# Lecture 3: Monte Carlo and Generalization

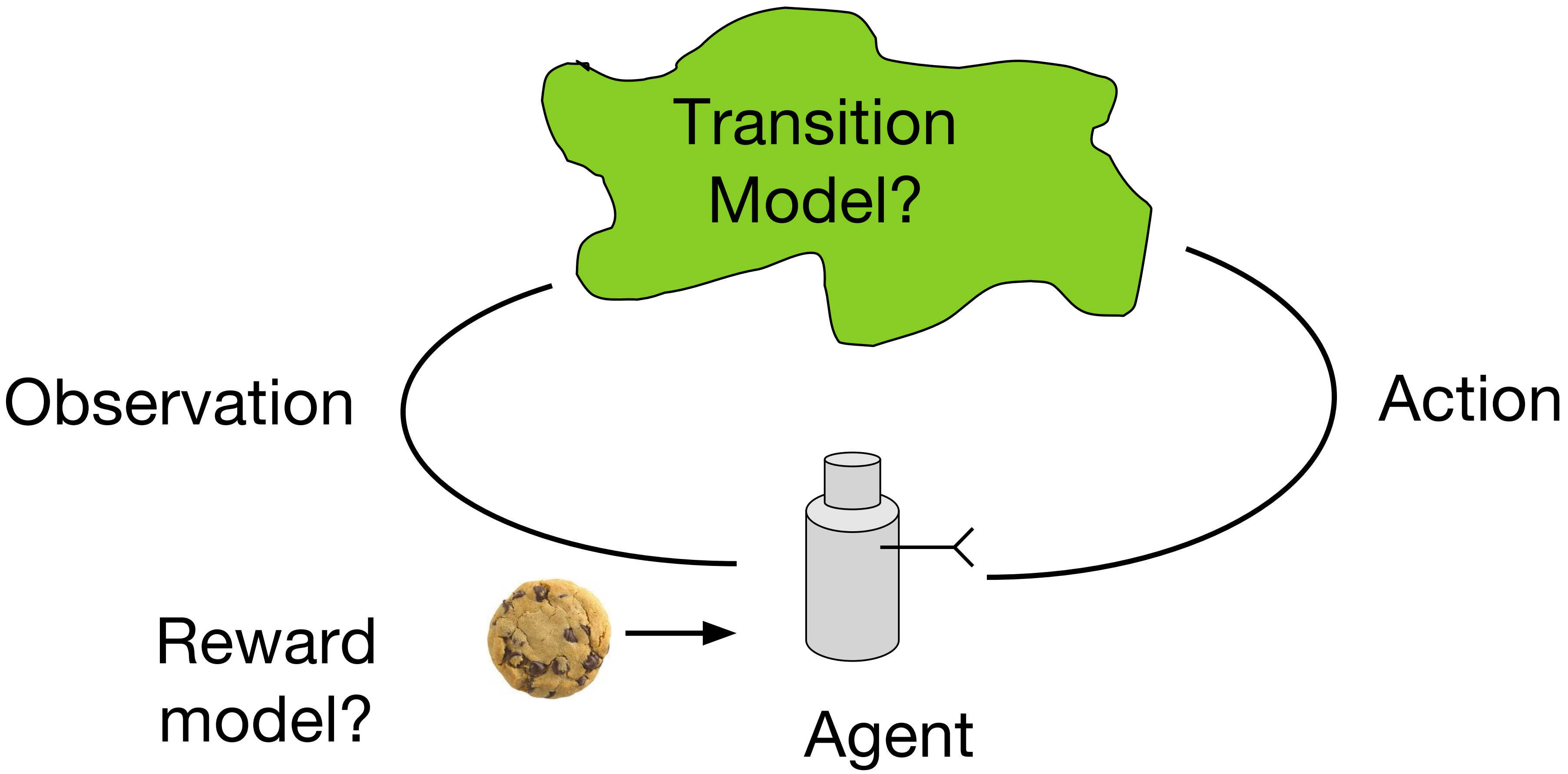
CS234: RL

Emma Brunskill

Spring 2017

Much of the content for this lecture is borrowed from Ruslan Salakhutdinov's class, Rich Sutton's class and David Silver's class on RL.

# Reinforcement Learning



*Goal: Maximize expected sum of future rewards*

## Outline

- Model free RL: Monte Carlo methods
- Generalization
    - Using linear function approximators
    - and MDP planning
    - and Passive RL

# Monte Carlo (MC) Methods

- Monte Carlo methods are learning methods
  - Experience → values, policy
- Monte Carlo uses the simplest possible idea: value = mean return
- Monte Carlo methods can be used in two ways:
  - Model-free: No model necessary and still attains optimality
  - Simulated: Needs only a simulation, not a full model
- Monte Carlo methods learn from complete sample returns
  - Only defined for episodic tasks (this class)
  - All episodes must terminate (no bootstrapping)

# Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π

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

- Remember that the return is the total discounted reward:

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

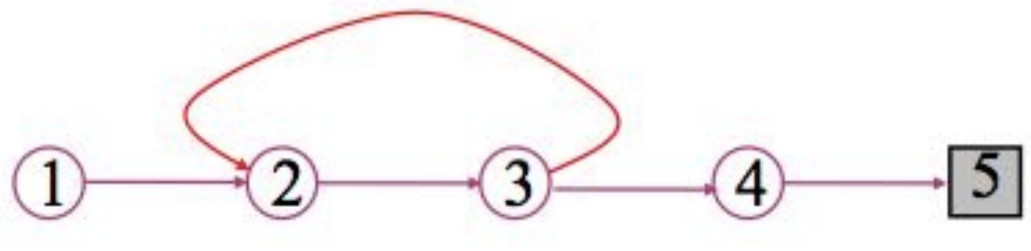
- Remember that the value function is the expected return:

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

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

# Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π
- Idea: Average returns observed after visits to s:



- Every-Visit MC: average returns for every time s is visited in an episode
- First-visit MC: average returns only for first time s is visited in an episode
- Both converge asymptotically
  - Showing this for First-visit is a few lines— see chp 5 in new Sutton & Barto textbook
  - Showing this for Every-Visit MC is more subtle, see Singh and Sutton 1996 Machine Learning paper

## First-Visit MC Policy Evaluation

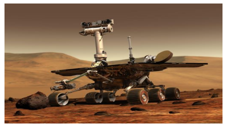
- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

## Every-Visit MC Policy Evaluation

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

| S1 | S2 | S3 | S4 | S5 | S6 | S7 |
|---|---|---|---|---|---|---|
| Okay Field Site +1 |  |  | |  |  | Fantastic Field Site +10 |

- Policy: TryLeft (TL) in all states, use $\gamma$=1, H=4
- Start in state S3, take TryLeft, get r=0, go to S2
- Start in state S2, take TryLeft, get r=0, go to S2
- Start in state S2, take TryLeft, get r=0, go to S1
- Start in state S1, take TryLeft, get r=+1, go to S1
- Trajectory = (S3,TL,0,S2,TL,0,S2,TL,0,S1,TL,1,S1)

| S1 | S2 | S3 | S4 | S5 | S6 | S7 |
|---|---|---|---|---|---|---|
| Okay Field Site +1 | | |  | | | Fantastic Field Site +10 |

- Policy: TryLeft (TL) in all states, use $\gamma$=1, H=4
- Start in state S3, take TryLeft, get r=0, go to S2
- Start in state S2, take TryLeft, get r=0, go to S2
- Start in state S2, take TryLeft, get r=0, go to S1
- Start in state S1, take TryLeft, get r=+1, go to S1
- Trajectory = (S3,TL,0,S2,TL,0,S2,TL,0,S1,TL,1,S1)
- First visit MC estimate of all states?
- Every visit MC estimate of S2?

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

  - Every-Visit MC: average returns for every time s is visited in an episode
  - First-visit MC: average returns only for first time s is visited in an episode

$$V_{samp}(s) = r + \gamma V^{\pi}(s')$$

- TD estimate of all states (init at 0)

$$V^{\pi}(s) = (1-\alpha)V^{\pi}(s) + \alpha V_{samp}(s)$$

# Incremental Mean

- The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally:

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)
\end{aligned}$$

## Incremental Monte Carlo Updates

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state $S_t$ with return $G_t$

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

# MC Estimation of Action Values (Q)

- Monte Carlo (MC) is most useful when a model is not available
  - We want to learn q*(s,a)

- $q_\pi$(s,a) - average return starting from state s and action a following π

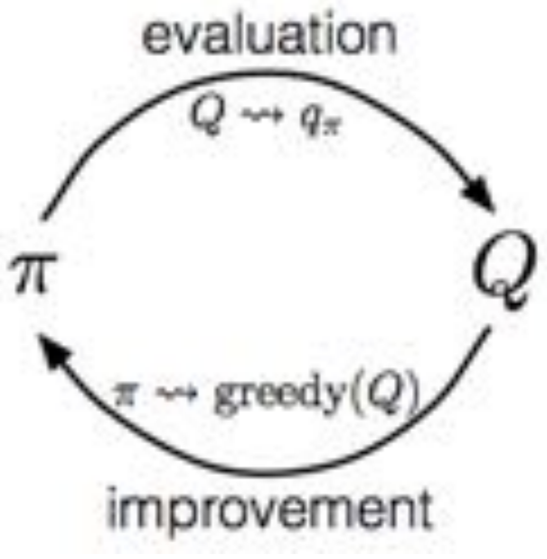
$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

- Converges asymptotically if every state-action pair is visited

- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

# Monte-Carlo Control

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$


evaluation
$Q \rightsquigarrow q_\pi$
$\pi$
$Q$
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

- MC policy iteration step: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedify with respect to value (or action-value) function

# Greedy Policy

- For any action-value function q, the corresponding greedy policy is the one that:
  - For each s, deterministically chooses an action with maximal action-value:

$$\pi(s) \doteq \arg\max_a q(s, a).$$

- Policy improvement then can be done by constructing each $\pi_{k+1}$ as the greedy policy with respect to $q_{\pi k}$ .

## Convergence of MC Control

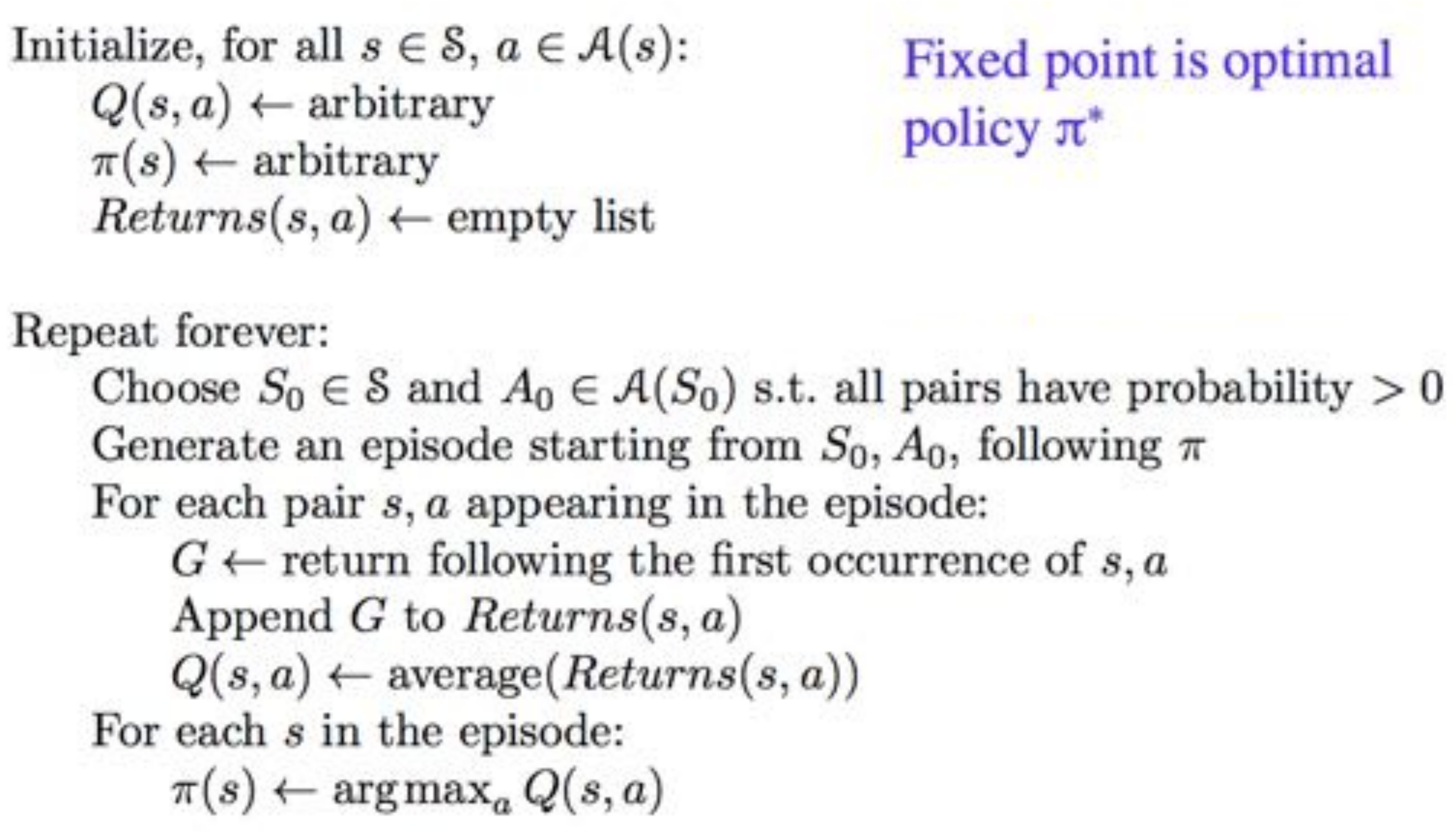
- Greedified policy meets the conditions for policy improvement:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

- And thus must be $\geq \pi_{k.}$

- This assumes exploring starts and infinite number of episodes for MC policy evaluation

# Monte Carlo Exploring Starts



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s, a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability $> 0$
    Generate an episode starting from $S_0, A_0$, following $\pi$
    For each pair $s, a$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s, a$
        Append $G$ to $Returns(s, a)$
        $Q(s, a) \leftarrow$ average$(Returns(s, a))$
    For each $s$ in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s, a)$
```

Fixed point is optimal policy $\pi^*$

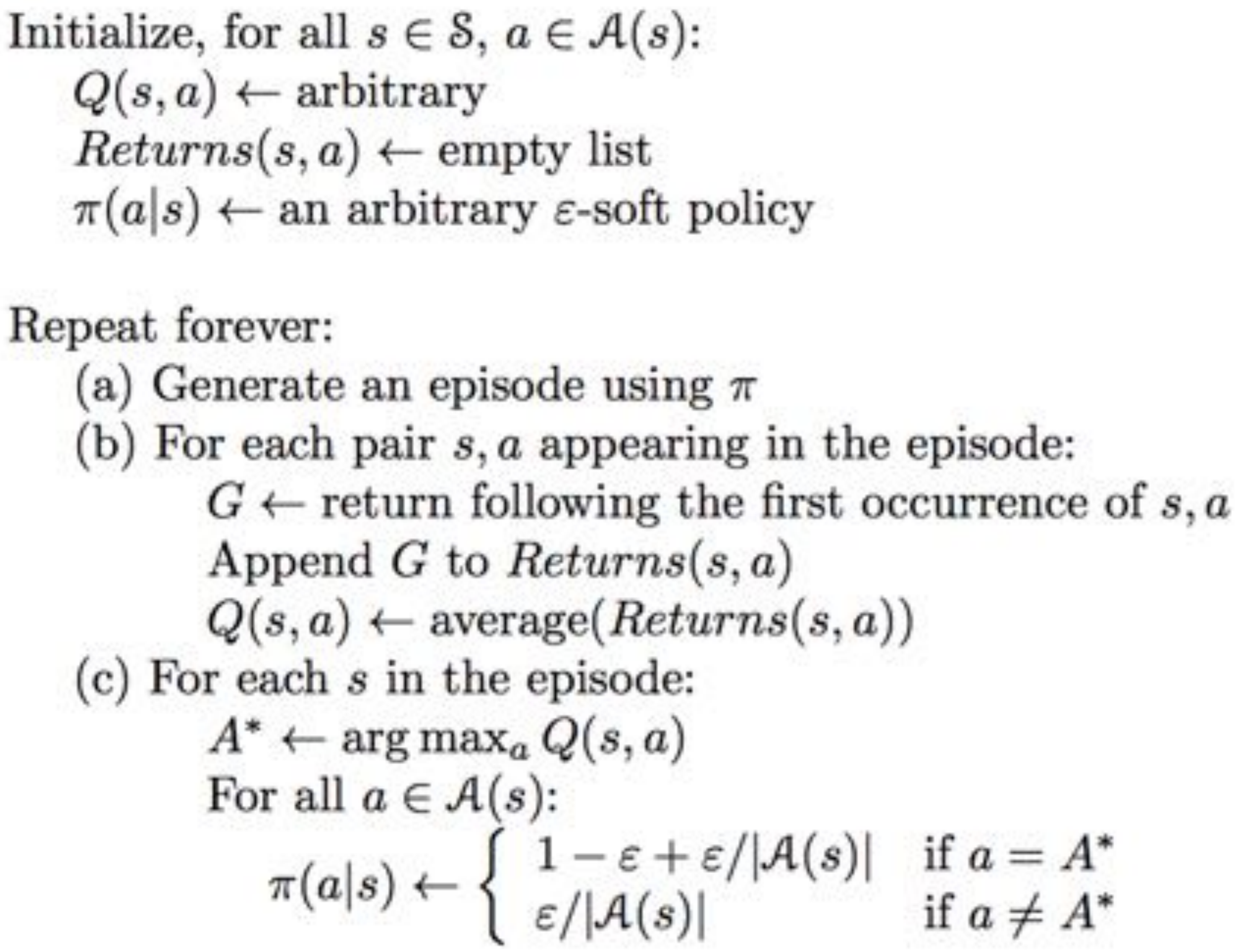
# On-policy Monte Carlo Control

- On-policy: learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally soft: π(a|s) > 0 for all s and a.
- For example, for ε-soft policy, probability of an action, π(a|s),

$$= \underset{\text{non-max}}{\frac{\epsilon}{|\mathcal{A}(s)|}} \quad \text{or} \quad \underset{\text{max (greedy)}}{1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}}$$

- Similar to GPI: move policy towards greedy policy
- Converges to the best ε-soft policy.

# On-policy Monte Carlo Control



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list
    $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $G \leftarrow$ return following the first occurrence of $s, a$
            Append $G$ to $Returns(s,a)$
            $Q(s,a) \leftarrow$ average($Returns(s,a)$)
    (c) For each $s$ in the episode:
            $A^* \leftarrow \arg\max_a Q(s,a)$
            For all $a \in \mathcal{A}(s)$:
                $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$
```

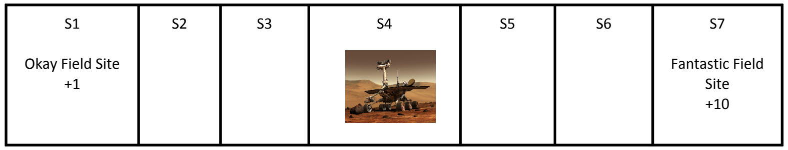
# Summary so far

- MC has several advantages over DP:
  - Can learn directly from interaction with environment
  - No need for full models
  - No need to learn about ALL states (no bootstrapping)
  - Less harmed by violating Markov property (later in class)
- MC methods provide an alternate policy evaluation process
- One issue to watch for: maintaining sufficient exploration:
  - exploring starts, soft policies

## Model Free RL Recap

- Maintain only V or Q estimates
- Update using Monte Carlo or TD-learning
  - TD-learning
    - Updates V estimate after each (s,a,r,s') tuple
    - Uses biased estimate of V
  - MC
    - Unbiased estimate of V
    - Can only update at the end of an episode
- Or some combination of MC and TD
- Can use in off policy way
  - Learn about one policy (generally, optimal policy)
  - While acting using another

## Scaling Up



- Want to be able to tackle problems with enormous or infinite state spaces
- Tabular representation is insufficient

## Generalization

- Don’t want to have to explicitly store a
  - dynamics or reward model
  - value
  - state-action value
  - policy
- for every single state
- Want to more compact representation that generalizes

# Why Should Generalization Work?

- Smoothness assumption
  - if $s_1$ is close to $s_2$, then (at least one of)
    - Dynamics are similar, e.g. $p(s'|s_1,a_1) \cong p(s'|s_2,a_1)$
    - Reward is similar $R(s_1,a_1) \cong R(s_2,a_1)$
    - Q functions are similar, $Q(s_1,a_1) \cong Q(s_2,a_1)$
    - optimal policy is similar, $\pi(s_1) \cong \pi(s_2)$
- More generally, dimensionality reduction / compression possible
  - Unnecessary to individually represent each state
  - Compact representations possible

## Benefits of Generalization

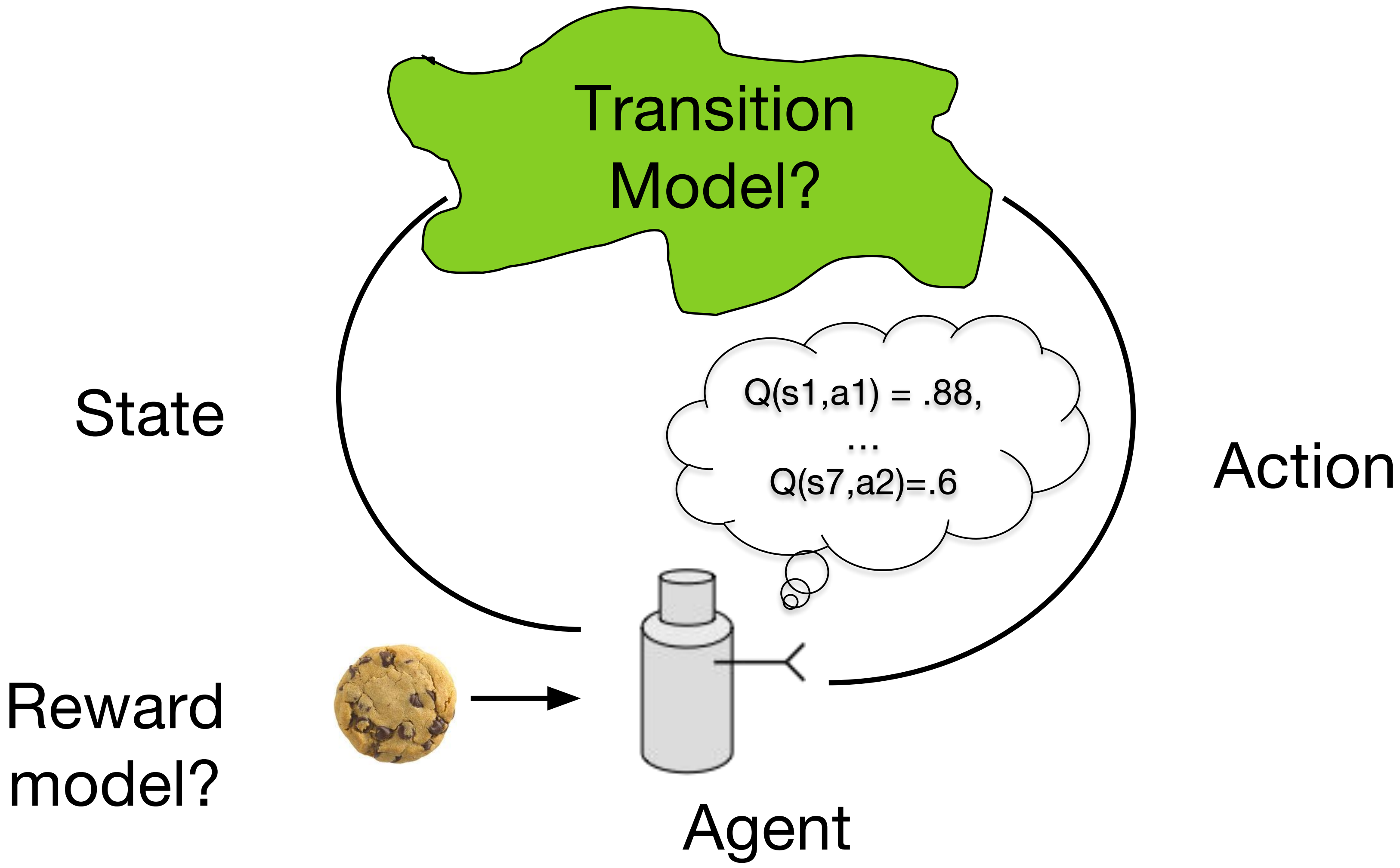
- Reduce memory to represent T/R/V/Q/policy
- Reduce computation to compute V/Q/policy
- Reduce experience need to find V/Q/policy

# Function Approximation

- Key idea: replace lookup table with a function
- Today: model-free approaches
  - Replace table of Q(s,a) with a function
  - Similar ideas for model-based approaches

# Model-free Passive RL:

## Only maintain estimate of V/Q

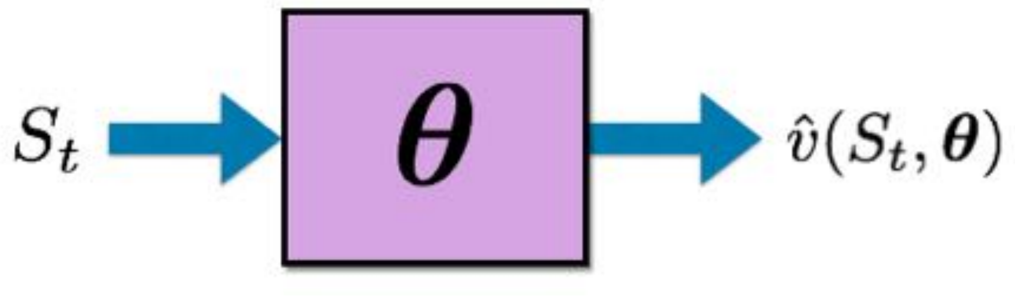


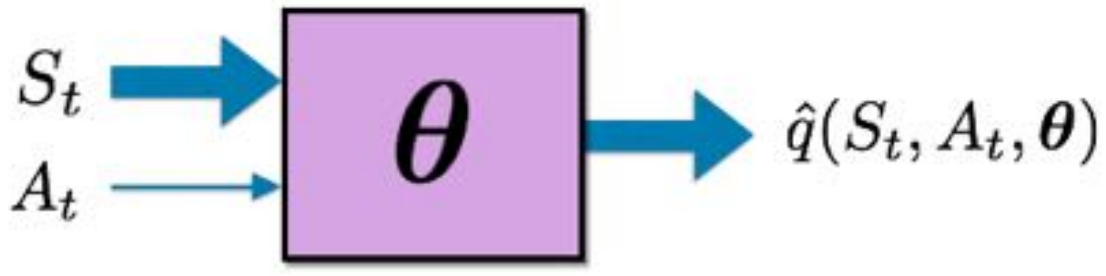
## Value Function Approximation

- Recall: So far V is represented by a lookup table
  - Every state s has an entry V(s), or
  - Every state-action pair (s,a) has an entry Q(s,a)
- Instead, to scale to large state spaces use function approximation.
- Replace table with general parameterized form

# Value Function Approximation (VFA)

- Value function approximation (VFA) replaces the table with a general parameterized form:





# Which Function Approximation?

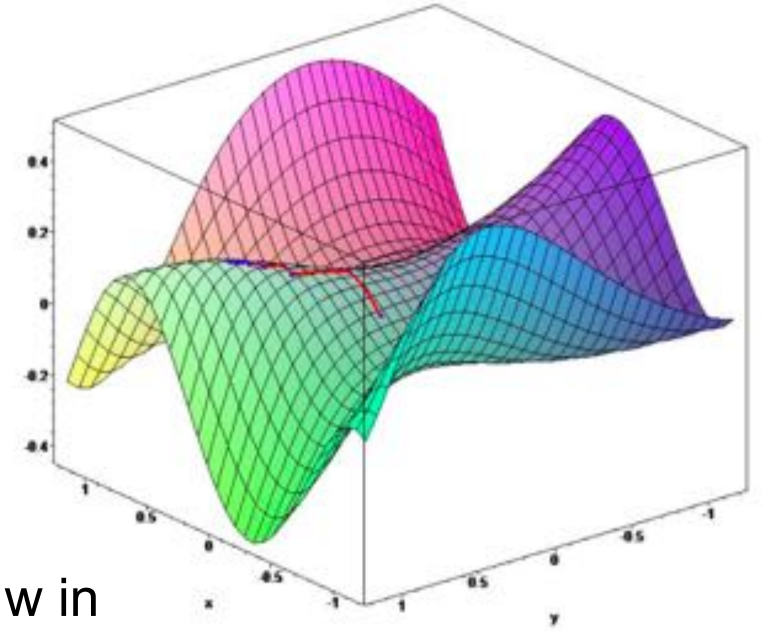
- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural networks
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - …

- We consider differentiable function approximators, e.g.
  - Linear combinations of features
  - Neural networks

# Gradient Descent

- Let J(w) be a differentiable function of parameter vector w
- Define the gradient of J(w) to be:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$



- To find a local minimum of J(w), adjust w in direction of the negative gradient:

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

Step-size

## VFA: Assume Have an Oracle

- Assume you can obtain V*(s) for any state s
- Goal is to more compactly represent it
- Use a function parameterized by weights **w**

# Stochastic Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_\mathbf{w} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent (SGD) samples the gradient:

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

# Feature Vectors

- Represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

# Linear Value Function Approximation (VFA)

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters w

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ (v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

- Update = step-size × prediction error × feature value
- Later, we will look at the neural networks as function approximators.

# Incremental Prediction Algorithms

- We have assumed the true value function $v_\pi(s)$ is given by a supervisor
- But in RL there is no supervisor, only rewards

# VFA for Passive Reinforcement Learning

- Recall in passive RL
    - Following a fixed $\pi$
    - Goal is to estimate $V^{\pi}$ and/or $Q^{\pi}$
- In model free approaches
    - Maintained an estimate of $V^{\pi}$ / $Q^{\pi}$
    - Used a lookup table for estimate of $V^{\pi}$ / $Q^{\pi}$
    - Updated it after each step (s,a,s',r)

# Monte Carlo with VFA

- Return $G_t$ is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, ..., \langle S_T, G_T \rangle$$

- For example, using linear Monte-Carlo policy evaluation

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum

# Monte Carlo with VFA

**Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$**

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \rightarrow \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
    For $t = 0, 1, \ldots, T-1$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big[ G_t - \hat{v}(S_t,\boldsymbol{\theta}) \big] \nabla \hat{v}(S_t,\boldsymbol{\theta})$
```

## Recall: Temporal Difference Learning

- Maintain estimate of $V^{\pi}(s)$ for all states
  - Update $V^{\pi}(s)$ each time after each transition (s, a, s', r)

$$V_{samp}(s) = r + \gamma V^{\pi}(s')$$

$$V^{\pi}(s) = (1-\alpha)V^{\pi}(s) + \alpha V_{samp}(s)$$

# TD Learning with VFA

- Maintain estimate of $V^\pi(s)$ for all states
  - Update $V^\pi(s)$ each time after each transition (s, a, s', r)

$$V_{samp}(s) = r + \gamma V^\pi(s')$$

  - Now treat $V_{samp}$ as the target/ true value function $V^\pi$
  - Adjust weights of approximate V towards $V_{samp}$
  - Remember

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w})$$

$$\uparrow$$

$$V_{samp}(s)$$

# TD Learning with VFA

**Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$**

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\boldsymbol{\theta}$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})\big]\nabla\hat{v}(S,\boldsymbol{\theta})$
        $S \leftarrow S'$
    until $S'$ is terminal
```

# Control with VFA

- Policy evaluation Approximate policy evaluation: $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
- Policy improvement ε-greedy policy improvement

# Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between the true action-value function $q_\pi(S,A)$ and the approximate action-value function:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

# Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S,A) = \begin{pmatrix} \mathbf{x}_1(S,A) \\ \vdots \\ \mathbf{x}_n(S,A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S,A,\mathbf{w}) = \mathbf{x}(S,A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S,A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S,A,\mathbf{w}) = \mathbf{x}(S,A)$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w}))\mathbf{x}(S,A)$$

# Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S,A)$
- For MC, the target is the return $G_t$

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- For TD(0), the target is the TD target: $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

# Incremental Control Algorithms

**Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$**

```
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R, S'$
        If $S'$ is terminal:
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
            Go to next episode
        Choose $A'$ as a function of $\hat{q}(S', \cdot, \boldsymbol{\theta})$ (e.g., $\varepsilon$-greedy)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$
```

# Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is not sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent’s experience (“training data”)

# Least Squares Prediction

- Given value function approximation: $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And experience D consisting of ⟨state,value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Find parameters w that give the best fitting value function v(s,w)?
- Least squares algorithms find parameter vector w minimizing sum-squared error between $v(S_t, w)$ and target values $v_t^\pi$:

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_\mathcal{D}\left[(v^\pi - \hat{v}(s, \mathbf{w}))^2\right] \end{aligned}$$

## SGD with Experience Replay

- Given experience consisting of ⟨state, value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

- Converges to least squares solution

- We will look at Deep Q-networks later.

## Impact of Selected Features

- Crucial
- Features affect
    - **How well can approximate the optimal V / Q**
        - **Approximation error**
    - Memory
    - Computational complexity

## If We Can Represent Optimal V/ Q Can We Always Converge to It?

## Linear Value Function Approximation

- 1 feature, can take on two values
  - f1(s1) = 1, f1(s2) = 2
- 1 action
- R(s1) = R(s2) = 0
- p(s2|s1,a1) = 1 = p(s2|s2,a1)
- ~Q(s,a) = f1(s)*w where w is a (scalar) weight
- What is Q*?
- Can ~Q represent Q*?

## Linear Value Function Approximation

- 1 feature, can take on two values
  - f(s1) = 1, f(s2) = 2
- 1 action, R(s1) = R(s2) = 0,  p(s2|s1,a1) = 1 = p(s2|s2,a1)
- ~Q(s,a) = f(s)*w where w is a (scalar) weight
- What is Q*?    Can ~Q represent Q*?
- Let gamma < 1 and $w_1 = 1$
- Assume two data tuples: (s1,a1,0,s2)  ( s2,a1,0,s2)
- Compute $w_2$ as a function of gamma.
- Will gradient descent converge to the right Q*?

$$w_{k+1} = \arg\min \sum^{N} (V(s_i) - [r_i + \gamma V(s_i')])^2$$

$$w_{k+1} = \arg\min \sum_{i=1}^{N} (wf(s_i) - [r_i + \gamma w f(s_i')])^2$$

$$\frac{d}{dw} \sum_{i=1}^{N} (wf(s_i) - [r_i + \gamma w f(s_i')])^2$$

## Feature Selection

1. Use domain knowledge
2. Use a very flexible set of features & regularize
    - Supervised learning problem!
    - Success of deep learning inspires application to RL
    - With additional challenge that have to gather data