$

$(G_t - v(s_t))$ is a type of error (what happened, minus what I expect to happen!).

Incremental Monte Carlo Update

$$N(s_t) \leftarrow N(s_t) + 1$$

$$v(s_t) \leftarrow v(s_t) + \frac{1}{N(s_t)}(G_t - v(s_t))$$

Also (for non-stationary problems), we may not want to remember all episodes (this happens when factoring by $\frac{1}{N(s_t)}$).

Forgetting Old Things

Instead, we can forget old episodes by factoring by a constant α (non-stationary environments).

Definition (Incremental MC: Constant-α MC method)

$$v(s_t) \leftarrow v(s_t) + \alpha(G_t - v(s_t))$$

We are correcting our estimate of $v(s_t)$ by moving the value a *little bit* (α) in the direction of the error.

Summary

The take-away points of this lecture are:

- Components of the RL problem: States, Actions, observations, Rewards.
- Policy: mapping from states to actions. Stochastic or eterministic.
- (State-/Action) Value Function: how good a state/action is.
- Goal: maximizing total future reward.
- Sampling an MRP.
- How to act optimally.

Acknowledgements

Most of the materials for this course are based on:

- Prof. David Silver's course on Reinforcement Learning:
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Prof. Sanaz Mostaghim's course on Computational Intelligence in Games:
  http://is.cs.ovgu.de/Teaching/SS+2015/Computational+Intelligence+in+Games+_+SS+2015-p-196.html
- Prof. Julian Togelius (et al.) book on Procedural Content Generation:
  http://pcgbook.com/
- Alex Champandard's web/course on Game AI:
  http://aigamedev.com/
  http://courses.nucl.ai/