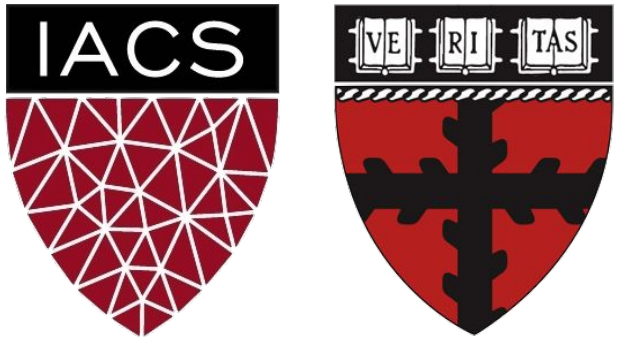
Advanced Section: Reinforcement Learning

CS10B Data Science 2

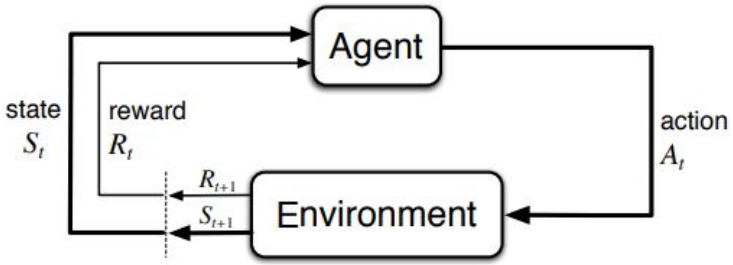
Javier Zazo



Outline

1. Review.
    a. Markov Decision process.
    b. Value Functions.
    c. Bellman Equation.
2. Optimality.
3. Value Iteration vs. Policy Iteration.
4. Exploration - Exploitation tradeoff
5. Temporal Difference
    a. SARSA
    b. Q learning
6. Approximate Q Learning
    a. Linear functions.
    b. Neural Networks.
7. **Resources**

Problem Setting



- Receive feedback in the form of rewards
- Agent's utility is defined by the reward function
- Must (learn to) act so as to maximize expected rewards
- **Model Free:** All learning is based on observed samples of outcomes!

Markov Decision Process

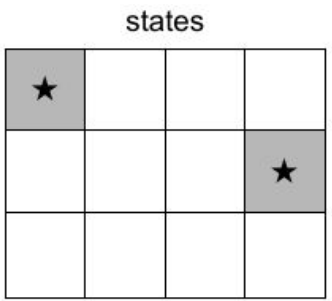
- Mathematical formulation of the RL problem
- Markov property: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

actions = {

1. right
2. left
3. up
4. down

}

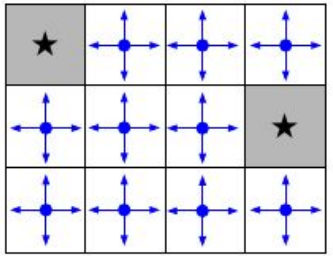


$\mathcal{S}$ - set of possible states
$\mathcal{A}$ - set of possible actions
$\mathcal{R}$ - distribution of reward given (state, action) pair
$\mathbb{P}$ - transition probability i.e. distribution over next state given (state, action) pair
γ - discount factor

Objective: reach terminal state
Rewards: -1 for every movement

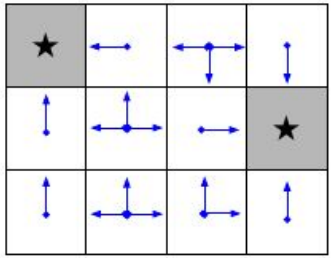
Markov Decision Process Example

1. At time step t=0, environment samples initial state $\mathbf{s}_0 \sim \mathrm{p}(\mathrm{s}_0)$
2. For t=0 until done:
   a. Agent selects action a_t
   b. Environment samples reward $r_t \sim R(.|s_t, a_t)$
   c. Environment samples next state $s_{t+1} \sim P(.|s_t, a_t)$
   d. Agent receives reward r_t and next state s_{t+1}



Random policy

- A policy is a function from S to A that specifies what action to take in each state.
- Objective: find policy π^* that maximizes cumulative discounted reward: $\sum_{t>0} \gamma^t r_t$



Optimal policy

Value function and Q-value function

- Following a policy produces sample trajectories (or paths) s0, a0, r0, s1, a1, r1, …

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right] \quad \leftarrow \text{random vs. deterministic} \rightarrow \quad \pi(s) = a$$

- The value function at state s, is the expected cumulative reward from following the policy from state s:

$$v_\pi(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- The Q-value function at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$q_\pi(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Bellman Equation

- State Value function

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

- State-Action Value Function

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

- **Solution to the Bellman Equation:**

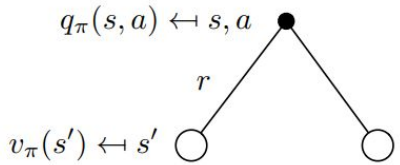
$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P})\, v &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

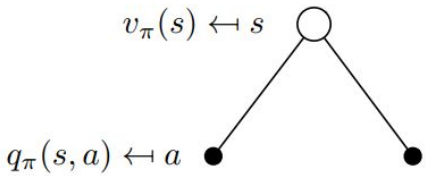
(w.r.t. policy π)

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

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

Conversions





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

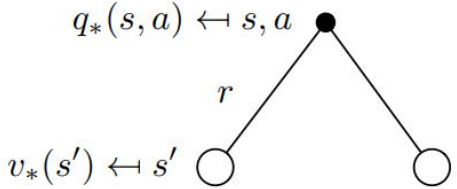
$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$ random

$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s')$

Optimal Value and Policy Functions

- Optimal value → one which yields maximum value.

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



- Optimal policy → one which results in optimal value function.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

Value Iteration

- Use Bellman equation as an iterative update:

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i\left(s', a'\right) | s, a\right]$$

- Qi will converge to Q* as i → infinity.
- Convergence in value means convergence in policy, vice versa not true.
  - REASON : Multiple reward/value structures can cause the same policy.
- Both algorithms (value & policy) have theoretical guarantees of convergence.
- Policy Iteration is expected to be faster.

Model-based Methods

Policy Evaluation & Iteration

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

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[r(s, \pi(s), s') + \gamma v(s') \Big]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

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

Figure 4.3: Policy iteration (using iterative policy evaluation) for v_*. This algorithm has a subtle bug, in that it may never terminate if the policy continually switches between two or more policies that are equally good. The bug can be fixed by adding additional flags, but it makes the pseudocode so ugly that it is not worth it. :-)

Value Iteration

finding optimal value function

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
      $temp \leftarrow v(s)$
      $v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v(s')]$
      $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \gamma v(s') \Big]$

Figure 4.5: Value iteration.

one policy update (extract policy from the optimal value function

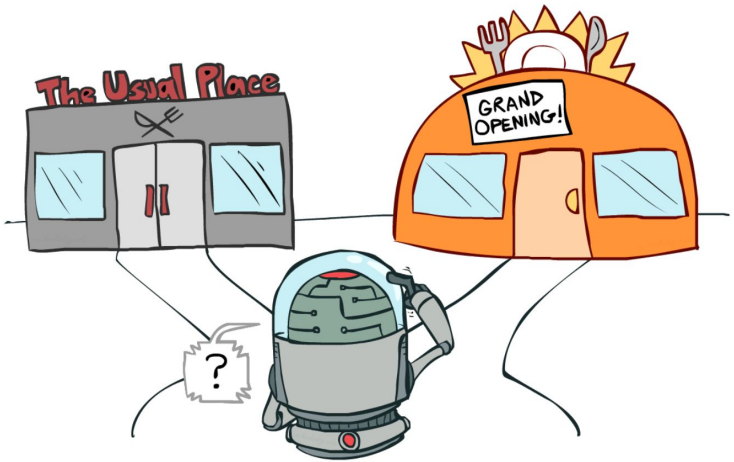
Model Free Methods

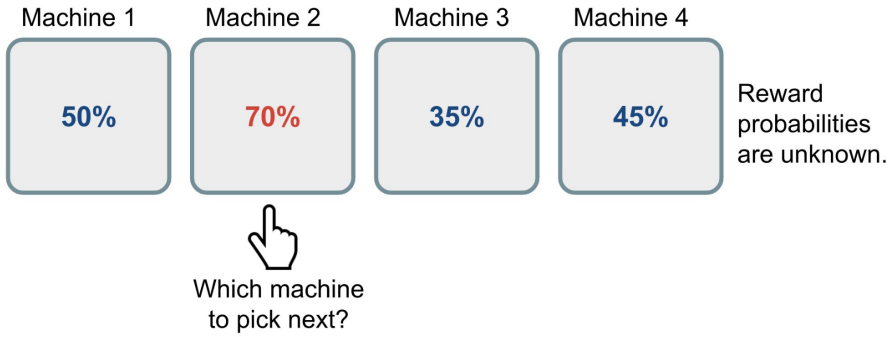
1. Learning or providing a transition model can be hard in several scenarios.
   a. Autonomous Driving
   b. ICU Treatments
   c. Stock Trading
2. **What do we have then ?** → episodic realizations
   (s,a,r,s')
3. **E.g.** Using sensors to understand robot's new position when it does an action, Recording new patient vitals when given a drug from a state etc.

Exploration-Exploitation tradeoff

- Exploitation: stick with what you know at risk of missing out.
- Exploration: look for states w/ more reward at risk of wasting time.

  - If you need to learn, you can't exploit all the time;
  - if you need to do well, you can't explore all the time





Action Selection Algorithms

Input: Q function, current state **s**

Greedy Algorithm:

- $a \in \arg\max Q_\pi(s, a)$

ϵ-Greedy Algorithm:

$\epsilon \in (0, 1]$

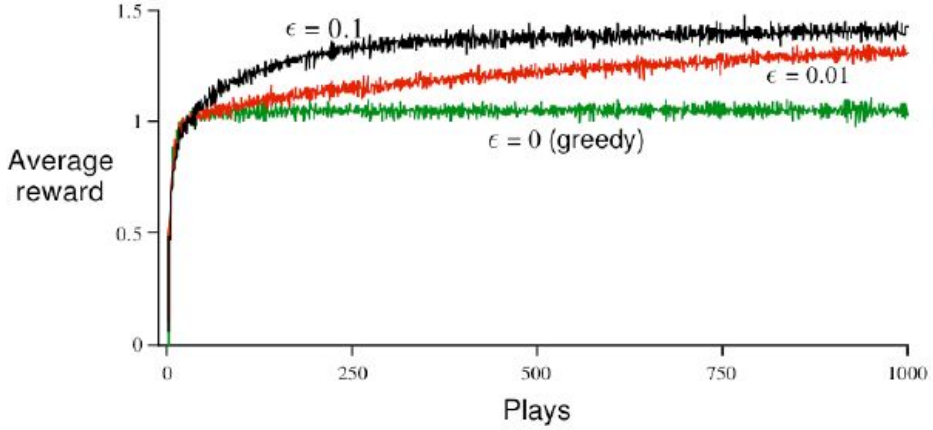
- With probability ϵ :
  a. Return random action **a**.
- With probability 1- ϵ:
  a. Return $a \in \arg\max Q_\pi(s, a)$

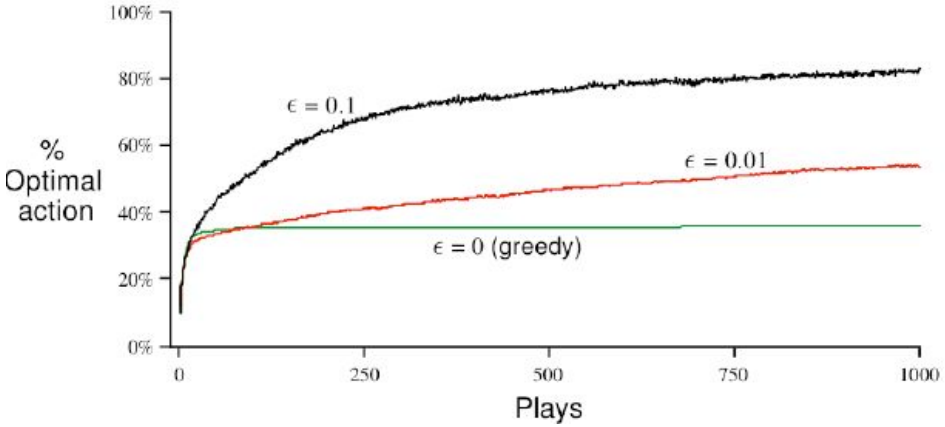
Softmax Action Selection:

- Probabilities $\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$

10-Armed Testbed

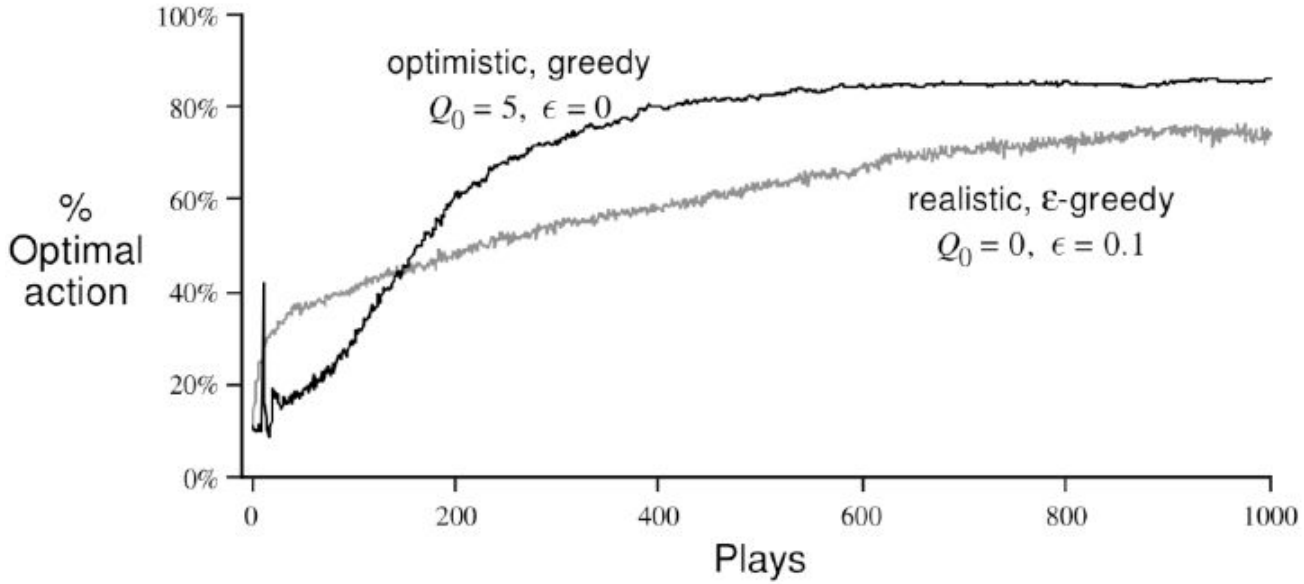
- n = 10 possible actions
- Each is chosen randomly from a normal distribution with mean 0 and variance 1.
- Each **r** is also normal, with mean Q*(a) and variance 1.
- 1000 plays.
- repeat the whole thing 2000 times and average results.
- Use sample average to estimate Q





Optimistic Initial Values

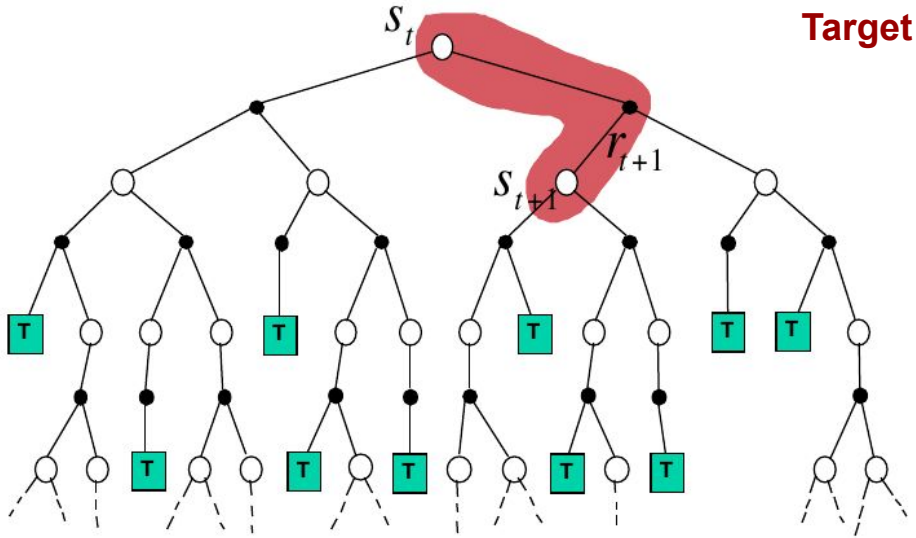
- All methods so far depend on $Q_0(a)$, i.e., they are biased.
- Suppose instead we initialize the action values optimistically:



Temporal Difference (TD) Learning

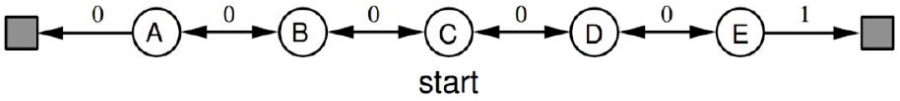
- Simplest TD method, TD(0):

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\left[\underbrace{r_{t+1} + \gamma V_\pi(s_{t+1})}_{\textbf{Target}\text{ (estimate of return)}} - V_\pi(s_t)\right]$$

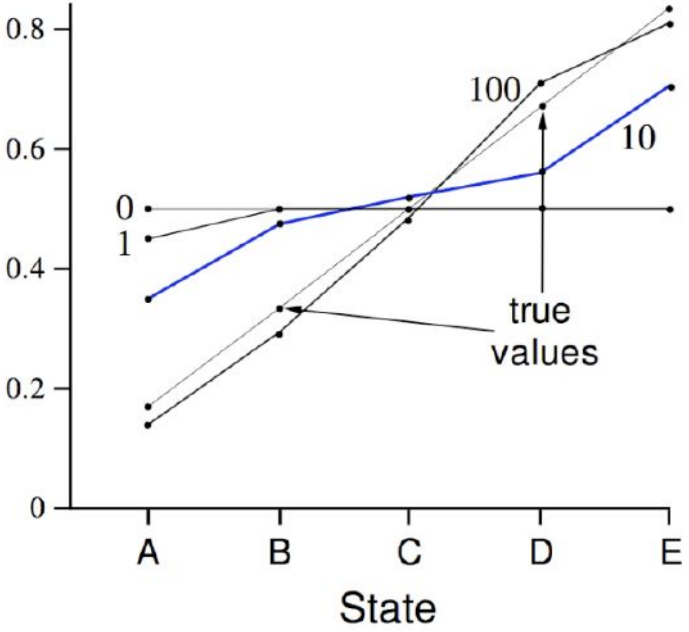


"Model-free policy estimation"

Random Walk in Hallway



Values learned by TD(0) after various numbers of episodes

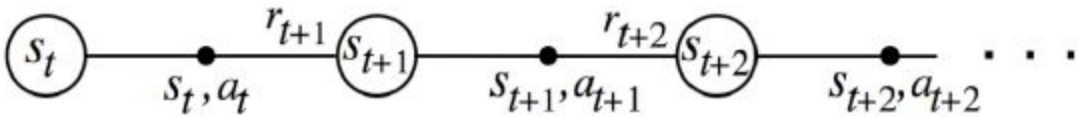


Advantages of TD Learning

- TD methods do not require a model of the environment, only experience.
- TD methods can be fully incremental
  - You can learn before knowing the final outcome (less memory req., less peak comp.).
  - You can learn without the final outcome.
- TD converges to an optimal policy:
  - For any finite Markov prediction task, under batch updating, TD(0) converges for sufficiently small α.

➢ How do we learn a policy ?

Evaluating the State-Action Value Function



- Substitute state value function **V** with **Q** state-action value function:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha \underbrace{[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})}_{\text{Target}} - Q_\pi(s_t, a_t)]$$

Target (estimate of return)

- If we choose $a_{t+1} = Q(s_{t+1}, \pi(s_{t+1}))$, same equation as before, TD(0).

On-Policy vs Off-Policy Learning

- On-Policy Learning
  - Learn on the job.
  - Evaluate policy π when sampling experiences from π.

- Off-Policy Learning
  - Look over someone's shoulder.
  - Evaluate policy π_a (target policy) while following a different policy π_b (behavior policy) in the environment.

Some domains prohibit on-policy learning. For instance, treating a patient in ICUs you cannot learn about random actions by testing them out.

SARSA: On-policy **TD Control**

- Learn while following current control policy (e.g., ϵ-greedy):

$$Q(s_i, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

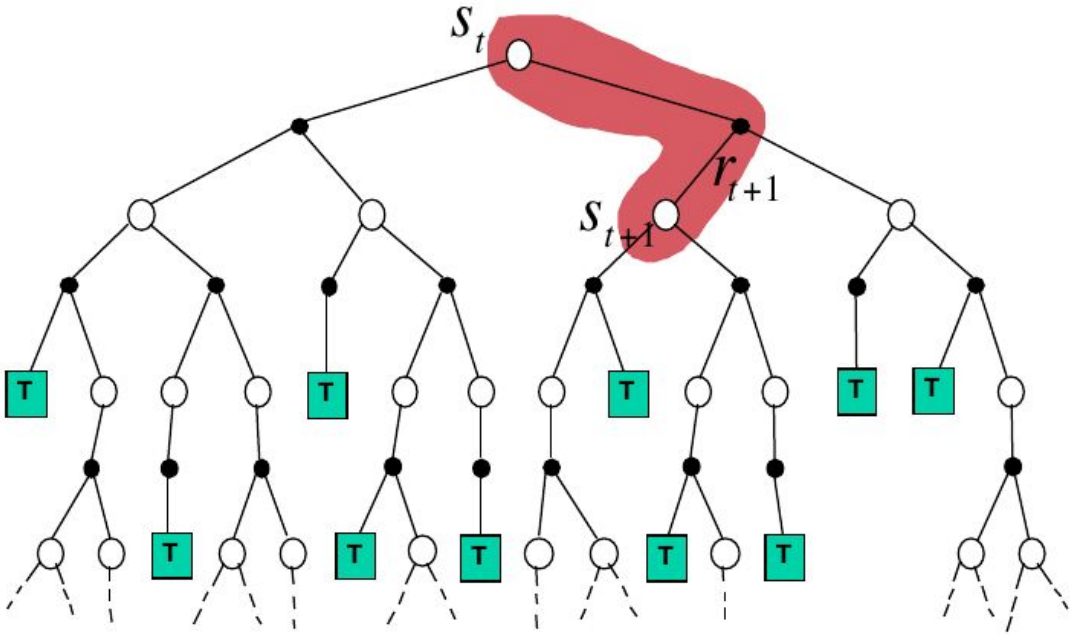
Q Learning: Off-policy **TD Control**

- One-step update: select best action for error update:

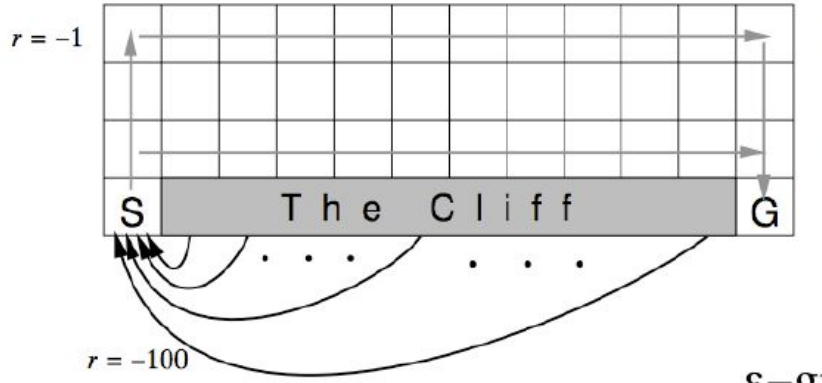
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    until s is terminal

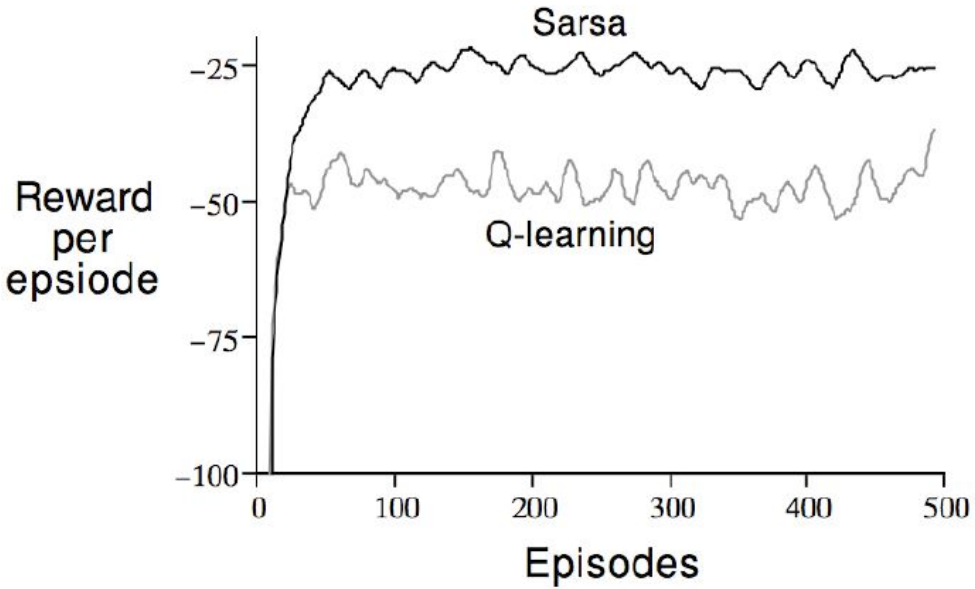
Visualization



Cliff-walking



ε−greedy, ε = 0.1



When do we choose SARSA vs. Q Learning ?

1. Q-learning directly learns the optimal policy, whilst SARSA learns a near-optimal policy whilst exploring.
2. If you want to learn an optimal policy using SARSA, then you will need to decide on a strategy to decay ϵ in ϵ-greedy action choice.
3. Q-learning (and off-policy learning in general) has higher per-sample variance than SARSA, and may suffer from problems converging as a result.
4. SARSA will approach convergence allowing for possible penalties from exploratory moves, whilst Q-learning will ignore them. That makes SARSA more conservative.

https://stats.stackexchange.com/questions/326788/when-to-choose-sarsa-vs-q-learning

Approximate Q-Learning

1. Basic Q-Learning keeps a table of all q-values.
2. **Not scalable** → must compute Q(s,a) for every state-action pair.
   a. E.g. current game state pixels: computationally infeasible to compute for entire state space!
3. **Solution** → use a function approximator to estimate Q(s,a).
   a. E.g. a neural network!

Parametric Q learning $Q(s, a; \theta) \approx Q^*(s, a)$

➢ In standard q-learning, we cannot extrapolate unexplored states

Linear Combination of Features

- Parametrize value functions using linear estimators:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Experience is summarized in finite length memory →
  - Continuous state/action values.
- States sharing features may have very different values.

Approximate Q Learning

1. Initialize Q(s, a)=0 for all s,a.
2. Repeat until convergence:
   a. s ← current state.
   b. a ← choose action (e.g. ϵ-greedy).
   c. s_{t+1}, r_{t+1} ← move using action a.
   d. Update estimated Q:

$$\mathrm{Q}(s, \mathrm{a}) \leftarrow \mathrm{Q}(\mathrm{s}, \mathrm{a}) + \alpha([\mathrm{R}(\mathrm{s}, \mathrm{a}, \mathrm{s}') + \gamma \mathrm{max}_{\mathrm{a}'} \mathrm{Q}(\mathrm{s}', \mathrm{a}')] - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$

Remark: parametrization of Q depends on w:

Deep Q Learning

- We want to learn optimal policy Q: $Q(s,a;\theta) \approx Q^*(s,a)$
- Identify loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \,|\, s,a\right]$$

Iteratively try to make the Q-value close to the target value y_i it should have, if Q-function corresponded to optimal Q* (and optimal policy π*)

- Gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\, \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

NN backprop

Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated → inefficient learning
  - Current Q-network parameters determines next training samples → bad feedback loops if diversity of examples is not well balanced.

- **Experience replay:**
  - Continually update a replay memory of transitions (st, at, rt, st+1) as episodes happen.
  - Train Q-network on random mini-batches from the replay memory, instead of consecutive samples.

DQN:

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Algorithm 1 Deep Q-learning with Experience Replay

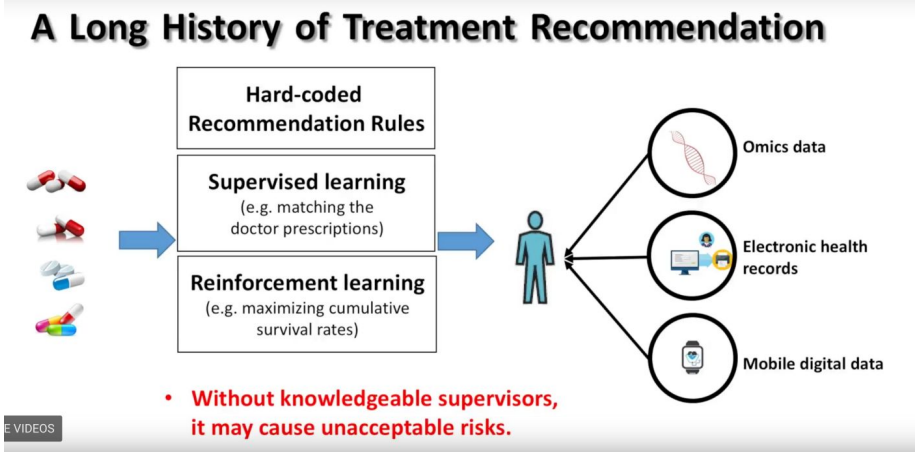
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

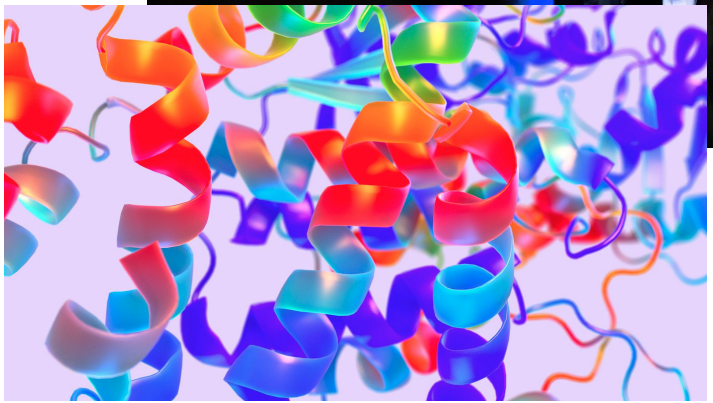
Deep RL General Techniques

- DQN requires a discrete set of actions (to retrieve policy).
- How to generalize to continuous action spaces ?
  - Example: a robot grasping an object has a very high-dimensional state → hard to learn exact value of every (state, action) pair.
  - Define a parametric class of policies (e.g., NN) & Q function.
  - 2xNNs: actor (policy) & critic (Q function).
- Slow learning → duplicate NNs (learner, and target).
- ...

Thank you!

Any Questions ?





References & Credits

1. “Reinforcement learning: An Introduction”, Sutton and Barto, 2018.
2. Slides 1, Slides 2, by Richard Sutton, mod by Dan Lizotte.
3. What is the difference between value iteration and policy iteration? link
4. When to choose SARSA vs. Q-learning ? link.
5. Reinforcement Learning: Markov Decision Process: part 1, part 2.
6. TD in Reinforcement Learning: link.
7. Monte Carlo in Reinforcement Learning: link.
8. Deep Reinforcement Learning class from Stanford: link.
9. OpenAI: spinning-up.