# Reinforcement Learning

## CS 229: Machine Learning

**Emaad Ahmed Manzoor**

April 28, 2014

# Introduction

# Reinforcement Learning

- Learning by interacting with the environment

# A Simple Example: k-armed bandit

Target: earn money AMAP

Task: decide which lever to pull

Agent: you

State: single state (one slot machine)

Action: choose one lever to pull

Reward: money given by the machine after one action (immediate reward after a single action)


Supervised learning? Need a teacher to tell you which one?

# Another Example: a machine to play chess

Target: win the game

Task: decide a sequence of moves

Agent: player

State: state of the board(environment)

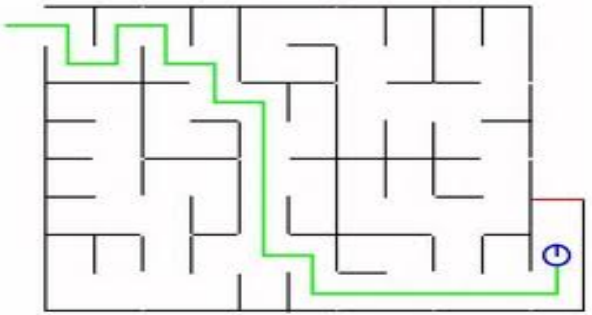
Action: decide a legal move

Reward: win or lose

(when game is over)

Is there a supervised learner who can tell you how to move?

# Another Example: a robot in a maze



Target: reach the exit

Task: decide a sequence of movements

Agent: robot

State: position of the robot in the maze

Action: choose one of the 4 directions without hitting the walls

Reward: length of  trajectories (playing time)

(when the robot reaches the exit)

# Reinforcement Learning

- Learning by interacting with the environment
- Delayed rewards

    **Credit attribution**

# Reinforcement Learning

- Learning by interacting with the environment
- Delayed rewards

  **Credit attribution**

- Explore and exploit

# VS Supervised Learning

- Supervision provides training samples with the **right answer**

    **Instructive** feedback, independent of the action taken

    Difficult to provide in many cases

- Reinforcement provides only **rewards** based on states and actions

    **Evaluative** feedback, dependant on the action taken

    The agent selects actions over time to maximise its total reward

# RL vs Supervised Learning

| Reinforcement Learning | Supervised Learning |
| --- | --- |
| Learning with a critic | Learning with a teacher |
| How well we have been doing in the past | What's good or bad |
| Learn to generate an internal value for the intermediate states or actions (how good they are) | Learn to minimize general risk on predicting |
| Exploration and Exploitation | Over-fitting and Under-fitting |

# Successful examples of RL

- **Robocup Soccer Teams Stone & Veloso, Reidmiller etal.**
  - World's best player of simulated soccer, 1999; Runner--up 2000
- **Dynamic Channel Assignment Singh & Bertsekas, Nie &Haykin**
  - World's best assigner of radio channels to mobile telephone calls
- **Elevator Control Crites & Barto**
  - (Probably) world's best down---peak elevator controller
- **Many Robots**
  - navigation, bi--pedal walking, grasping, switching between skills...
- **TD---Gammonand Jellyfish Tesauro, Dahl**
  - World's best backgammon player

# Elements of Reinforcement Learning

- Model of the environment
- Policy
- Reward function
- Value function

# Elements of Reinforcement Learning

**- Model of the environment**

States

State transitions: stochastic or deterministic

- Policy

- Reward function

- Value function

# Elements of Reinforcement Learning

- Model of the environment
- **Policy**
    - Maps states to actions to be taken in those states
    - Lookup table or function
    - Stochastic or deterministic
- Reward function
- Value function

# Elements of Reinforcement Learning

- Model of the environment
- Policy
- **Reward function**

    *Rewards* are numbers indicating the desirability of a state

    Provided by the environment

    May be stochastic or deterministic
- Value function

# Elements of Reinforcement Learning

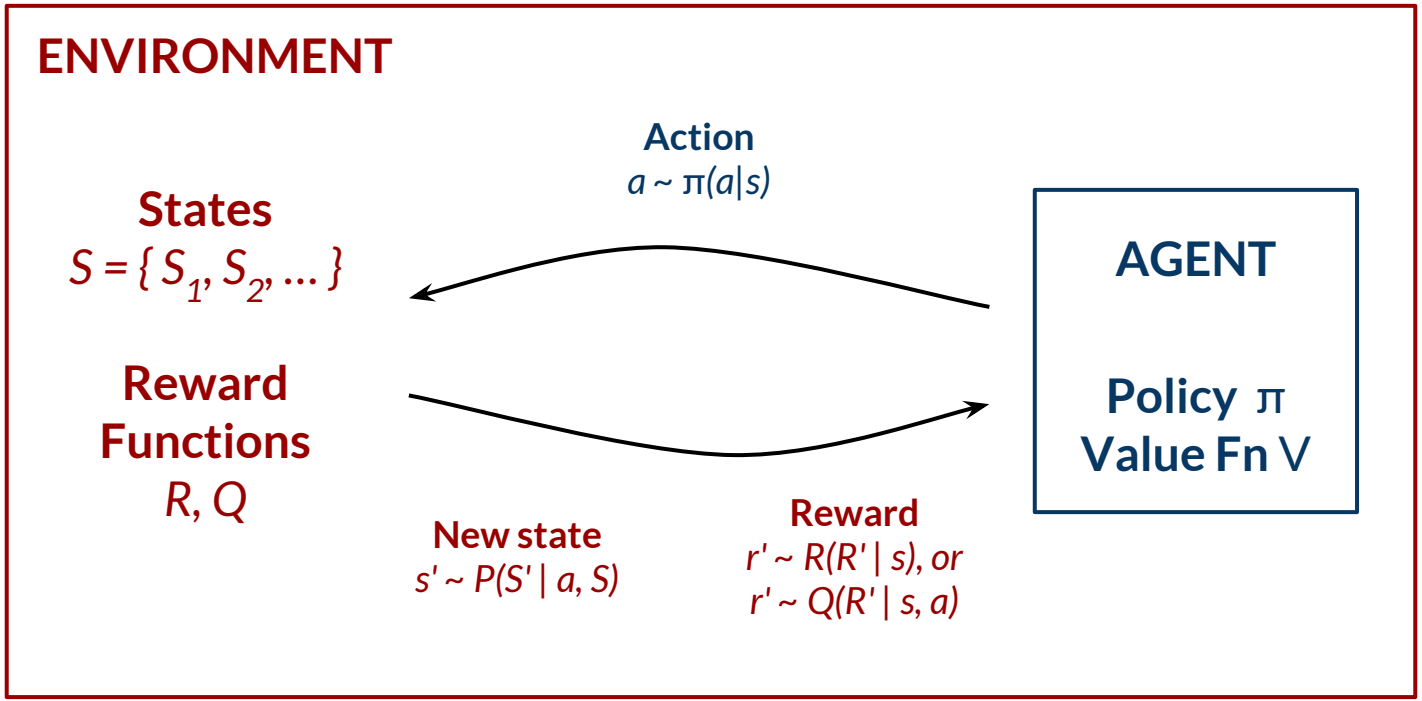
- Model of the environment
- Policy
- Reward function
- **Value function**

  *Values* are numbers indicating the *long-term* desirability of a state

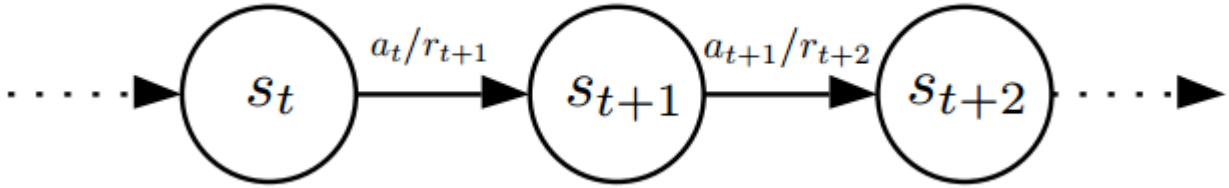
  The total amount of reward the agent can expect to accumulate, starting from that state

  Estimated from sequences of observed actions and rewards

# Mechanics of Reinforcement Learning



# Mechanics of Reinforcement Learning




- $t$ Discrete time step
- $s_t$ State at time $t$
- $r_t$ Reward when in state $s_t$ at time $t$
- $a_t$ Action taken at time $t$, leading to new state $s_{t+1}$ with reward $r_{t+1}$

*Foundations of Machine Learning.* Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.

# Goals

- Maximise the expected return (finite, episodic tasks)

$$E[r_{t+1} + r_{t+2} + ...r_{t+T}] = E[\sum_{k=1}^{T} r_{t+k}]$$

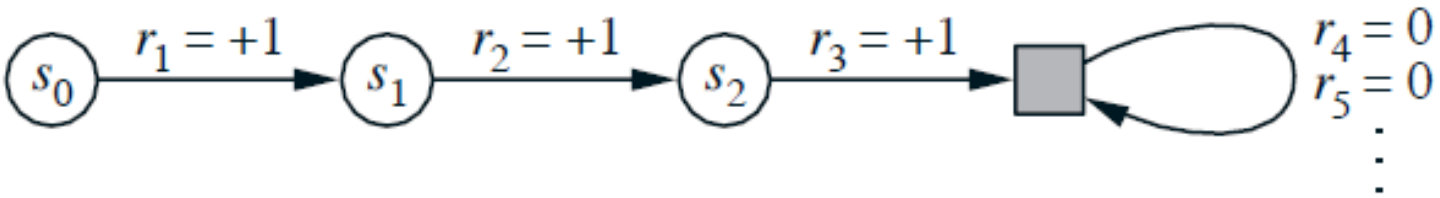
- Maximise the expected **discounted return** (infinite, continuous)

$$E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...] = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}] \qquad 0 \leq \gamma < 1$$

(short-sighted) $0 \leftarrow \gamma \rightarrow 1$ (far-sighted)

# Goals

- Unified notation for episodic and continuous tasks




- Every episode ends in an **absorbing state** that always produces a reward 0

- Both episodic and continuous tasks can now maximise: $E[\sum_{k=0}^{T} \gamma^k r_{t+k+1}]$

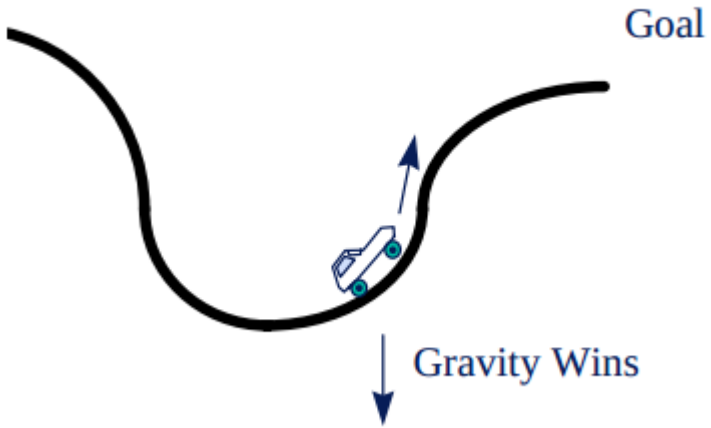
# Examples of Reinforcement Learning

- **Actions:** { Forward, Backward, No Thrust }
- **Reward:** -1 for all state transitions,

  except the goal, where the reward is 0
- **Return:** -Time to reach the goal

The agent does not have enough thrust to simply drive up the hill.

It minimizes the time it takes to reach the goal, by learning to use momentum to its advantage




*Reinforcement Learing: A Tutorial.*
Mance E. Harmon, Stephanie S. Harmon

# Examples of Reinforcement Learning

- Chess: What is the **reward?**
- Robot in a maze: What is the **reward?**
- **Flappy Bird RL hack**: http://sarvagyavaish.github.io/FlappyBirdRL/

# Bandit Problems

# The Multi-Armed Bandit

**A**

**$1 every**
**5 rounds**

**B**

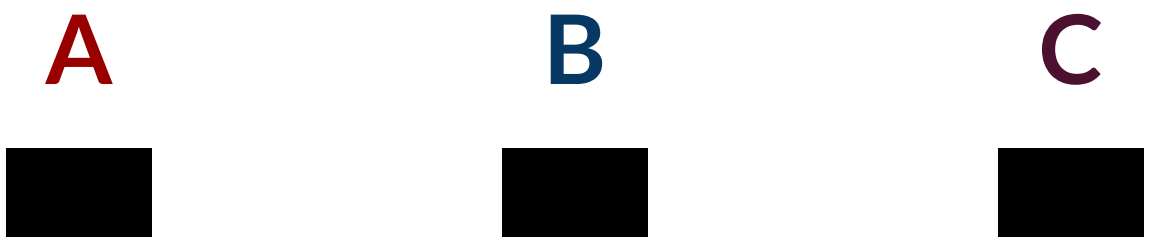
**$1 every**
**7 rounds**

**C**

**$1 every**
**3 rounds**

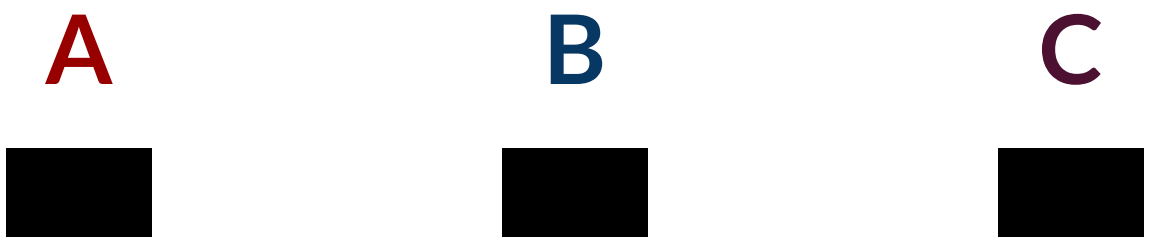
**1000 rounds available**

# The Multi-Armed Bandit



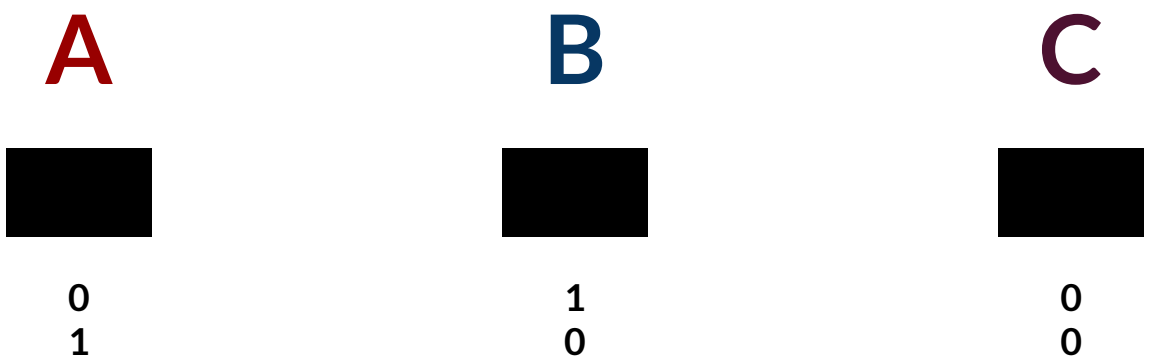

**1000 rounds available**

# The Multi-Armed Bandit




| | | | |
|---|---|---|---|
| **ESTIM. VALUE** | **0** | **1** | **1** |

# The Multi-Armed Bandit




| | | | |
|---|---|---|---|
| **ESTIM. VALUE** | **0.5** | **0.5** | **0** |

# The Multi-Armed Bandit

| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

# The Multi-Armed Bandit

| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

Estimate value by averaging rewards

$$Q_t(a) = \frac{R_1 + R_2 + \ldots R_k}{K_a}$$

# The Multi-Armed Bandit

| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

Estimate value by averaging rewards incrementally

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{t}\left( R_t - Q_t(a) \right)$$

# The Multi-Armed Bandit

| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

Action selection policy: **greedy**

Always select the action with the maximum estimated value

# The Multi-Armed Bandit

| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

Action selection policy: **$\epsilon$-greedy**

With probability $\epsilon$, select an action **uniformly at random**

# The Multi-Armed Bandit

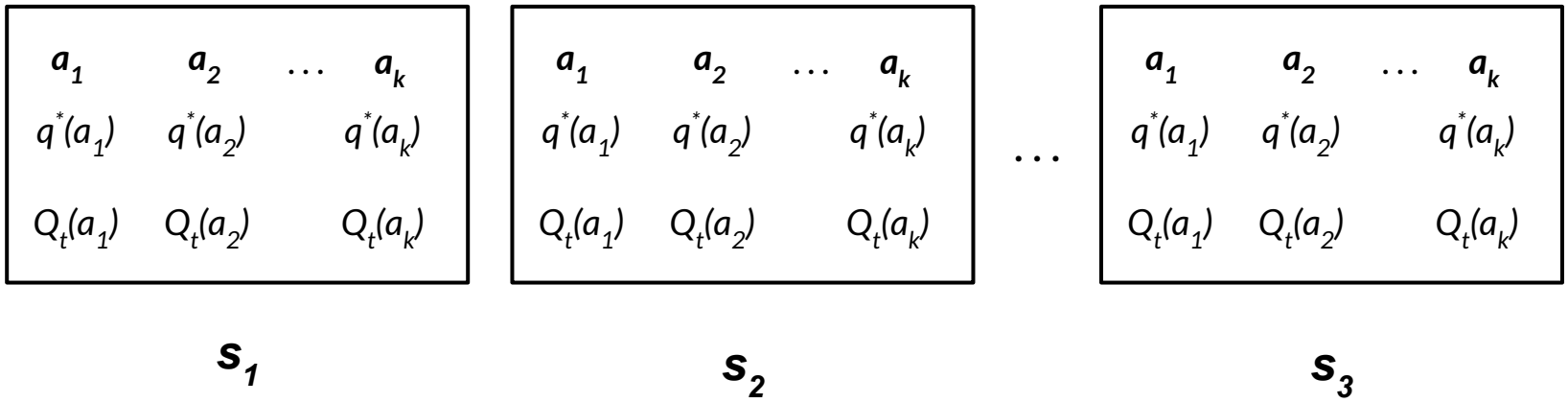
| | $a_1$ | $a_2$ | $\cdots$ | $a_k$ |
|---|---|---|---|---|
| **Actual Value** | $q^*(a_1)$ | $q^*(a_2)$ | | $q^*(a_k)$ |
| **Estimated Value** | $Q_t(a_1)$ | $Q_t(a_2)$ | | $Q_t(a_k)$ |

**Softmax action selection policy**

With probability $\epsilon$, select an action with probability

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{i=1}^{k} e^{\frac{Q_t(i)}{\tau}}}$$

# The Contextual Multi-Armed Bandit




**Policy π(s) = a**

# Formalization

# Markov Property

$$P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0) =$$
$$P(s_{t+1} = s', r_{t+1} = r | s_t, a_t)$$

# Markov Decision Processes

A tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where:

- ***S*** Set of states
- ***A*** Set of actions
- $P_{sa}$ State transition probabilities; a distribution over the state space
- $\gamma$ Discount factor
- ***R*** Reward function of a state (or a state-action pair)

# MDP Dynamics

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

# MDP Dynamics

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots .$$

# MDP Dynamics

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots .$$

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots .$$

# MDP Goal

Choose actions over time so as to maximise the expected value of the total payoff

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

# Policy and Value Functions

## Policy

$$\pi : S \mapsto A$$

$$a = \pi(s)$$

# Policy and Value Functions

**Value function for a policy, starting at state s**

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right].$$

# Policy and Value Functions

**Value function for a policy, for taking an action a in state s**

$$Q^{\pi}(s,a) = E[R(s_0,a_0) + \gamma R(s_1,a_1) + \gamma^2 R(s_2,a_2) + ...|s_0 = s, a_0 = a, \pi]$$

# Bellman Equations

**Current state-dependent reward and deterministic policy**

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

# Bellman Equations

**Current state-dependent reward and stochastic policy**

$$V^{\pi}(s) = R(s) + \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

# Bellman Equations

**Current and next state-dependent reward and stochastic policy**

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{sa}(s') R(s, a, s') + \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

# Optimal Value Function

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

# Bellman Optimality Equations

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s').$$

$$Q^*(a, s) = R(s) + \max_{a' \in A} \gamma \sum_{s' \in S} P_{sa}(s') Q^*(a', s')$$

Value of a state under an optimal policy is the expected return for the best action from that state

# Optimal Policy for all States

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

$$\pi^*(s) = \arg\max_{a \in A} Q^*(a, s)$$

This is the **greedy policy** with respect to **V*** (requires a one-step search**)**
or **Q*** (does not require any search)

# Solving Bellman Equations

Transition probability matrix $\mathbf{P}_{s,s'} = \Pr[s' | s, \pi(s)]$

Value column matrix $\mathbf{V} = V_\pi(s)$

Reward column matrix $\mathbf{R}$

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}$$

*Foundations of Machine Learning.* Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.

# Solving Bellman Equations

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

$$\mathbf{V}_0 = (\mathbf{I} - \gamma \mathbf{P})^{-1}\mathbf{R}.$$

*Foundations of Machine Learning.* Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.

# Planning

# Value Iteration

1. For each state $s$, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$.

   }

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

# Value Iteration

- Has converged when the largest ***V(s)*** update is smaller than $\epsilon$
- Updates to ***V(s)*** may be in a **batch** or for **single states**

# Policy Iteration

1. Initialize $\pi$ randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state $s$, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.
   }

# Policy Iteration VS Value Iteration

- Policy iteration converges in fewer iterations
  But solving a large system of equations in each iteration is expensive

- In practice, value iteration is used more often

# Learning

# Mean Estimation

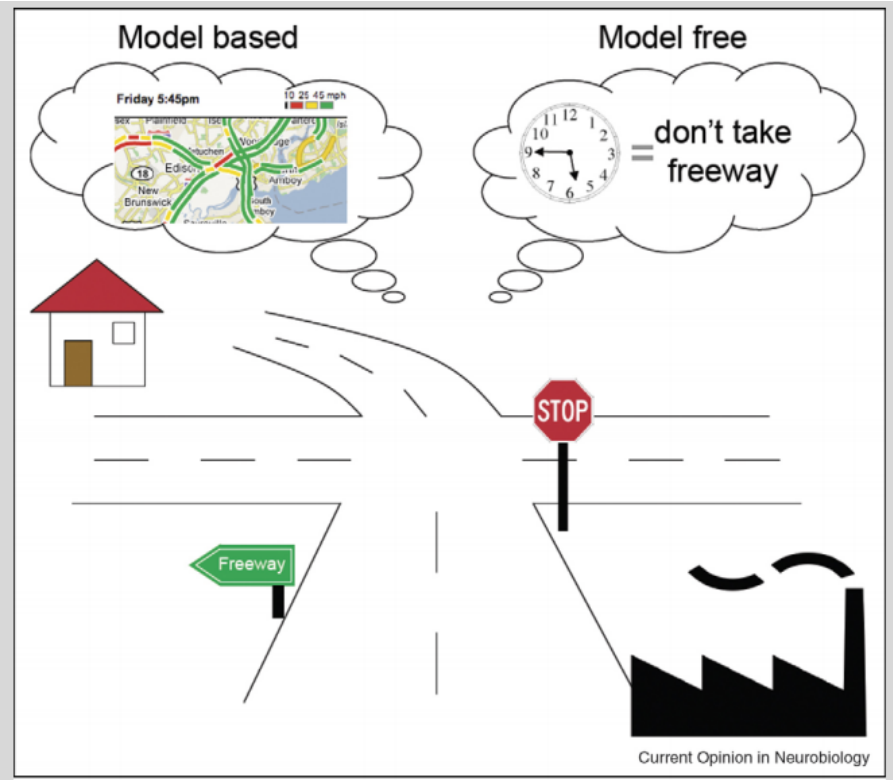
$$
\begin{aligned}
&s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots \\
&s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots \\
&\dots
\end{aligned}
$$

$$
P_{sa}(s') = \frac{\#\text{times took we action } a \text{ in state } s \text{ and got to } s'}{\#\text{times we took action a in state } s}
$$

# Mean Estimation

1. Initialize $\pi$ randomly.
2. Repeat {
   (a) Execute $\pi$ in the MDP for some number of trials.
   (b) Using the accumulated experience in the MDP, update our estimates for $P_{sa}$ (and $R$, if applicable).
   (c) Apply value iteration with the estimated state transition probabilities and rewards to get a new estimated value function $V$.
   (d) Update $\pi$ to be the greedy policy with respect to $V$.
   }

# Model-based VS. Model-free Learning




*Reinforcement learning: The Good, The Bad and The Ugly*. Peter Dayana and Yael Niv.

# Model-based VS. Model-free Learning

- **Model-based:** Try to model the environment

  (state transition probabilities, reward probabilities)

  Eg. Mean estimation

- **Model-free:** Model the value functions directly, without modeling the environment

  Eg. Action selection, TD learning, Q learning, Sarsa algorithm

# References

- **Reinforcement Learning: An Introduction.** Sutton and Barto.
- **Foundations of Machine Learning.** Mehryar Mohri.
- **Reinforcement Learning and Control.** CS229 lecture notes, Andrew Ng.