Reinforcement Learning (basics)
Categorization of Machine Learning: Type of Dataset

- **Supervised learning**: dataset with label = the intended output of function

- **Un/semi-supervised learning**: dataset with no/few labels

- **Reinforcement learning**: dataset with reward signal = (partial) evaluation of function, e.g., feel good = +1 and feel bad = -1
Reinforcement Learning

Wikipedia says:

- Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward

Speculative trivia:

- The term “reinforcement” may come from that in behavioral psychology

*reinforcement is a consequence applied that will strengthen an organism's future behavior whenever that behavior is preceded by a specific antecedent stimulus
Animal Psychology

Training animals:

- Positive reinforcements: pleasure and food
- Negative reinforcements: pain and hunger

*Reinforcement learning (RL) performs the same with computers!*
Outline

Today (basics)
- Markov Decision Process (MDP): basic model
- Planning: how to find optimal control given model
- Reinforcement learning (RL): how to learn the model
- RL with function approximation: how to design scalable RL

Next class (advances)
- Exploration vs. exploitation
- Advanced topics: Soft Actor-critic, meta RL, ...
Basic Framework of RL

- **Setting**: once agent takes an action based on state, it receives numerical reward and environment evolves to new state.

- **Goal**: learn to choose actions that maximize rewards.
Examples of RL

- Vehicle routing
- Helicopter control
- Game playing
- Self-driving car
- Computational finance
- Elevator scheduling
- Data center energy optimization
- ...
Example of RL: Vehicle Routing

- **Agent**: vehicle routing software
- **Environment**: stochastic demand
- **State**: vehicle location, capacity and depot requests
- **Action**: vehicle route
- **Reward**: negative of travel costs
Example of RL: Helicopter Control

- **Agent**: controller
- **Environment**: helicopter
- **State**: position, orientation, velocity, angular velocity, ...
  
  *altitude, power, direction of flight, tail rotor, rpm*
- **Action**: collective pitch, cyclic pitch, pedals, throttle, ...
- **Reward**: negative of deviation from desired trajectory

Helicopter demo [Andrew Ng, 08]
Example of RL: Game Playing

- **Agent**: player
- **Environment**: opponent
- **State**: configuration of stones on board
- **Action**: next location of stone
- **Reward**: +1 for win and -1 for lose

AlphaGo defeats the legendary professional Go player, Sedol Lee (2016)

- AlphaGo’s 37-th move in second round was critical but unexpected (a pro play wouldn’t play this: odds 0.01%)
- This shows advantage of RL over supervised learning as such creative move might not be possible under supervision of pro-players
Agent-Environment Interface

We consider **discrete time**, and at each step $t = 0, 1, 2, ...$,

- For given state $S_t$, the agent executes action $A_t \in \mathcal{A}$; receives reward $R_{t+1} \in \mathbb{R}$; and observes next state $S_{t+1}$ of environment.
Markov Decision Process (MDP)

An MDP is defined by a tuple $\langle S, A, R, p \rangle$:

- **State space** $S$, **action space** $A$, and **reward space** $R$ are the set of all possible states, actions, and rewards, respectively.

- **Markov property**: each $S_t \in S$ contains the information deciding what will happen at next $t + 1$ given action $A_t \in A$.

- **Kernel** $p : S \times R \times S \times A \rightarrow [0, 1]$ defines the conditional distribution of next state and reward given history $H_t := (S_0, A_0, R_1, S_1, A_1, \ldots, R_t, S_t, A_t)$:

  \[
  \mathbb{P}[S_{t+1}, R_{t+1} | H_t] = p(S_{t+1}, R_{t+1} | S_t, A_t),
  \]

- Current state and action **fully determine** the distribution of next state and reward.

*combined state transition and reward*
Markov Property

The same kernel at \( \text{\textbullet} \) and \( \text{\textbullet} \)

*transition kernel
Notation

- Transition probability $p(s' \mid s, a)$ such that
  \[ p(s' \mid s, a) := \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] . \]

- Mean reward $r(s, a)$ such that
  \[ r(s, a) := \mathbb{E}[R_{t+1} \mid S_t = s, A_0 = a] . \]

- A policy $\pi : S \rightarrow A$ is a mapping from states to action:
  - There can be random policy
  - Thanks to Markov property, i.e., current state and action describes dynamics of future, it is enough to consider control which maps from state and action
Policy Evaluation

A typical evaluation is the expectation of cumulated value of policy with some discount factor $\gamma \in [0, 1)$.

- **V-function/state-value function**

\[
V^\pi(s) := \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right]
\]

- The undiscounted case with $\gamma = 1$ requires somewhat non-trivial treatment due to the risk of diverging value.

- $\gamma$ quantifies the appreciation of future reward, c.f., inflation.
Policy Improvement

- **Q-function/action-value function**

\[ Q^\pi(s, a) := \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right] \]

- This is the value of policy \( \pi \) with a single deviation at the beginning

\[ R_1, R_2, R_3, \ldots \]

\[ \underbrace{a}_{\text{a}}, \underbrace{\pi}_{\text{\pi}} \]

- \((Q^\pi(s, a) - V^\pi(s))\) quantifies the improvement by the deviation, i.e., an improved policy \( \pi' \) can be obtained by

\[ \pi'(s) = \arg \max_a Q^\pi(s, a) \]
Optimal Policy

- Optimal policy $\pi^* := \arg\max_{\pi} V^\pi(s)$ has no possible improvement, i.e., $\pi$ is optimal if and only if
  
  $$\max_a \left( Q^\pi(s, a) - V^\pi(s) \right) = 0 \quad \forall s \in S$$

- **Policy iteration**: such an optimal policy can be computed by interleaving
  
  - **Policy evaluation** to compute $Q^\pi(s, a)$
  - **Policy improvement** to update $\pi(s) \leftarrow \arg\max_a Q^\pi(s, a)$

  *value iteration: optimal value function computing $V^\pi(s)$
Planning based on Dynamic Programming

How to compute the value functions? *Bellman equation

\[ Q^\pi(s, a) = \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right] \]

\[ = \mathbb{E}^\pi \left[ R_1 + \gamma Q^\pi(S_1, \pi(S_1)) \mid S_0 = s, A_0 = a \right] \]

\[ = r(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a) Q^\pi(s', \pi(s')) \]

\[ = r(s, a) + \gamma \sum_{s' \in S} \sum_{a' \in A} \mathbb{1}_{\pi(s') = a'} p(s' \mid s, a) Q^\pi(s', a') \]

i.e., computing \( Q^\pi \in \mathbb{R}^{\mid S \times A\mid} \) is solving linear equation:

\[ Q^\pi = R + P^\pi Q^\pi \]

where \( R \in \mathbb{R}^{\mid S \times A\mid} \) with \( R(s, a) = r(s, a) \) and \( P^\pi \in \mathbb{R}^{\mid S \times A\mid \times \mid S \times A\mid} \) with \( P^\pi(s, a, s', a') = \mathbb{1}_{\pi(s') = a'} p(s' \mid s, a) \)
Reinforcement Learning (RL)

When we do not know model dynamics $p$ and reward $r$, from some random observations, the optimal policy can be learned by (i) model-based approaches and (ii) model-free ones

$$ Q^\pi(s, a) = \mathbb{E}^\pi [R_1 + \gamma Q^\pi(S_1, \pi(S_1)) \mid S_0 = s, A_0 = a] $$

$$ = r(s, a) + \gamma \sum_{s' \in S} \sum_{a' \in A} \mathbb{1}_{\pi(s') = a'} p(s' \mid s, a) Q^\pi(s', a') $$
Model-based Approach

Model-based approach directly estimates model \((p\text{ and } r)\), and perform policy iteration:

\[
Q^\pi(s, a) = \mathbb{E}^\pi [R_1 + \gamma Q^\pi(S_1, \pi(S_1)) \mid S_0 = s, A_0 = a]
\]

\[
= r(s, a) + \gamma \sum_{s' \in S} \sum_{a' \in A} \mathbbm{1}_{\pi(s') = a'} p(s' \mid s, a) Q^\pi(s', a')
\]

\[
\approx \hat{r}(s, a) + \gamma \sum_{s' \in S} \sum_{a' \in A} \mathbbm{1}_{\pi(s') = a'} \hat{p}(s' \mid s, a) Q^\pi(s', a')
\]

*Count \(s'\&r\) for each \(s, a\) from experience (data) \((s,a,s',r)\)*
Model-free Approach

Model-free approach indirectly estimates model, e.g., computing $Q$ or $V$-functions

$$Q^\pi(s, a) - \mathbb{E}^\pi [R_1 + \gamma Q^\pi(S_1, \pi(S_1)) | S_0 = s, A_0 = a] = 0$$

$$\iff \min_Q \mathbb{E}_{(r,s') \sim p(\cdot, \cdot | s,a)} (Q(s, a) - (r + \gamma Q(s', \pi(s'))))^2$$

For each observation of $(s, a, r, a')$,

- One can update the evaluation of current policy $\pi$ with

  $$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha((r + \gamma Q(s', \pi(s')))) - Q(s, a))$$

  c.f., SARSA

*on-policy learning, care about performance during learning (e.g. expensive robot); for convergence, try some random policy under epsilon-greedy
Model-free Approach

*policy evaluation vs policy improvement

It is possible to evaluate and **improve** policy at the same time:

\[ Q^*(s, a) - \mathbb{E}^\pi \left[ R_1 + \max_{a'} \gamma Q^*(S_1, a') \mid S_0 = s, A_0 = a \right] = 0 \]

\[ \iff \min_Q \mathbb{E}_{(r, s') \sim p(\cdot, \cdot \mid s, a)} (Q(s, a) - (r + \gamma \max_{a'} Q(s', a')))^2 \]

For each observation of \((s, a, r, a')\),

▷ **Q-learning** evaluates and improves simultaneously current policy \(\pi\) by

*off-policy learning, switch over to greedy policy during learning, e.g. inexpensive game

\[ Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha ((r + \gamma \max_{a'} Q(s', a')) - Q(s, a)) \]

as a part of \(\min_Q \mathbb{E}_{(r, s') \sim p(\cdot, \cdot \mid s, a)} (Q(s, a) - (r + \gamma \max_{a'} Q(s', a')))^2 \)
Toward Scalable RL

It is often intractable and inefficient to track a full table of $p(s' | s, a)$ or $Q(s, a)$ in practice

▶ **Intractable** when the system size $S \times A$ is large, i.e., not scalable in terms of computation

▶ **Inefficient** as there is no generalization from $(s, a)$ to similar $(s', a')$, i.e., not scalable in terms of samples

AlphaGo (black) vs. Lee, Sedol (white)
Function approximation is widely used where $p$ or $Q$ are modeled with a number of parameters $\theta$, i.e., $p \approx p_\theta$ or $Q \approx Q_\theta$

- There are a number of candidate approximators: linear model, neural network model, nearest neighbor, ...

- When RL uses deep neural networks as function approximator, it is called deep RL
Deep Q-Learning

Deep Q-network [DQN] is neural network approximating Q-function and deep Q-learning aims at solving:

$$\min_{\theta} L(\theta) = \sum_{(s,a,r,s') \in \text{Experience}} \left( Q_\theta(s, a) - (r + \gamma \max_{a'} Q_\theta(s', a')) \right)^2$$

▶ To enhance stability, DQN uses several heuristics: experience replay, doubling techniques...

*using two q-network for for action selection and for action evaluation

*input feature

*output action

*[using a large buffer of our past experience and sample training data to avoid correlation between experiences

*iterate execution & training (update parameter)

Reinforcement Learning (advances)
Outline

Exploration vs. exploitation
- Multi-Armed Bandit (MAB) problem
- Formulation of regret minimization
- UCB (Upper Confidence Bound) algorithm

Advances in efficient exploration
- Typical exploration in RL
- Structured exploration: meta RL, hierarchical RL
Exploration and Exploitation

RL agent can actively or *adaptively collect* samples hence the trade-off between exploration and exploitation is a fundamental problem in RL:

- **Exploration**: try under-sampled options (policies/actions) as they can be better than the current best

- **Exploitation**: stick with the current best only if a certain confidence is guaranteed from sufficient evidence

*balancing reward maximization based on the knowledge already acquired with attempting new actions to further increase knowledge. This is known as the exploitation vs. exploration tradeoff.*
Multi-Armed Bandit (MAB) Problem

Suppose slot machines have not necessarily the same but initially unknown payoffs. Which one should we play at each turn in order to maximize rewards?
A Mathematical Model: Stochastic MAB

\( K \)-armed bandit problem

- Random variable \( X_{i,n} \in [0, 1] \) denotes arm \( i \)'s \( n \)-th reward (the bounded support is assumed w.o.l.g.)

- Assume that for \( n \geq 1 \), \( X_{i,n} \) is \textbf{independently} identically drawn from a distribution with expectation \( \mu_i \)

- Also, for given \( i \), \( X_{i,n} \)'s are assumed to be \textbf{independent} to each other

\[ \text{source: Microsoft Research} \]
Examples

The independence assumption implies that actions make no change in system state, i.e., MAB = an RL problem with single state

Real-world Applications

- Clinical trials to find cure for cancer
- Rate adaptation in wireless network
- Online ad placement or recommender system
Clinical Trials

To cure cancer:

- **Arms** = medicines
- **Reward** = reduction of cancer size
- c.f., best medicine may depend on cancer size, and there can be some correlation between successive treatments
  (MDP: an RL model with multiple states)

*MAB vs MDP: single vs multiple states (state transition)*
To maximize throughput over noisy wireless channel:

- **Arms** = transmission rate
- **Reward** = bytes of successful transmission
- c.f., interference with other AP’s may need to be considered (multi-agent RL?)
Recommender System

To maximize number of hits in online ad placement:

- **Arms** = ads/contents
- **Reward** = user click
- c.f., personalization may improve hit-rate (contextual MAB?)
Regret Minimization

- *K*-armed bandit problem is to \textbf{maximize cumulated reward}

- Equivalently, we want to minimize the \textbf{expected regret} after \( n \) plays, which is defined as

\[
\text{Regret}(n) := \mu^* n - \sum_{i=1}^{K} \mu_i \mathbb{E}[T_i(n)] = \sum_{i=1}^{K} (\mu^* - \mu_i) \mathbb{E}[T_i(n)]
\]

where \( \mu^* := \max_{i \in [K]} \mu_i \) is the best expected reward, and \( T_i(n) \) is the number of times playing arm \( i \in [K] \).

*Regret minimization to study the exploration-exploitation trade-off*
Simple Heuristics

For regret minimization,

- **Greedy strategy**: select the arm with the highest average after sampling $L$ times for each arm at the beginning
  - A little knowledge is dangerous
  - We may need some **continuous exploration**

- **$\epsilon$-greedy**: select an arm at random with probability $\epsilon \in [0, 1]$ and otherwise do a greedy selection
  - The value of $\epsilon$ controls the trade-off between exploration and exploitation
Regret Analysis of $\epsilon$-greedy

When $\epsilon$ is constant,

- After sufficient time $t \gg 1$, it eventually identifies best arm $i^*$
- However, it may keep exploring other arms with constant portion of time, i.e., linear regret $\simeq \sum_{t=1}^{n} \epsilon = O(n)$

When $\epsilon_t \propto 1/t$,

- After sufficient time $t \gg 1$, it also identifies best arm $i^*$ eventually, i.e., logarithmic regret

$$\text{Regret} \simeq \sum_{t=1}^{n} \epsilon_t = \sum_{t=1}^{n} \frac{1}{t} = O(\log n)$$

Inefficient exploration! All the arms are explored equally! An efficient one may explore little clearly bad arm $i$ s.t. $\mu_i \ll \mu^*$. 


Upper Confidence Bound (UCB)

UCB selects arm with the highest potential which is quantified by UCB:

- **Initialization**: play each arm once.

- **Loop**: play arm $i$ that maximizes $\text{UCB} \bar{x}_i + \sqrt{\frac{2 \log n}{T_i(n)}}$, where
  - $n$: the overall number of plays done so far
  - $\bar{x}_i$: the empirical mean of arm $i$
  - $T_i(n)$: the number of times arm $i$ has been played so far

Observation: as $n$ increases, the term $\sqrt{\frac{2 \log n}{T_i(n)}}$ increases. 
⇒ This ensures that all arms are tried infinitely often

*UCB is higher for less explored arms*
Outline

Exploration vs. exploitation

▶ Multi-Armed Bandit (MAB) problem
▶ Formulation of regret minimization
▶ UCB (Upper Confidence Bound) algorithm

Advances in efficient exploration

▶ Typical exploration in RL *in MAB/MDP
▶ Structured exploration: meta RL, hierarchical RL
▶ Generative model: world model
Convergence of Q-learning

Q-learning evaluates and improves simultaneously current policy $\pi$: when $(S_n, A_n, R_n, S'_n) = (s, a, r, s')$,

$$Q_{n+1}(s, a) \leftarrow (1 - \alpha_n(s, a))Q_n(s, a) + \alpha_n(s, a)\left((r + \gamma \max_{a'} Q_n(s', a')) - Q_n(s, a)\right)$$

Theorem ([Wat:QL])

For MDP with bounded reward, i.e., $\|R_n\| < \infty$, $Q_n \to Q^*$ w.p. 1 as $n \to \infty$ if

*with probability 1

$$\sum_{n=0}^{\infty} \alpha_n(s, a) = \infty \quad \text{and} \quad \sum_{n=0}^{\infty} (\alpha_n(s, a))^2 < \infty \quad \forall (s, a) \in S \times A,$$

where the first part ensures that every pair of state and action is visited infinitely often

Typical Exploration in RL

\textbf{\(\varepsilon\)-greedy exploration}

- Select action maximizing \(Q(s, a)\) at state \(s\) w.p. \(1 - \varepsilon\)
- Select action \textit{uniformly} at random w.p. \(\varepsilon\)

\textbf{Softmax exploration}

- Select random action \(a\) from \textit{Boltzmann distribution}:

\[
\text{softmax}(a; s) = \frac{\exp(\beta Q(s, a))}{\sum_{a'} \exp(\beta Q(s, a'))}
\]

where \(\beta\) controls the degree of noise

If an MDP is communicating, i.e., for any \(s, s' \in S\), there always exists a sequence of action connecting them with positive probability, both exploration schemes guarantee \textbf{infinitely often} visits at each state and action

*Q-learning converge
Soft Actor Critic [Haa:SAC]

**Algorithm 1 Soft Actor-Critic**

- Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.

  **for** each iteration do
  **for** each environment step do
    - $a_t \sim \pi_\phi(a_t|s_t)$ *forward*
    - $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    - $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  **end for**
  **for** each gradient step do
    - $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    - $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    - $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ 
    - $\bar{\psi} \leftarrow \tau \bar{\psi} + (1 - \tau)\psi$ *backward*
  **end for**
  **end for**

$J_\pi(\phi) = \mathbb{E}_{s_t \sim D} \left[ D_{KL} \left( \pi_\phi(\cdot|s_t) \mid \exp \left( \frac{Q_\theta(s_t, \cdot)}{Z_\theta(s_t)} \right) \right) \right].$

*actor-critic architecture with separate policy (actor) and value function (critics) networks [Q-learning, DQN]*

*Jv: soft value network, JQ: two soft Q-function network (for speed-up); target value network parameter (with bar)*

*policy improvement step; minimize J for policy network (for soft actor) [with softmax Q-function]*

Soft Actor Critic [Haa:SAC]

*Soft actor-critic (yellow) performs consistently across all continuous control benchmarks*

We cannot train machine for every single task, whereas there are some structure underlying a set of tasks

- e.g., for the set of tasks, the state dynamics are the same, but each task is described by different parameters $g = (3, 2, 1), (1, 3, 2), ...$
Meta Reinforcement Learning (2)

We cannot train machine for every single task, whereas there are some structure underlying a set of tasks

- e.g., a policy to accomplish task can be decomposed into several sub-policies in common

Frans, Kevin, et al. ”Meta learning shared hierarchies.” ICRL 2018