Beyond Q-Learning

Roberto Capobianco

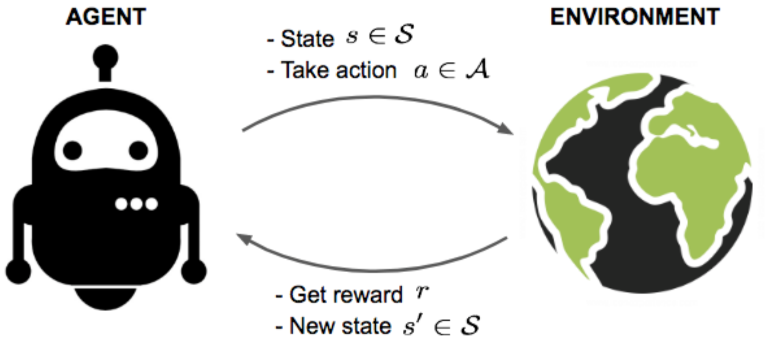
Reinforcement Learning

RL Recap

Differently from dynamic programming (and more generally planning) we do not assume complete knowledge of the environment.

RL methods require only **experience**

- Model (world) generates only transitions
- Probability distribution of transitions is unknown
- Reward function is unknown



Value of a state is the expected cumulated return from that state

Notation & Assumptions

Previous part of the course: state is *x*

Now: state is *s*

Transition probability: P(s’|s,a)

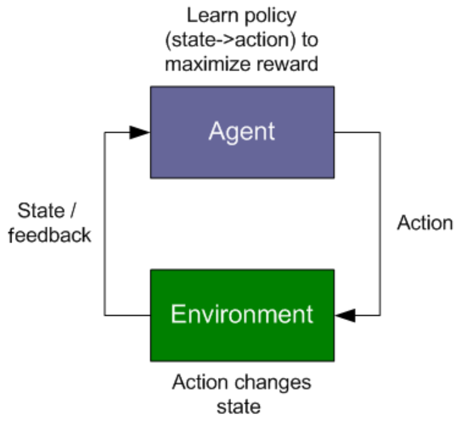
Assumptions:

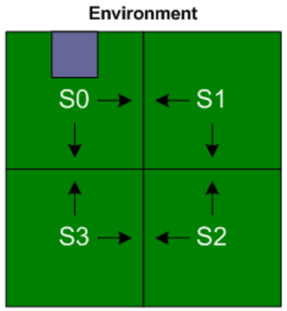
- Experience is divided into episodes
- All episodes eventually terminate
- Only on completion of episode values are estimated and policies changed

Action Values

- If model not available, action values *q* are **very** useful
  - More than state values, that do not determine a policy
  - State values need a lookahead step to see which action is best
  - Without model, state values are not sufficient

- Policy evaluation if model unknown consists in estimating $q_\pi(s, a)$
  - Expected return starting in *s*, taking action *a* and then following π
  - Essentially the same as estimating state values

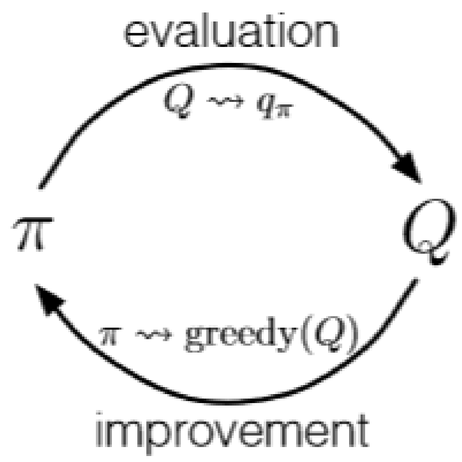




State	Action	Q-Value
S_0	East	Q_{S1}
S_0	South	Q_{S3}
S_1	West	Q_{S0}
S_1	South	Q_{S2}
S_2	West	Q_{S3}
S_2	North	Q_{S1}
S_3	East	Q_{S2}
S_3	North	Q_{S0}

Overall Schema

- Similar to dynamic programming (policy iteration)
  - Maintain approximate policy and approximate value for policy
  - Value is repeatedly altered to better approximate value of π
  - Policy is repeatedly improved with respect to current value
  - Creates moving target for each other, while approaching optimality



RL Policy Iteration

- Alternate complete steps of policy evaluation (E) and improvement (I)
- Begin with arbitrary policy π_0
- End with optimal policy and action-value function

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

- Evaluation is done with value estimate
  - Many episodes experienced
  - Approximate action-value function approaches true one
  - Assume we observe infinite episodes: q_{π_k} is exact for an arbitrary policy π_k

- Policy evaluation is **DIFFICULT**

RL Policy Improvement

- Policy improvement: make policy greedy wrt current value function
  - No model is needed, because we have action value function
  - $\pi(s) = \text{argmax}_a q(s, a)$
  - Policy improvement is done by setting π_{k+1} as greedy policy wrt q_{π_k}

- Policy improvement theorem applies to π_k and π_{k+1}, since

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}\left(s, \text{argmax}_a q_{\pi_k}(s, a)\right) = \max_a q_{\pi_k}(s, a) \geq q_{\pi_k}(s, \pi_k(s)) \geq v_{\pi_k}(s)$$

RL Policy Improvement

- Policy improvement: make policy greedy wrt current value function
  - No model is needed, because we have action value function
  - $\pi(s) = \text{argmax}_a q(s,a)$
  - Policy improvement is done by setting π_{k+1} as greedy policy wrt q_{π_k}

- Policy improvement theorem applies to π_k and π_{k+1}, since

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}\left(s, \text{argmax}_a q_{\pi_k}(s,a)\right) = \max_a q_{\pi_k}(s,a) \geq q_{\pi_k}(s, \pi_k(s)) \geq v_{\pi_k}(s)$$

This is the easy part

SARSA

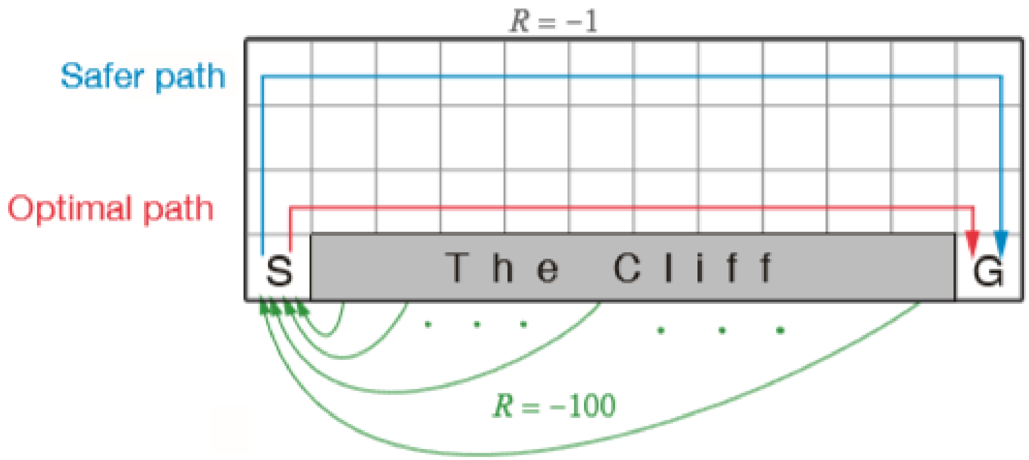
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

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
  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



Data Collection

- Affects the policy evaluation step

- **On-policy** (e.g. SARSA) learns *q*s for a near-optimal policy that explores
  - Try to learn action values conditional on subsequent optimal behavior
  - Need to behave non-optimally to explore all actions
  - Behavior and learning policy are the same

- **Off-policy** (e.g., Q-Learning)**:**
  - Use two policies:
    - One to be learned (**target policy**)
    - One to generate behavior (**behavior policy**)

Data Collection

- **On-policy** (e.g. SARSA) learns *q*s for a near-optimal policy that explores
  - Try to learn action values conditional on subsequent optimal behavior
  - Need to behave non-optimally to explore all actions
  - Behavior and learning policy are the same

- **Off-policy** (e.g., Q-Learning)**:**
  - Use two policies:
    - One to be learned (**target policy**)
    - One to generate behavior (**behavior policy**)

  WHY IS Q-LEARNING OFF-POLICY?

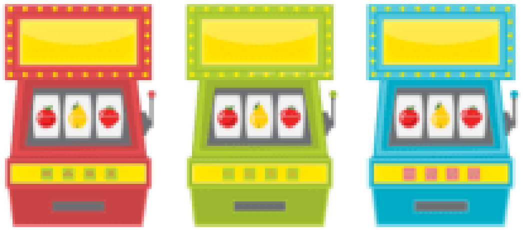
ϵ-greedy

- Policy is generally **soft**
  - $\pi(a|s) > 0, \; \forall\, s \in S, \forall a \in A(s)$
  - Gradually shifted closer and closer to deterministic optimal policy
- We consider ϵ-greedy policies
  - All nongreedy actions are given the minimal probability of selection $\frac{\epsilon}{|A(s)|}$
  - Greedy action has probability $1 - \epsilon + \frac{\epsilon}{|A(s)|}$
  - ϵ-soft policy: $\pi(a|s) \geq \frac{\epsilon}{|A(s)|}$ for all states, actions and for some $\epsilon > 0$
  - Closest to greedy among ϵ-soft policies
- **Why do we need this?**

ϵ-greedy

- Policy is generally **soft**
  - $\pi(a|s) > 0,\ \forall\ s \in S, \forall a \in A(s)$
  - Gradually shifted closer and closer to deterministic optimal policy
- We consider ϵ-greedy policies
  - All nongreedy actions are given the minimal probability of selection $\frac{\epsilon}{|A(s)|}$
  - Greedy action has probability $1 - \epsilon + \frac{\epsilon}{|A(s)|}$
  - ϵ-soft policy: $\pi(a|s) \geq \frac{\epsilon}{|A(s)|}$ for all states, actions and for some $\epsilon > 0$
  - Closest to greedy among ϵ-soft policies
- **Why do we need this?** We need to estimate value of all actions, not just favored ones: EXPLORATION

Explore VS Exploit



EXPLOITATION

Playing the machine that (currently) pays out the most.

EXPLORATION

Playing the other machines to see if any pay out more.

RL Target

- Central idea: update a(n) (action-)value function
- All approaches use this general update step

$$V(s_t) \leftarrow V(s_t) + \alpha[T_t - V(s_t)]$$

  - T_t: target computed at time t
  - α: constant or adaptive step-size
  - Update is done every time non-terminal state is visited

- Target can be computed via
  - (n-step) Bootstrapping or TD error (sampled or expected)
  - Monte-Carlo sampling

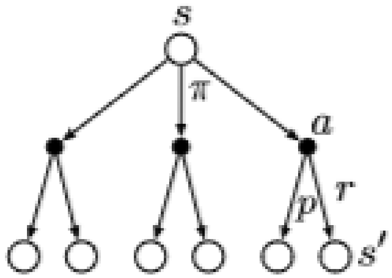
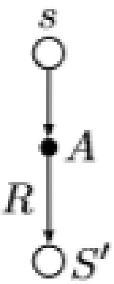
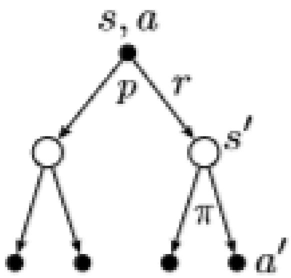
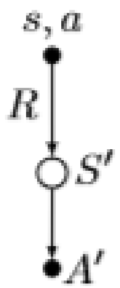
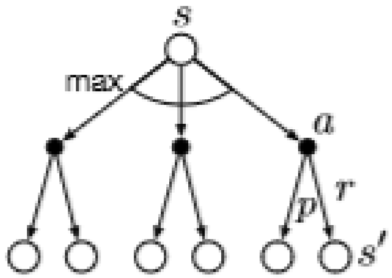
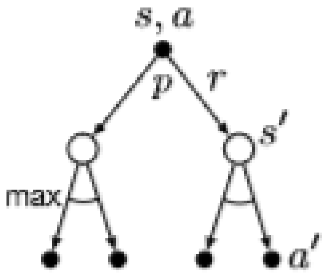
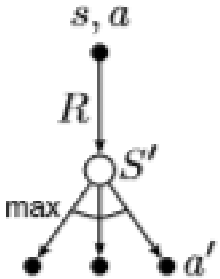
Temporal Difference

- Estimates based on other learned estimates without waiting final outcome
- **Bootstrapping**
  - Q-Learning and SARSA are 1-step TD (lookahead of 1 step)
    Note also that Q-learning is exactly the same as dynamic programming in deterministic MDPs
  - TD error measures difference between:
    - Estimated value of s_t
    - Better estimate $\mathrm{T}_{t+1} = R_{t+1} + \gamma V(s_{t+1})$

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

  - TD error at each time is the error in estimate made at that time
    - Depends on next state and reward, so not available until *t+1*

Expected VS Sampled

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$	 policy evaluation	 TD(0)
$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
$v_*(s)$	 value iteration	
$q_*(s,a)$	 q-value iteration	 Q-learning

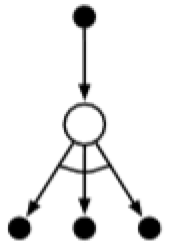
Expected TD

- Like Q-learning, except that instead of max uses expected value
  - Takes into account how likely each action is under current policy

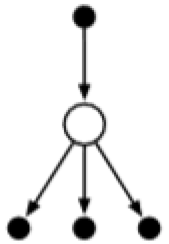
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma E_\pi[Q(s_{t+1}, a_{t+1})|s_{t+1}] - Q(s_t, a_t)]$$

$$\leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Moves deterministically in the same direction as Sarsa moves in expectation
  - More complex than Sarsa
  - Removes variance from Sarsa due to random selection of a_{t+1}
- Can be on-policy or off-policy (named **Expected Sarsa**)



Q-learning



Expected Sarsa

Monte Carlo Methods

Monte Carlo methods:

- Sample and average **complete** returns for each state-action pair
- Idea from the definition of value function (expected return)
- Target G_t: return after time t (needs the episode to finish)

Why **Monte Carlo**?

Estimation involves significant random component (here, complete return)

Can be on-policy or off-policy

On-policy MC Control

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Off-Policy Evaluation

- Happens at policy evaluation: both target and behavior policies are fixed
  - Try to estimate v_π or q_π for target policy π
  - All we have are episodes following another policy b

- To use episodes from *b*, require that
  - Every action taken under π is also taken under *b*
  - $\pi(a|s) > 0 \ \rightarrow b(a|s) > 0$ *(coverage assumption)*
  - *b* must be stochastic in states where it is not identical to π
  - Target policy is typically deterministic greedy wrt current estimate
  - Behavior policy remains stochastic (e.g., ϵ-greedy)

Importance Sampling

- Importance sampling:
  - Estimates expected values under a distribution given samples from another
  - Applied to off-policy learning by weighting returns
    - Weight: relative probability of trajectories occurring under both policies
    - Known as **importance-sampling ratio**

- Relative probability (importance-sampling ratio) is

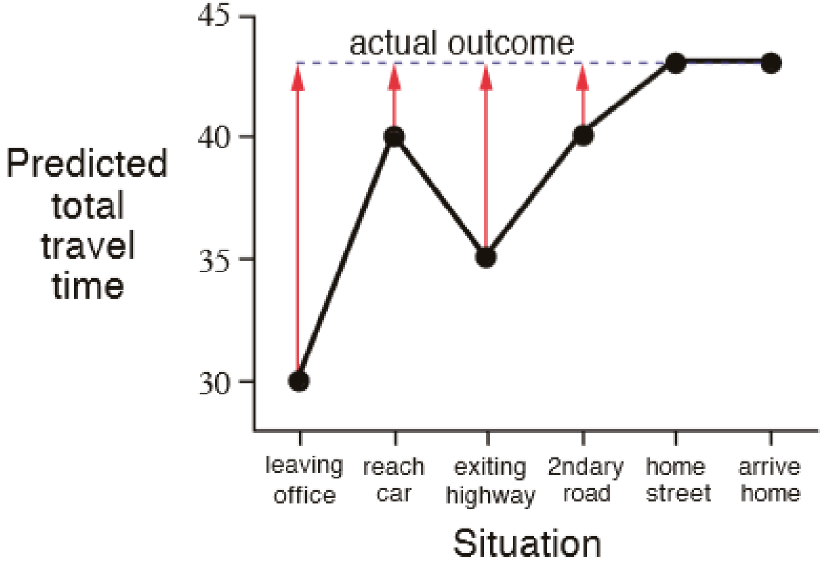
$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(a_k|s_k)p(s_{k+1}|s_k, a_k)}{\prod_{k=t}^{T-1} b(a_k|s_k)p(s_{k+1}|s_k, a_k)} = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{b(a_k|s_k)}$$

- We want to estimate expected returns of π, with returns G_t from b
  - Ratio transforms them:

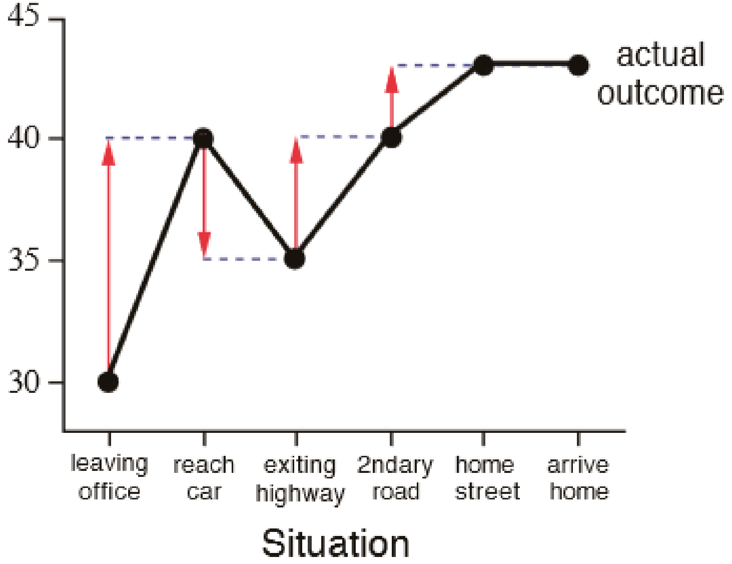
$$\mathrm{E}[\rho_{t:T-1} G_t|s_t] = v_\pi(s) = \frac{\sum_{t \in T(s)} \rho_{t:\mathrm{Termination}(t)-1} G_t}{|T(s)|}$$

MC vs TD

- If estimate of V does not change during episode (as in MC methods)
- If V is updated during episode (as in TD(0)) identity is not exact
  - If step size is small it still holds approximately
- TD does better credit assignment and does not need to wait termination
- MC better ‘offline’ and more stable, TD more incremental



Monte Carlo Update



1-step TD Update

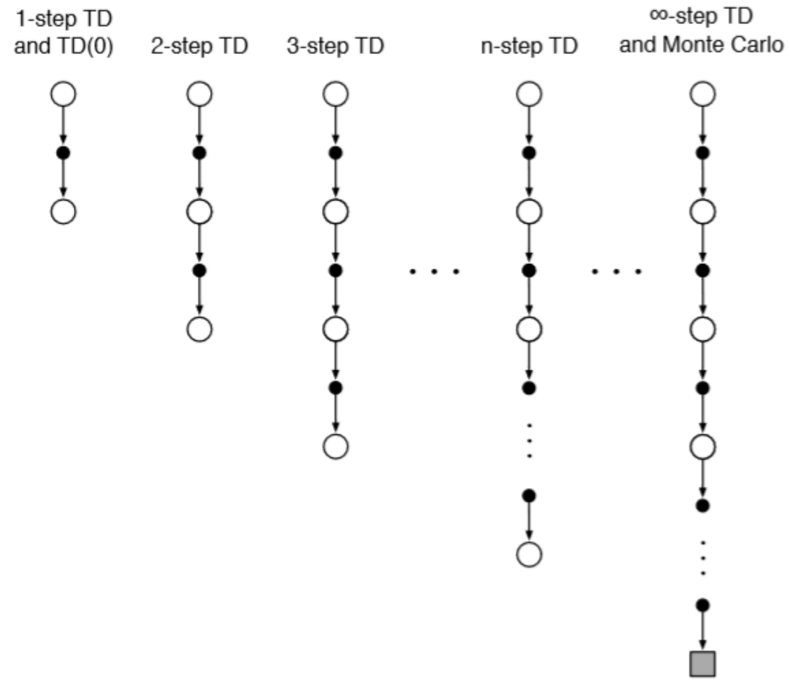
n-step TD Methods

- Unify MC and one-step TD methods
- Generalize them so to smoothly switch among them
- **Motivation**: one-step might not be enough to get significant state changes
- **Solution**: enable bootstrapping to occur over multiple steps

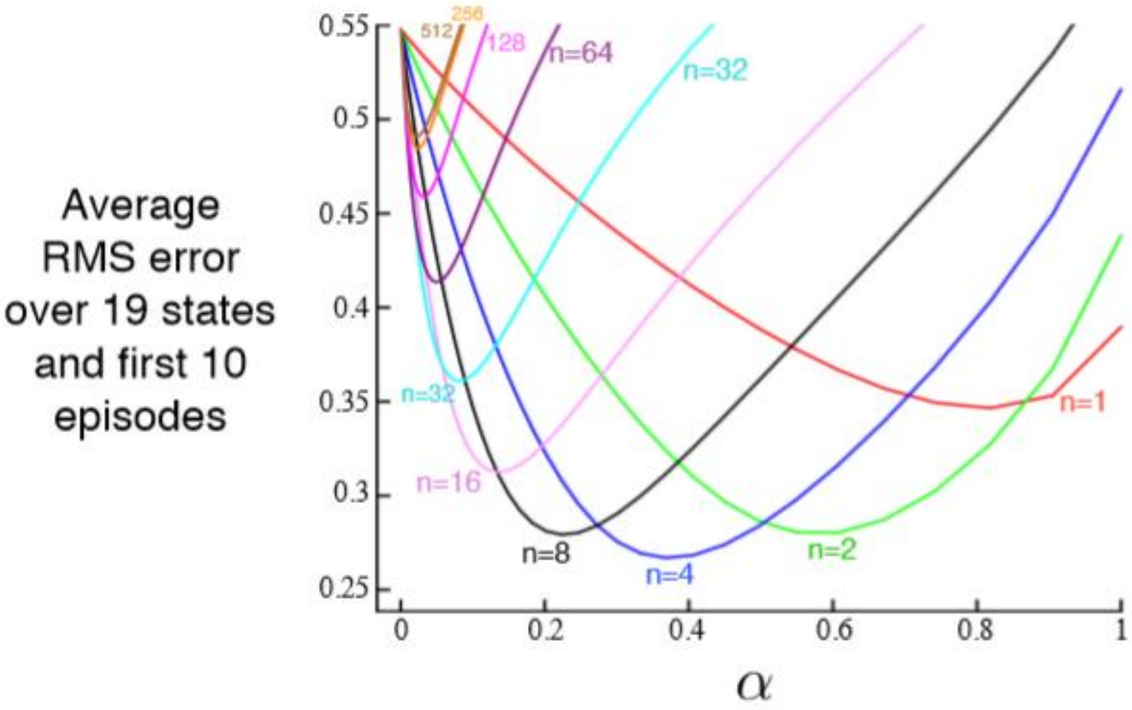
- *n*-step return:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \gamma^{n-1} r_{t+n} + \gamma^n V_{t+n-1}(s_{t+n})$$

  - For all n, t such that $n \geq 1$ and $0 \leq t < T - n$
  - Approximates full return truncated after n steps
  - Needs to wait until it sees r_{t+n} and computed V_{t+n-1} (at *t+n*)



n-step TD Methods

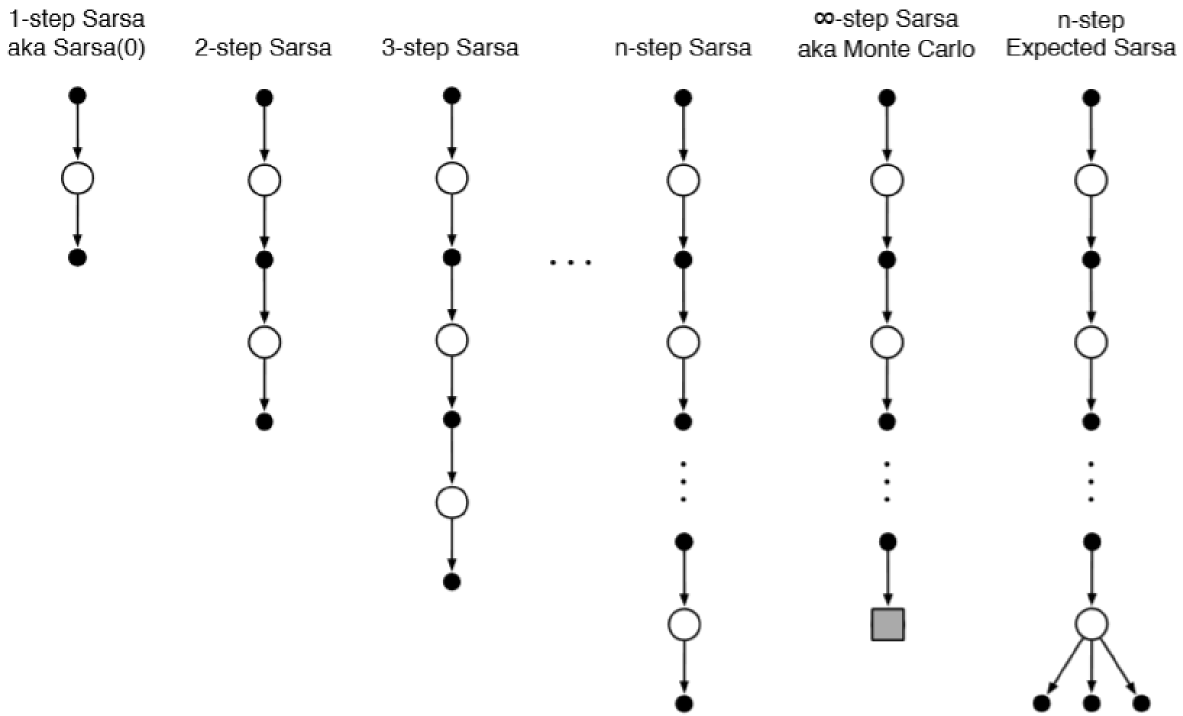


Can you explain why?

n-step SARSA

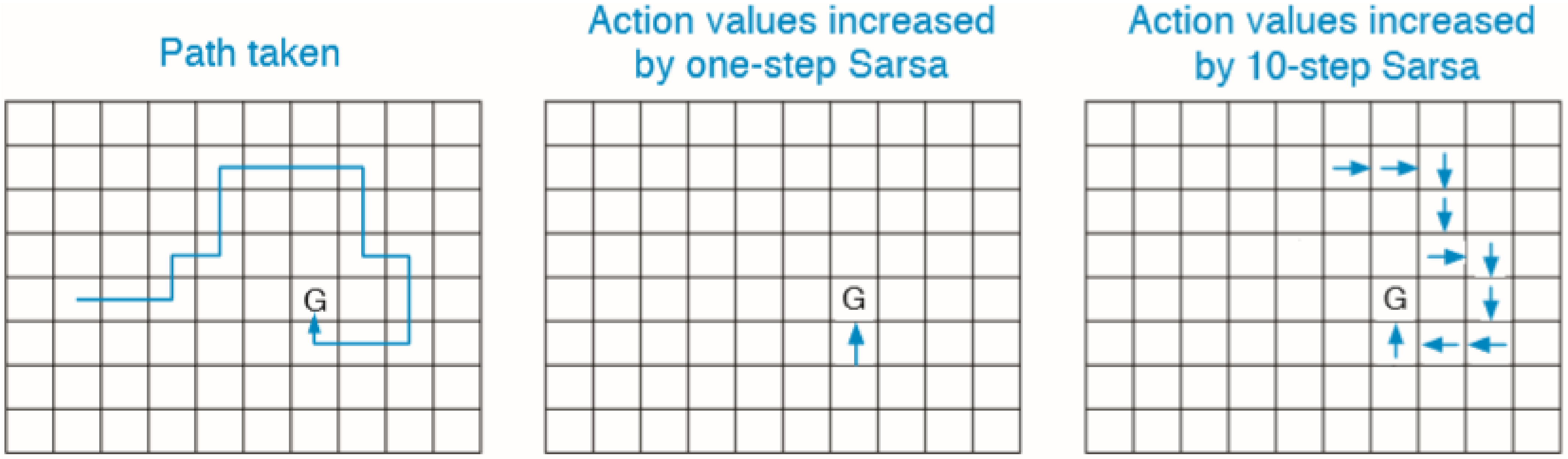
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take action A_t
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then:
    | $T \leftarrow t+1$
    | else:
    | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
    | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
    Until $\tau = T - 1$



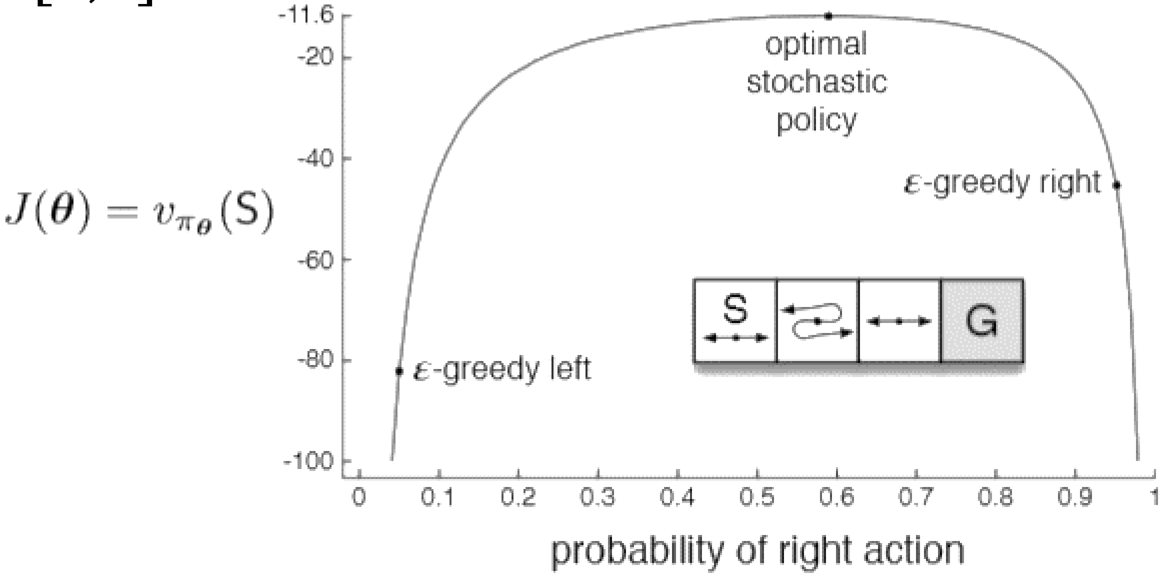
The off-policy version also exists

- Reward 0 everywhere except at G
- One-step methods strengthen only last action
- *n*-step methods strengthen last *n* actions



Action-Value Unstable

- Scenario:
  - Short corridor, reward -1 per step
  - Actions: right and left
  - Actions effect are as usual in first and third states, reversed in second state
  - All states appear identical in their featurization $x(s, \text{right}) = [1,0]^T$ and $x(s, \text{left}) = [0,1]^T$



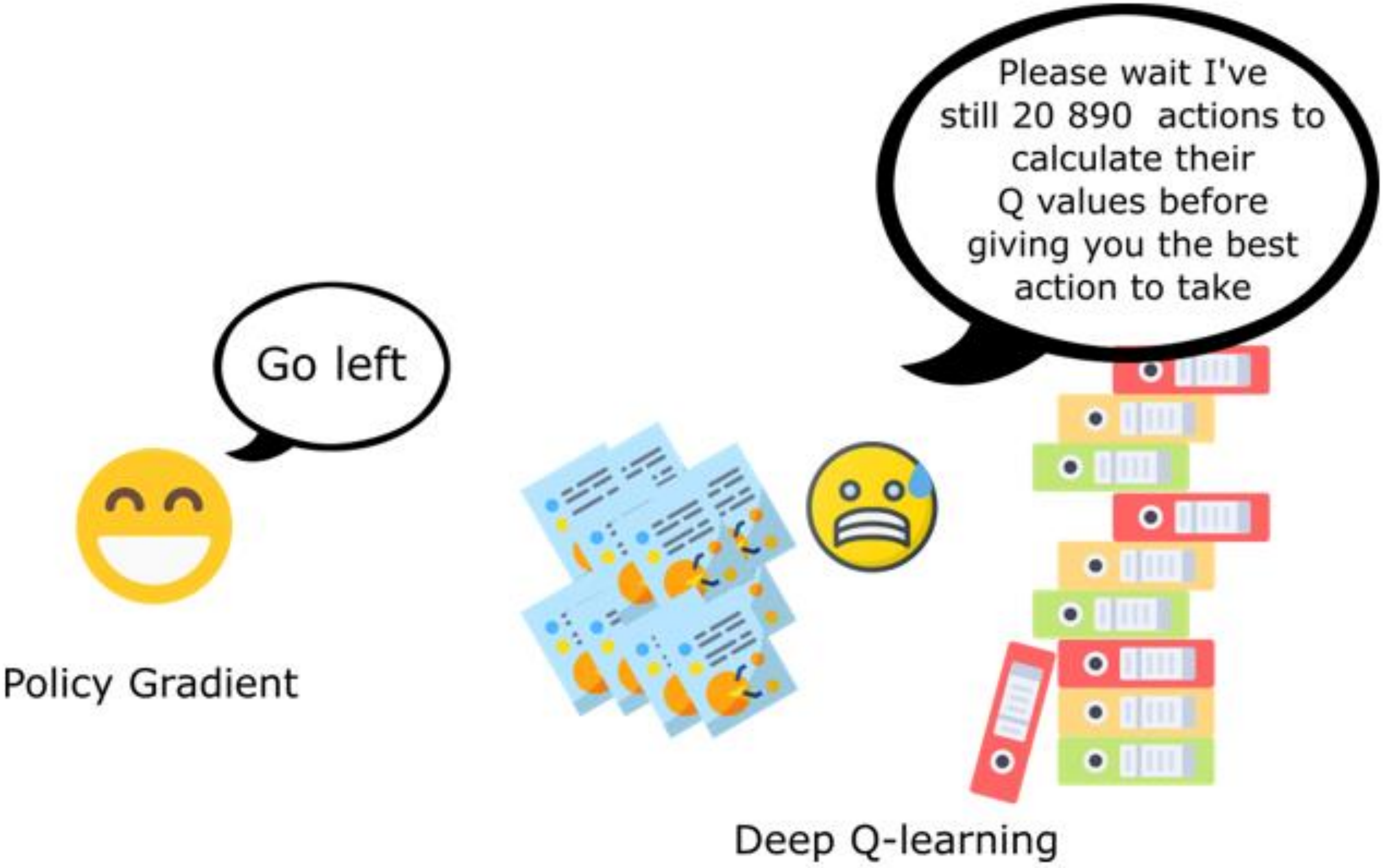
- Value-based methods have big oscillations while training
- Choice of actions may change dramatically for arbitrarily small change in action value

Beyond Action-Value

- Action-value methods
  - Learn values of actions
  - Select actions based on their estimated action-value
  - Difficult for continuous actions
- Policy gradient methods
  - Learn a parameterized policy $\pi(a|s,\theta)$ - Parameters θ
  - Select actions without consulting a value function
  - **Requirements:** Policy must be differentiable and should never become deterministic

- Value function may still be used to learn policy parameters
  - Not required for action selection
  - Can be learned as well using approximation, as $\hat{v}(s,w)$ - Weights w

Beyond Action-Value

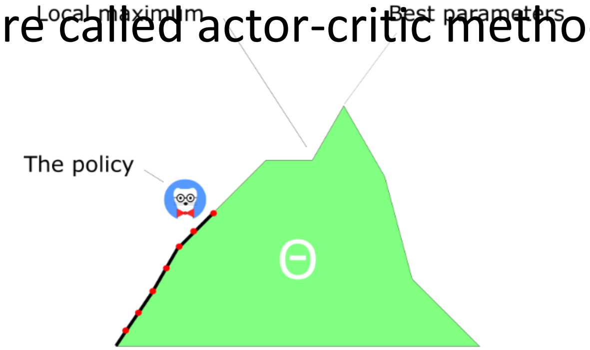


Policy Gradient

- Policy gradient methods learn policy parameters based on $J(\theta) = v_{\pi_\theta}(s_0)$
  - Metric with respect to policy parameters
  - Guaranteed to converge to local maximum or global maximum
  - Disadvantage: often converge only to local optimum
- Attempt to maximize performance through gradient ascent

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

- $\widehat{\nabla J(\theta_t)}$ is a stochastic estimate
  - Its expectation approximates gradient of performance wrt params θ_t
- Methods that learn approximations to both policy and value functions are called actor-critic methods
  - Subset of policy-gradient methods



PG Action Selection

- Action with highest preference are given highest probability of selection
  - E.g., exponential soft-max distribution

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$

  - Advantages:
    - Policy can approach deterministic (eps greedy can't)
    - Enables selection of actions with arbitrary probabilities
    - Easy to inject prior knowledge and more effective in high-dimensional action space
    - Action probabilities change smoothly

- Action preferences can be parameterized as desired
  - E.g., NN where parameters are network weights
  - E.g., linear in features: $\mathrm{h}(s,a,\theta) = \theta^T x(s,a)$, x(s,a) being computed features

- Policy might be simpler to approximate than action-value functions

PG Challenges

- Performance depends on:
  - Action selection
  - State distribution
  - Both are affected by policy params
- Given a state, effects of policy params can be easily computed
- Effects of state distribution depend on environment
  - They are typically unknown

Theoretical answers are given by policy gradient theorem

$$\nabla J(\theta) \propto E_\pi \left[\sum_a \nabla \pi(a|s_t) q_\pi(s_t, a) \right]$$

- Provides analytic expression for gradient of performance wrt policy params
  - **Does not involve derivative of state distribution**

REINFORCE

$$\nabla J(\theta) = E_\pi\left[\sum_a \nabla\pi(a|s_t,\theta) q_\pi(s_t,a)\right] = E_\pi\left[\sum_a \pi(a|s_t,\theta) q_\pi(s_t,a)\frac{\nabla\pi(a|s_t,\theta)}{\pi(a|s_t,\theta)}\right] = E_\pi\left[q_\pi(s_t,a_t)\frac{\nabla\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}\right] = E_\pi\left[G_t\frac{\nabla\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}\right]$$

Because $E_\pi[G_t|s_t,a_t] = q_\pi(s_t,a_t)$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Hence increment is proportional to product of:

- Return and Gradient of probability of taking action taken divided by probability of taking it

$$\theta_{t+1} = \theta_t + \alpha G_t \boxed{\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}}$$

- Direction in parameter space that most increases probability of taking that action in that state
- Increases proportional to return and Inversely proportional to action probability (otherwise frequent actions have advantage)

Actor-Critic

- Reinforce converges to local minimum
  - It's MC → Tends to learn slowly
  - Inconvenient for online or continuing problems

- TD methods help eliminating these problems
- To gain these advantages in case of PG we use actor-critic methods
  - Critic (value function) bootstraps
- Replace full return of REINFORCE with one-step return

$$\theta_{t+1} = \theta_t + \alpha(G_{t:t+1} - \hat{v}(s_t, w)) \frac{\nabla \pi(a_t|s_t, \theta_t)}{\pi(a_t|s_t, \theta_t)} = \theta_t + \alpha(r_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)) \frac{\nabla \pi(a_t|s_t, \theta_t)}{\pi(a_t|s_t, \theta_t)}$$

$$= \theta_t + \alpha \delta_t \frac{\nabla \pi(a_t|s_t, \theta_t)}{\pi(a_t|s_t, \theta_t)}$$

1-step Actor-Critic

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Continuous Actions

- PG is practical for large action spaces that are even continuous
- Learn statistics of probability distribution instead of computing learned probabilities for each action
  - E.g., choose actions from a normal distribution
  - Function approximation is done for mean and std

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\left(-\frac{\left(a-\mu(s,\theta)\right)^2}{2\sigma(s,\theta)}\right)$$

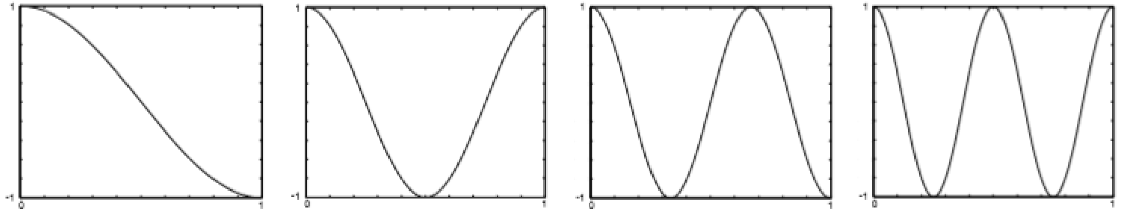
Features in RL

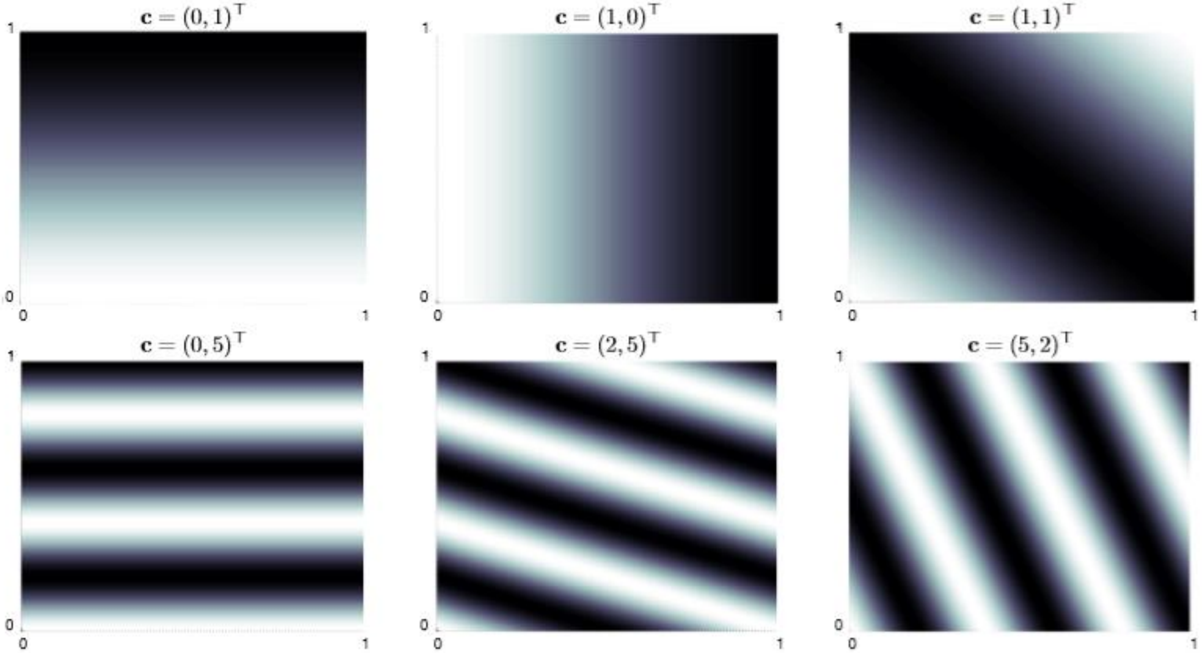
- Features add prior domain knowledge to RL systems
- Correspond to aspects of state space along which generalization is appropriate

- Examples:
  - Polynomials - E.g., $x(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_s^2)$
  - Fourier Basis
  - Coarse Coding
  - Tile Coding
  - Radial Basis Functions

Fourier Basis

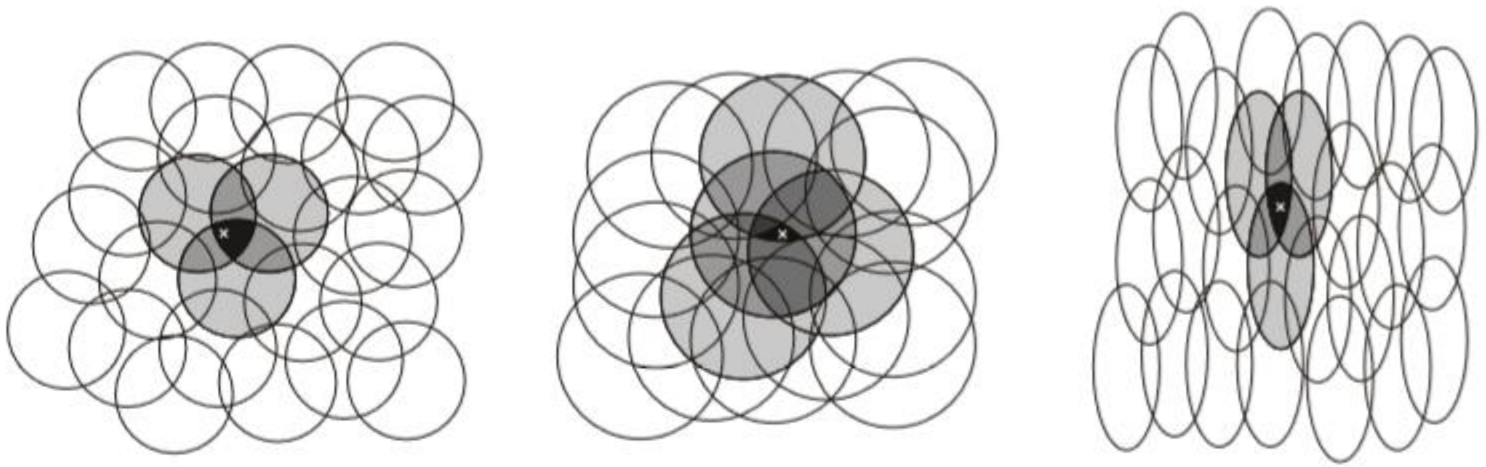
- Easy to use $$x_i(s) = \cos\left(\pi \mathbf{s}^\top \mathbf{c}^i\right)$$
- Perform well in a range of problems
- $s \in [0,1], \quad c \in \{0, \dots, n\}$





Coarse Coding

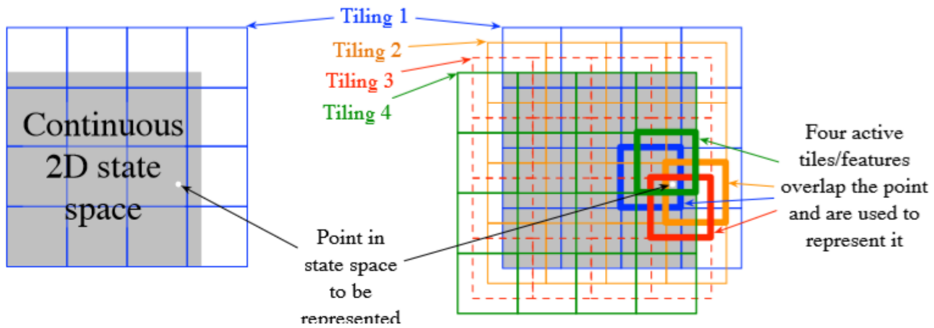
- State space is continuous
- Create 'circles' in state space
- If state is inside a circle, feature is 1, otherwise 0
- Features of this type overlap

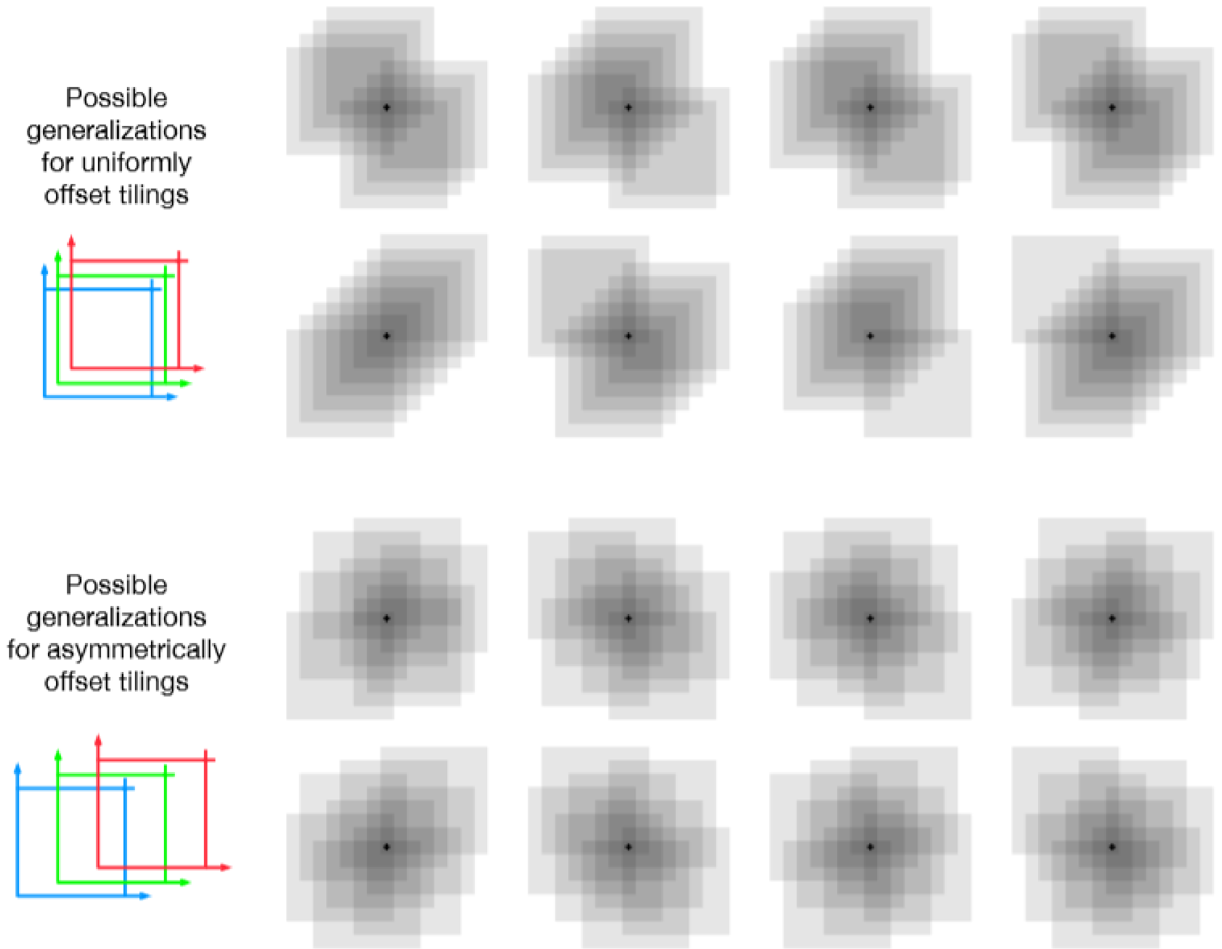


Narrow generalization   Broad generalization   Asymmetric generalization

Tile Coding

- Form of coarse coding that is flexible and computationally efficient
- More tilings enable generalization outside the same box





- Generalize coarse coding to continuous features

$$x_i(s) \doteq \exp\left(-\frac{||s - c_i||^2}{2\sigma_i^2}\right)$$