# CSC2541: Deep Reinforcement Learning

## Lecture 2: Markov Decision Processes

Slides borrowed from David Silver, Pieter Abbeel

Jimmy Ba

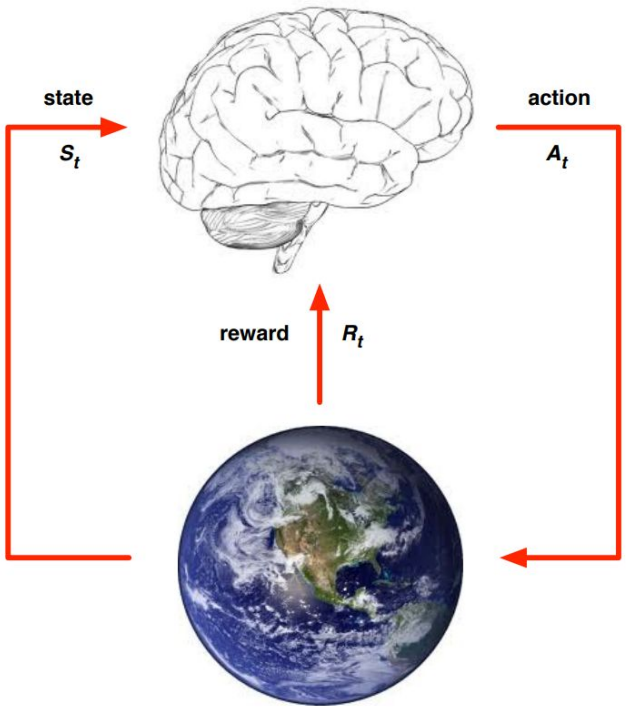
UNIVERSITY OF TORONTO

# Reinforcement learning

Learning to act through trial and error:

- An agent interacts with an environment and learns by maximizing a scalar reward signal.
- No models, labels, demonstrations, or any other human-provided supervision signal.
- Feedback is delayed, not instantaneous
- Agent’s actions affect the subsequent data it receives (data not i.i.d.)

# Fully observable environments




- The agent directly observe the **environment state** S^e_t.
  - O_t = S_t = S^e_t
  - And environment state is Markov
- Formally this turns into a **Markov Decision Process (MDP)**.

# Policy

- A policy $\pi$ is the agent's behaviour.
- It maps from the agent's state space to the action space.
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(s) = P(A_t = a | S_t = s)$

# Value function

- Value function is a prediction of the future reward.
- Used to evaluate the goodness/badness of the state.
  - $v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s \right]$
- We can use value function to choose the actions.

# Model

- A model predicts what will happen next in the environment.
  - **Dynamics model** predicts the next state given the current state and the action.
  - **Reward model** predicts the immediate reward given the state and the action.

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

# Outline

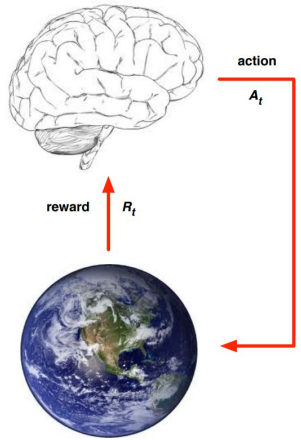
- Simple problem: one-step decision making, multi-armed bandit
- Markov decision processes with known models
    a. Finite discrete MDPs
    b. Gaussian continuous MDPs
- MDPs with unknown models (next lecture)
    a. Monte Carlo methods
    b. TD learning

# Simple problem: multi-armed bandit

- Imagining a gambler at a row of slot machines (sometimes known as "one-armed bandits"), who has to decide **which** machines to play, **how many times** to play each machine and in which order to play them, and whether to continue with the current machine or try a different machine.

# Simpler problem: multi-armed bandit with known model

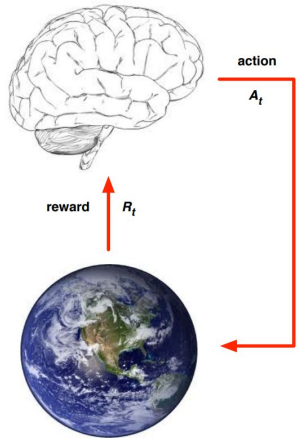
- A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
- A is a known set of k actions (or “arms”)
- $\mathcal{R}^a(r) = \mathbb{P}[r|a]$ is an known probability distribution over rewards
- The agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- The goal is to maximize reward.




What is the optimal policy?

# Simpler problem: multi-armed bandit with known model

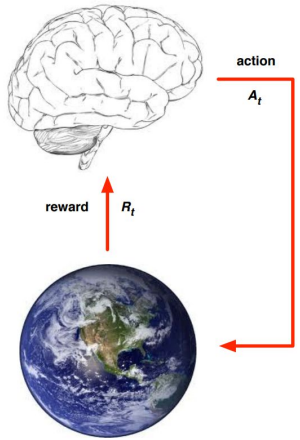
- A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
- A is a known set of k actions (or “arms”)
- $\mathcal{R}^a(r) = \mathbb{P}[r|a]$ is an known probability distribution over rewards
- The agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- The goal is to maximize reward.




What is the optimal policy? $\pi = \arg\max_a \mathbb{E}_{\mathbb{P}[r|a]}[r] = \arg\max_a \int_r r\mathbb{P}[r|a]dr$

# Simple problem: multi-armed bandit

- A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
- A is a known set of k actions (or “arms”)
- $\mathcal{R}^a(r) = \mathbb{P}[r|a]$ is an **unknown** probability distribution over rewards
- At each step t the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- The goal is to maximize reward **from its experiences of the environment without losing too much reward along the way**.



# Simple problem: multi-armed bandit

- Wikipedia trivia: [The problem] Originally considered by Allied scientists in World War II, it proved so intractable that, according to Peter Whittle, the problem was proposed to be dropped over Germany so that German scientists "could also waste their time on it”

# Regret

- We need to formally quantify the “loss of reward along the way” as an objective to derive learning algorithms
- The action-value is the mean reward for an action a $Q(a) = \mathbb{E}[r|a]$
- The optimal value V* is $V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a)$
- The regret is the opportunity loss for one step $l_t = \mathbb{E}[V^* - Q(a_t)]$
- The total regret is the total opportunity loss $L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right]$
- Maximise cumulative reward ≡ minimise total regret

# Counting regret

- The count N_t(a) is expected number of selections for action
- The gap ∆a is the difference in value between action a and optimal action a* $\Delta_a = V^* - Q(a)$
- Regret is a function of gaps and the counts

$$
\begin{aligned}
L_t &= \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right] \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right] \left(V^* - Q(a)\right) \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right] \Delta_a
\end{aligned}
$$

- A good algorithm ensures small counts for large gaps
- Problem: gaps are not known

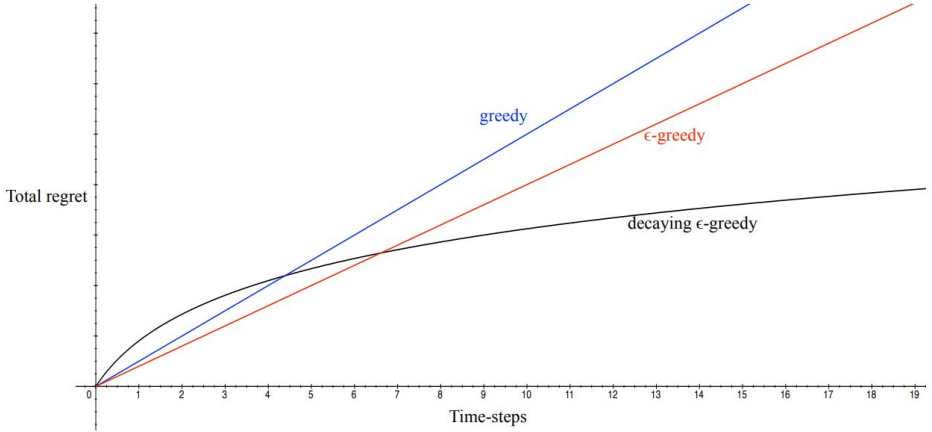
# Learning

- To solve multi-armed bandit, one needs to learn about the reward model
- A simple model only consider the mean reward/value for each action $\hat{Q}_t(a) \approx Q(a)$
- Learning the value of each action by **Monte-Carlo** evaluation

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t \mathbf{1}(a_t = a)$$

- Learning requires a training set
- How to generate the training data?

# Linear vs sublinear regret




- If an algorithm forever explores it will have linear total regret
- If an algorithm never explores it will have linear total regret
- Can we have sublinear total regret?

# Greedy algorithm

- We consider algorithms that estimate $\hat{Q}_t(a) \approx Q(a)$
- Estimate the value of each action by **Monte-Carlo** evaluation

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t \mathbf{1}(a_t = a)$$

- The greedy algorithm selects action with highest value $a_t^* = \underset{a \in \mathcal{A}}{\text{argmax}} \hat{Q}_t(a)$
- Greedy can lock onto a suboptimal action forever
- Greedy has linear total regret

# \epsilon-greedy algorithm

- We can have a mixture policy between exploration and greedy
- The \epsilon-greedy algorithm continues to explore with probability \epsilon
  a. With probability 1 − \epsilon select $a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \hat{Q}(a)$
  b. With probability \epsilon select a random action
- Constant \epsilon ensures minimum regret $l_t \geq \frac{\epsilon}{\mathcal{A}} \sum_{a \in \mathcal{A}} \Delta_a$
- \epsilon-greedy has linear total regret

# \epsilon-greedy algorithm

**A simple bandit algorithm**

Initialize, for $a = 1$ to $k$:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

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

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

# A reasonable heuristic: optimistic initialization

- Simple and practical idea: initialise Q(a) to high value
- Update action value by incremental Monte-Carlo evaluation
- Starting with N(a) > 0
- Encourages systematic exploration early on

Linear or sublinear regret?

# A reasonable heuristic: optimistic initialization

- Simple and practical idea: initialise Q(a) to high value
- Update action value by incremental Monte-Carlo evaluation
- Starting with N(a) > 0
- Encourages systematic exploration early on
- Can still stuck in suboptimal actions
    a. greedy + optimistic initialization has linear total regret
    b. \epsilon-greedy + optimistic initialization has linear total regret

# Decaying \epsilon-greedy

- Pick a decay schedule for \epsilon_1, \epsilon_2, \epsilon_3,...
- Consider the following schedule:

$$c > 0$$
$$d = \min_{a|\Delta_a > 0} \Delta_i$$
$$\epsilon_t = \min \left\{ 1, \frac{c|\mathcal{A}|}{d^2 t} \right\}$$

- Decaying \epsilon_t -greedy has logarithmic asymptotic total regret
- Unfortunately, schedule requires advance knowledge of gaps
- Find an algorithm with sublinear regret without knowing the gap

# Lower-bound: whats the best we can do

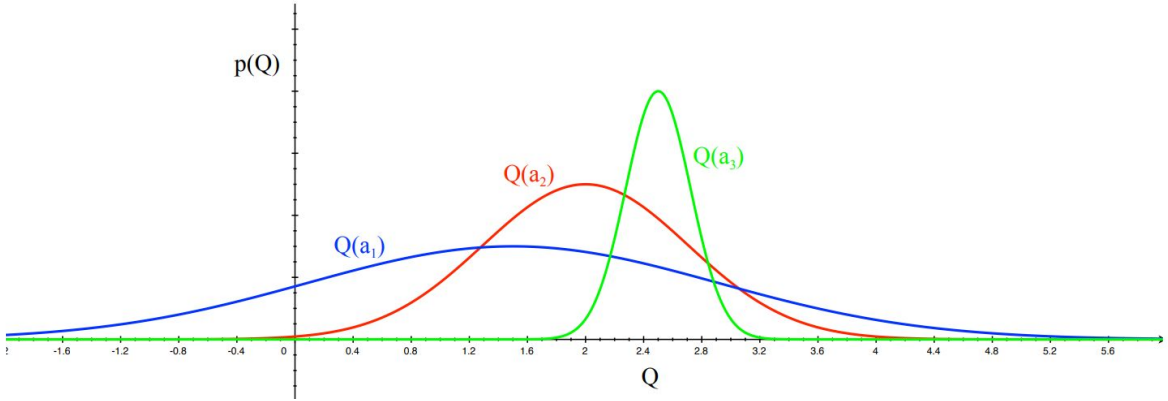
- The performance of any algorithm is determined by similarity between optimal arm and other arms
- Hard problems have similar-looking arms with different means
- This is described formally by the gap $\Delta a$ and the similarity in distributions between the estimated reward model and the ground truth reward $KL(\mathcal{R}^a||\mathcal{R}^a*)$

**Theorem (Lai and Robbins)**

*Asymptotic total regret is at least logarithmic in number of steps*

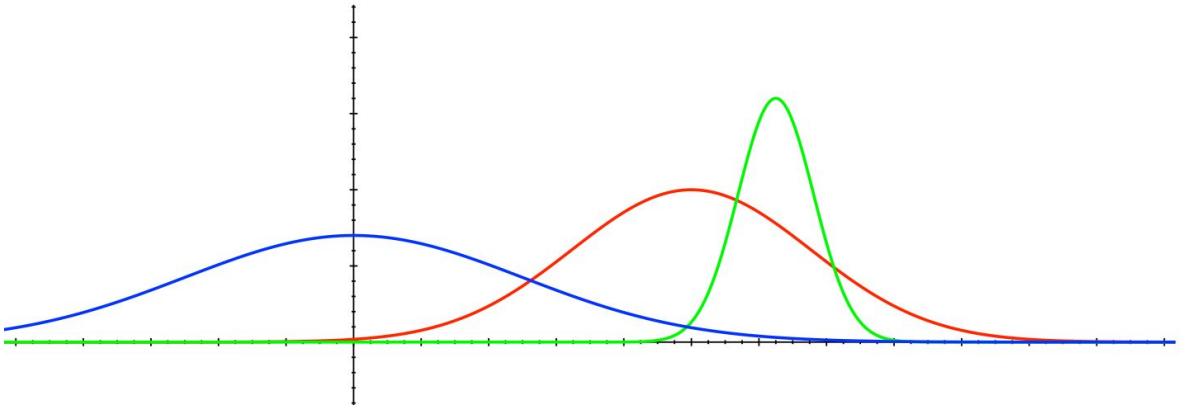
$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{KL(\mathcal{R}^a||\mathcal{R}^{a^*})}$$

# A better heuristic: optimistic about uncertainty




- Which action should we pick
- The more uncertain we are about an action-value
- The more important it is to explore that action
- It could turn out to be the best action

# A better heuristic: optimistic about uncertainty



- After picking blue action
- We are less uncertain about the value
- And more likely to pick another action
- Until we home in on best action

# Upper Confidence Bounds

- Estimate an upper confidence U^t(a) for each action value
- Such that with high probability $Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$
- This depends on the number of times N(a) has been selected
    a. Small Nt(a) ⇒ large Uˆt(a) (estimated value is uncertain)
    b. Large Nt(a) ⇒ small Uˆt(a) (estimated value is accurate)
- Select action maximising Upper Confidence Bound (UCB)

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \hat{Q}_t(a) + \hat{U}_t(a)$$

# Derive UCB

**Theorem (Hoeffding's Inequality)**

*Let $X_1, ..., X_t$ be i.i.d. random variables in [0,1], and let $\overline{X}_t = \frac{1}{\tau}\sum_{\tau=1}^{t} X_\tau$ be the sample mean. Then*

$$\mathbb{P}\left[\mathbb{E}[X] > \overline{X}_t + u\right] \leq e^{-2tu^2}$$

- We will apply Hoeffding's Inequality to rewards of the bandit conditioned on selecting action a

$$\mathbb{P}\left[Q(a) > \hat{Q}_t(a) + U_t(a)\right] \leq e^{-2N_t(a)U_t(a)^2}$$

# Derive UCB

- Pick a probability p, how optimistic we would like to be
- Now solve for Ut(a) $e^{-2N_t(a)U_t(a)^2} = p$

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- Reduce p as we observe more rewards, e.g. p = t^−4. I.e. our optimism decays as we have collected more data.
- To ensure we select optimal action as t -> \infty $U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$

# UCB1

- Pick action according to

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

**Theorem**

*The UCB algorithm achieves logarithmic asymptotic total regret*

$$\lim_{t \to \infty} L_t \leq 8 \log t \sum_{a | \Delta_a > 0} \Delta_a$$

# Bayesian bandits

- So far we have made no assumptions about the reward distribution R
    a. Except bounds on rewards
- Bayesian bandits exploit prior knowledge of reward dist. p[R]
- They compute posterior distribution of rewards given the history p[R | h_t]
- Use posterior to guide exploration
    a. Upper confidence bounds (Bayesian UCB)
    b. Probability matching (Thompson sampling)
- Better performance if prior knowledge is accurate

# Independent Gaussian Bayesian UCB

- Assume reward distribution is Gaussian $\mathcal{R}_a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2)$
- Compute Gaussian posterior over the parameters of the reward dist.

$$p\left[\mu_a, \sigma_a^2 \mid h_t\right] \propto p\left[\mu_a, \sigma_a^2\right] \prod_{t \mid a_t = a} \mathcal{N}(r_t; \mu_a, \sigma_a^2)$$

- Pick action that has UCB proportional to the standard deviation of Q(a)

$$a_t = \underset{a}{\text{argmax}}\ \mu_a + c\sigma_a / \sqrt{N(a)}$$

# Probability Matching

- Probability matching selects action a according to probability that a is the optimal action

$$\pi(a \mid h_t) = \mathbb{P}\left[Q(a) > Q(a'), \forall a' \neq a \mid h_t\right]$$

- Probability matching is optimistic in the face of uncertainty
- Can be difficult to compute analytically from posterior

# Thompson sampling

- Thompson sampling implements probability matching

$$\pi(a \mid h_t) = \mathbb{P}\left[Q(a) > Q(a'), \forall a' \neq a \mid h_t\right]$$
$$= \mathbb{E}_{\mathcal{R}|h_t}\left[\mathbf{1}(a = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(a))\right]$$

- Use Bayes rule to compute posterior distribution $p\left[\mathcal{R} \mid h_t\right]$
- Sample a reward distribution R from posterior
- Compute action-value function over the sampled reward
- Select action that gives the maximum action-value on the sampled reward
- Thompson sampling achieves Lai and Robbins lower bound!

# Value of information

- Exploration is useful because it gains information
- Can we quantify the value of information?
    a. How much reward a decision-maker would be prepared to pay in order to have that information, prior to making a decision
    b. Long-term reward after getting information - immediate reward
- Information gain is higher in uncertain situations
- Therefore it makes sense to explore uncertain situations more
- Knowing value of information allows trade-off exploration and exploitation optimally

# Application of bandits

- Recommended systems
- Ads
- Clinical trials
- Experimental design
- Hyperparameter tuning
- Resource allocation

# Harder problem: Markov decision process

- A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability function $\mathcal{P}^a_{s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma$ is a discount factor $\gamma \in [0, 1]$
- Goal is to find the optimal policy that maximize the total discounted future return

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

# Discounted reward

**Definition**

The *return* $G_t$ is the total discounted reward from time-step $t$.

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

- The objective in RL is to maximize long-term future reward
- That is, to choose a_t so as to maximize R_{t+1}, R_{t+2}, R_{t+3}...
    a. Episodic tasks - finite horizon
    b. continuous tasks - infinite horizon

# Discounted reward

**Definition**

The *return* $G_t$ is the total discounted reward from time-step $t$.

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

- Why discounted reward?

# Discounted reward

**Definition**

The *return* $G_t$ is the total discounted reward from time-step $t$.

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

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- Animal/human behaviour shows preference for immediate reward
- It is possible to use undiscounted Markov reward processes if all sequences terminate.

# Value functions in MDP

**Definition**

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

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

**Definition**

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

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

# Markov decision process

- Notice that the value function can be decomposed into two parts:
  a. immediate reward R_{t+1}
  b. discounted future reward $\gamma v_\pi(S_{t+1})$

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

- Also the states in an MDP are Markov ie:
  a. P(S_{t+1} | S_t) = P(S_{t+1} | S_1, S_2, …, S_t)

# Markov decision process

- Optimal substructure
    a. Principle of optimality applies
    b. Optimal solution can be decomposed into subproblems
- Overlapping subproblems
    a. Subproblems recur many times
    b. Solutions can be cached and reused
- Markov decision processes satisfy both properties
    a. Value function stores and reuses solutions

# Bellman expectation equation

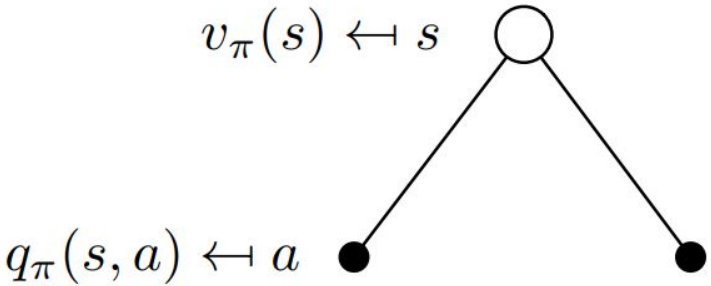
- Bellman equation gives recursive decomposition of the sub-solutions in an MDP
- The state-value function can be decomposed into immediate reward plus discounted value of successor state.

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

- The action-value function can similarly be decomposed.

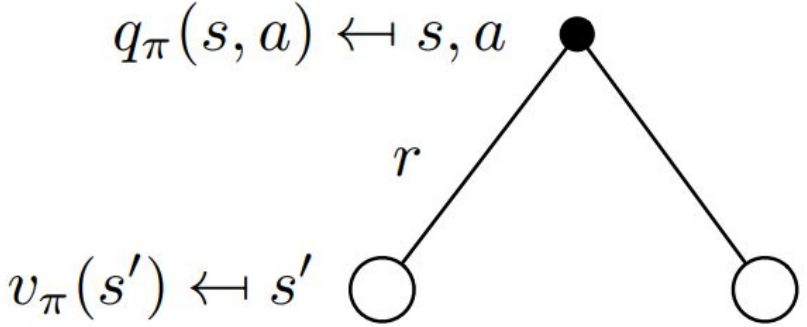
$$q_\pi(s, a) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

# Bellman expectation equation



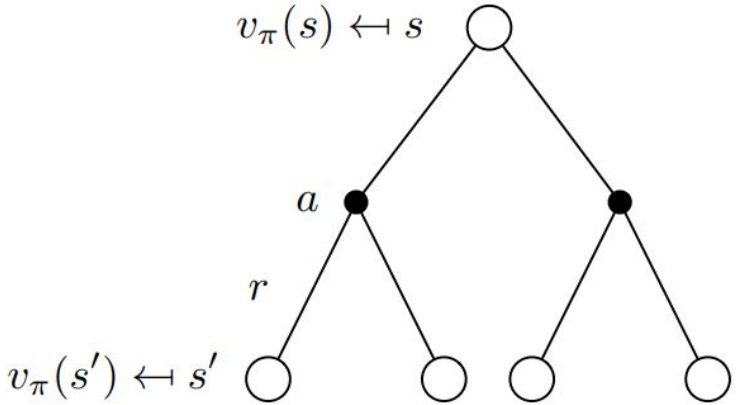

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

# Bellman expectation equation



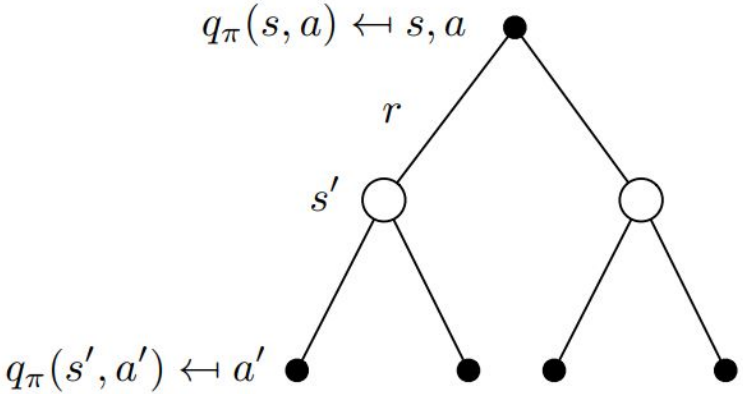

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

# Bellman expectation equation




$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

# Bellman expectation equation




$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

# Optimal value function

- Goal is to find the optimal policy that maximize the total discounted future return $\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

**Definition**

The *optimal state-value function* $v_*(s)$ is the maximum value function 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

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

- The optimal value function specifies the best possible performance in the MDP.

# Optimal policy

- Define a partial ordering over policies $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s$

**Theorem**

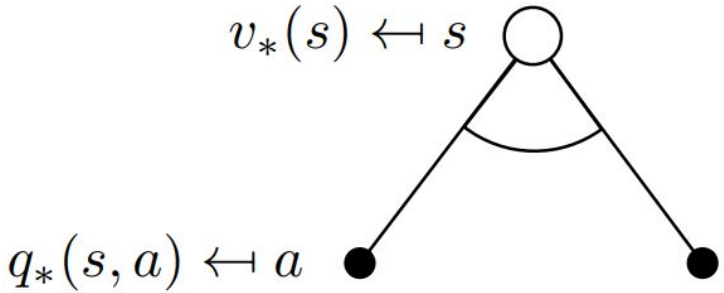
*For any Markov Decision Process*

- *There exists an optimal policy $\pi_*$ that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

- An optimal policy can be found by maximising over q*(s, a)
- If we know q*(s, a), we immediately have the optimal policy

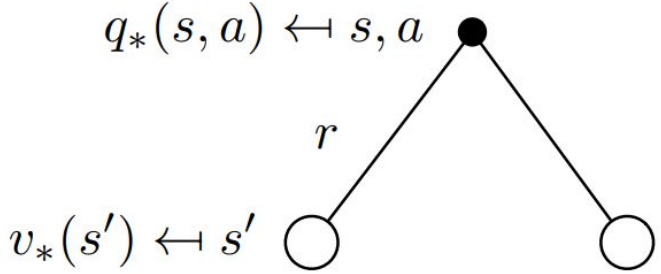
# Bellman optimality equation

- The optimal value functions are recursively related by the Bellman optimality equations:



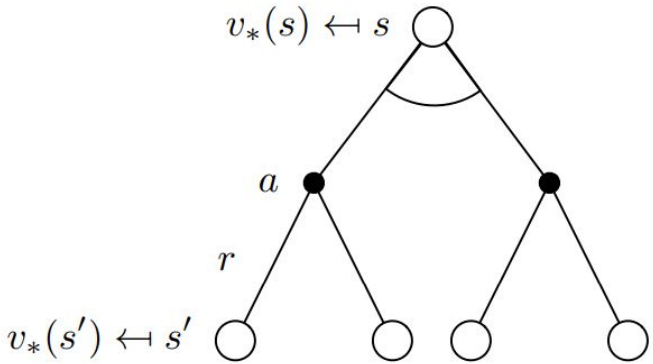

$$v_*(s) = \max_a q_*(s,a)$$

# Bellman optimality equation



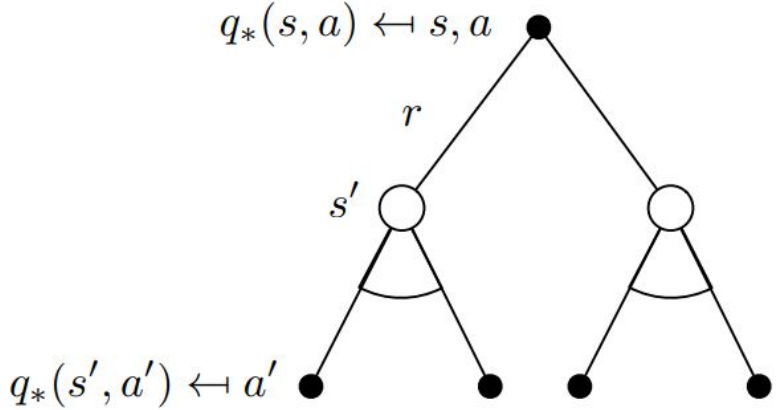

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

# Bellman optimality equation




$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

# Bellman optimality equation




$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

# Discrete MDP with known model

- A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a **state transition probability function (known)** $\mathcal{P}^a_{s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a **reward function (known)** $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma$ is a discount factor $\gamma \in [0, 1]$
- Goal is to find the optimal policy that maximize the total discounted future return

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

# Discrete MDP with known model

- Bellman Optimality Equation is non-linear.
    a. No closed form solution (in general)
- Many methods to solve discrete MDP with known model:
    a. Dynamic programming
        i. Value Iteration
        ii. Policy Iteration
    b. Monte-Carlo
    c. Temporal-Difference learning

# Planning by dynamic programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- For prediction:
    a. Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
    b. Output: value function v
- Or for control:
    a. Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
    b. Output: optimal value function / policy

# Planning by dynamic programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- For prediction:
    a. Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
    b. Output: value function v
- Or for control:
    a. Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
    b. Output: optimal value function / policy

# Value iteration

- If we know the solution to subproblems v*(s')
- Then solution v*(s) can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs, analogous to Viterbi algorithm or max-product algorithm

# Parallel synchronous value iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For i = 1, … , H

For all states s in S:

$$v_{i+1}(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_i(s')$$

- The algorithm generate a sequence of value functions v_0,... v_H
- Intermediate value functions may not correspond to any policy

# Policy iteration

- If we are given a policy π, we can predict the value function of the given policy using Bellman expectation equation:
- $$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$
- The policy can be improved by acting greedy with respect to v_π
- $$\pi' = \text{greedy}(v_\pi) = \arg\max_{a \in A} \sum_{s'} \mathcal{P}(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s')\right]$$
- Intuition: iteratively improve the current policy, analogous to k-means
- The algorithm generate a sequence of value functions π_0,... π_k
- This process of policy iteration always converges to π*

# Principle of optimality

- Any optimal policy can be subdivided into two components:
  a. An optimal first action A*
  b. Followed by an optimal policy from successor state S’

**Theorem (Principle of Optimality)**

*A policy $\pi(a|s)$ achieves the optimal value from state $s$, $v_\pi(s) = v_*(s)$, if and only if*

- *For any state $s'$ reachable from $s$*
- *$\pi$ achieves the optimal value from state $s'$, $v_\pi(s') = v_*(s')$*

# Continuous MDP with known model

- A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a **state transition probability function (known)** $\mathcal{P}^a_{s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a **reward function (known)** $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma$ is a discount factor $\gamma \in [0, 1]$
- Goal is to find the optimal policy that maximize the total discounted future return

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

# Continuous MDP with known model

- Solve the continuous problem by discretization
- Bellman’s Curse of Dimensionality
    a. n-dimensional state space
    b. The number of states in a problem can grow exponentially in n
- In practice, only computationally feasible up to 5 or 6 dimensional state spaces

# Continuous MDP with known model

- Consider a special case: Linear Dynamical Systems and Quadratic Cost (aka LQR setting).
- We can actually solve continuous state-space optimal control problem exactly and only requires performing linear algebra operations.

# Linear Quadratic Regulator (LQR) assumptions

The LQR setting assumes a linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t,$$

$x_t$: state at time $t$
$u_t$: input at time $t$
It assumes a quadratic cost function:

$$g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$$

with $Q \succ 0, R \succ 0$.

For a square matrix $X$ we have $X \succ 0$ if and only if for all vectors $z$ we have $z^\top X z > 0$. Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.

- Where g is the negative reward or the cost

# LQR value iteration

- Back-up step for i+1 steps to go
- Value iteration

$$J_{i+1}(s) = \min_u g(s,u) + \sum_{s'} P(s'|s,u) J_i(s')$$

- LQR

$$J_{i+1}(x) = \min_u x^\top Q x + u^\top R u + \sum_{x'=Ax+Bu} J_i(x')$$

$$= \min_u \left[ x^\top Q x + u^\top R u + J_i(Ax+Bu) \right]$$

# LQR value iteration

- Solve for the one-step Bellman back-up by minimizing a quadratic function

$$J_{i+1}(x) \leftarrow \min_u \left[x^\top Q x + u^\top R u + J_i(Ax + Bu)\right]$$

Initialize $J_0(x) = x^\top P_0 x$.

$$\begin{aligned} J_1(x) &= \min_u \left[x^\top Q x + u^\top R u + J_0(Ax + Bu)\right] \\ &= \min_u \left[x^\top Q x + u^\top R u + (Ax + Bu)^\top P_0 (Ax + Bu)\right] \qquad (1) \end{aligned}$$

To find the minimum over $u$, we set the gradient w.r.t. $u$ equal to zero:

$$\nabla_u [\ldots] = 2Ru + 2B^\top P_0 (Ax + Bu) = 0,$$

$$\text{hence: } u = -(R + B^\top P_0 B)^{-1} B^\top P_0 A x \qquad (2)$$

$$\begin{aligned} \text{(2) into (1): } J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{aligned}$$

# LQR value iteration

- Given the quadratic approximation from the previous iteration

$$
\begin{aligned}
J_0(x) &= x^\top P_0 x \\
x_{t+1} &= Ax_t + Bu_t \\
g(x,u) &= u^\top R u + x^\top Q x
\end{aligned}
$$

- We can update the negative value function J as

$$
\begin{aligned}
J_1(x) &= x^\top P_1 x \\
\text{for: } P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\
K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A.
\end{aligned}
$$

# LQR for nonlinear system

Nonlinear system: $x_{t+1} = f(x_t, u_t)$

We can keep the system at the state $x^*$ iff

$$\exists u^* \text{s.t.} \quad x^* = f(x^*, u^*)$$

Linearizing the dynamics around $x^*$ gives:

$$x_{t+1} \approx f(x^*, u^*) + \underbrace{\frac{\partial f}{\partial x}(x^*, u^*)}_{A}(x_t - x^*) + \underbrace{\frac{\partial f}{\partial u}(x^*, u^*)}_{B}(u_t - u^*)$$

Equivalently:

$$x_{t+1} - x^* \approx A(x_t - x^*) + B(u_t - u^*)$$

Let $z_t = x_t - x^*$ , let $v_t = u_t - u^*$, then:

$$z_{t+1} = Az_t + Bv_t, \qquad \text{cost} = z_t^\top Q z_t + v_t^\top R v_t \quad \text{[=standard LQR]}$$

$$v_t = Kz_t \Rightarrow u_t - u^* = K(x_t - x^*) \Rightarrow u_t = u^* + K(x_t - x^*)$$

# LQR for general control problems

- The generic optimal control problem can be formulated as the following:

$$\begin{aligned} \min_u \quad & \sum_{t=0}^{H} g(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \end{aligned}$$

- f is the dynamic model

# Iterative LQR(iLQR)

Initialize the algorithm by picking either (a) A control policy $\pi^{(0)}$ or (b) A sequence of states $x_0^{(0)}, x_1^{(0)}, \ldots, x_H^{(0)}$ and control inputs $u_0^{(0)}, u_1^{(0)}, \ldots, u_H^{(0)}$. With initialization (a), start in Step (1). With initialization (b), start in Step (2).

Iterate the following:

(1) Execute the current policy $\pi^{(i)}$ and record the resulting state-input trajectory $x_0^{(i)}, u_0^{(i)}, x_1^{(i)}, u_1^{(i)}, \ldots, x_H^{(i)}, u_H^{(i)}$.

(2) Compute the LQ approximation of the optimal control problem around the obtained state-input trajectory by computing a first-order Taylor expansion of the dynamics model, and a second-order Taylor expansion of the cost function.

(3) Use the LQR back-ups to solve for the optimal control policy $\pi^{(i+1)}$ for the LQ approximation obtained in Step (2).

(4) Set $i = i + 1$ and go to Step (1).

# Summary

Multi-armed bandits

Finite MDPs

Large/infinite MDPs

Theoretically tractable

Theoretically intractable

# Logistics

Seminar papers

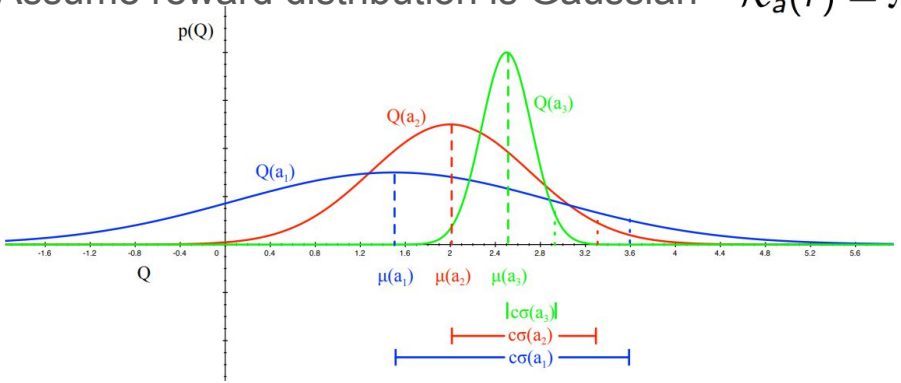
- Seminar papers will be announced at noon Fri
- A Google spreadsheet will be provided at the same time for the paper sign-up
- First come first serve policy

# Quiz

- What is the history in a multi-armed bandit problem?
- Is Bellman expectation equation linear in the value function? Why?
- Is Bellman optimality equation linear in the value function? Why?
- Show that the policy iteration algorithm always converges.
- What is the most computation intensive step within a single LQR backup step?

# Quiz: Independent Gaussian Bayesian UCB

- Assume reward distribution is Gaussian $\mathcal{R}_a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2)$




Which action will this Bayesian UCB algorithm pick? And why?

- Compute Gaussian posterior over the parameters of the reward dist.

$$p\left[\mu_a, \sigma_a^2 \mid h_t\right] \propto p\left[\mu_a, \sigma_a^2\right] \prod_{t \mid a_t = a} \mathcal{N}(r_t; \mu_a, \sigma_a^2)$$

- Pick action that has UCB proportional to the standard deviation of Q(a)

$$a_t = \underset{a}{\operatorname{argmax}}\ \mu_a + c\sigma_a / \sqrt{N(a)}$$