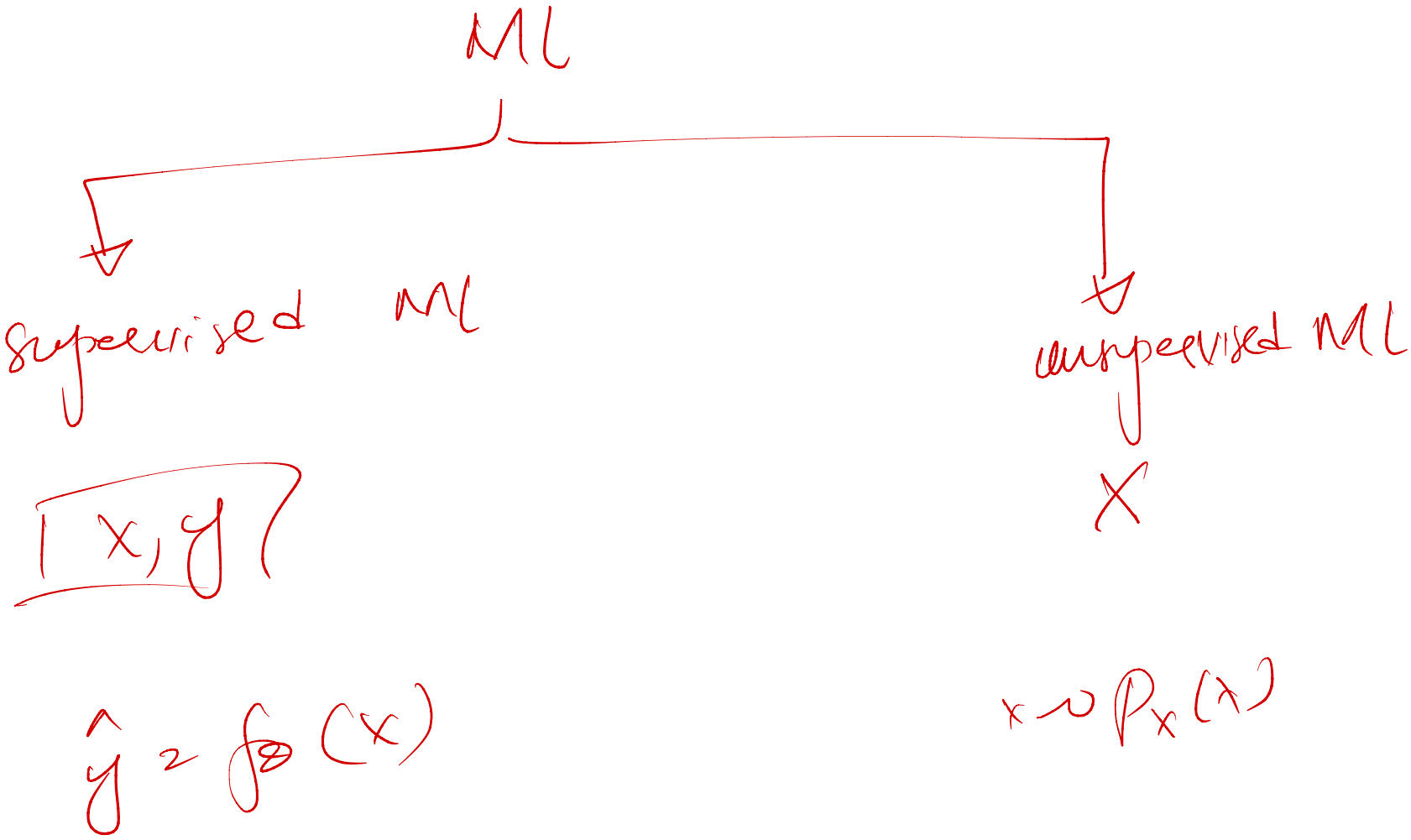
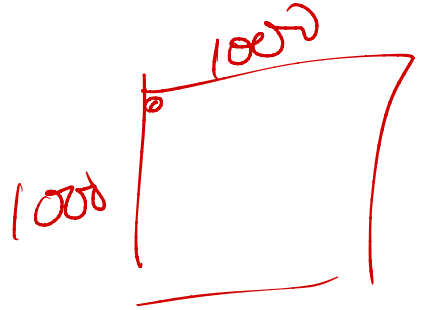
Reinforcement Learning

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

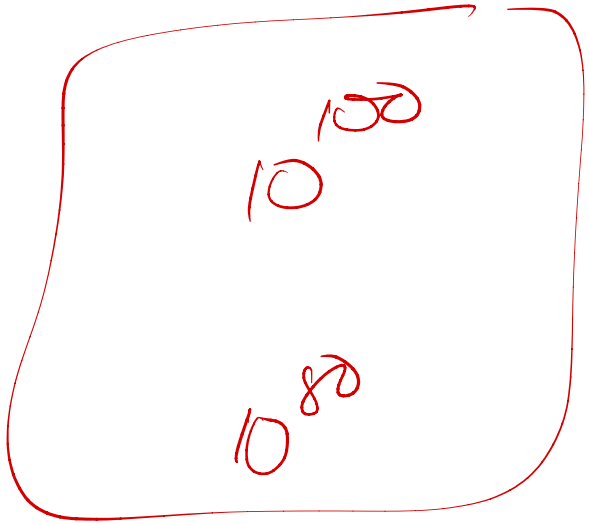

ML
Supervised ML
Unsupervised ML
(x, y)
X
$\hat{y} = f_\theta(x)$
$x \sim p_X(x)$

playing chess

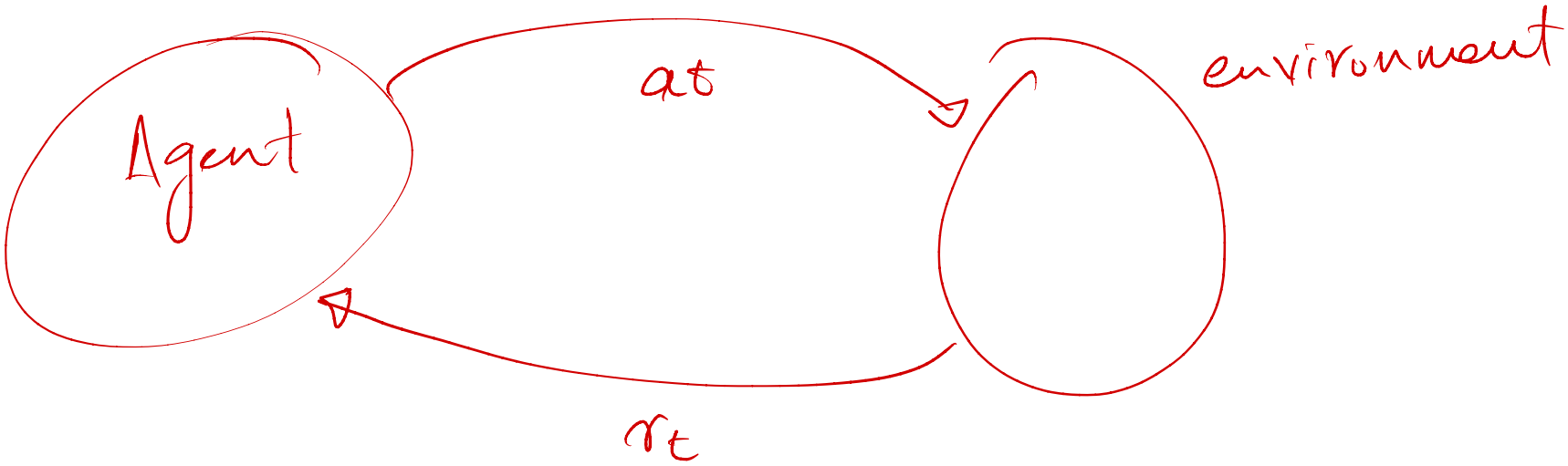


10^{-6}

(256)



S	a	r
S_1	a_1	r_1
S_2	a_2	r_2
$\vdots$	$\vdots$	$\vdots$
S_N	a_N	r_N



state s_t

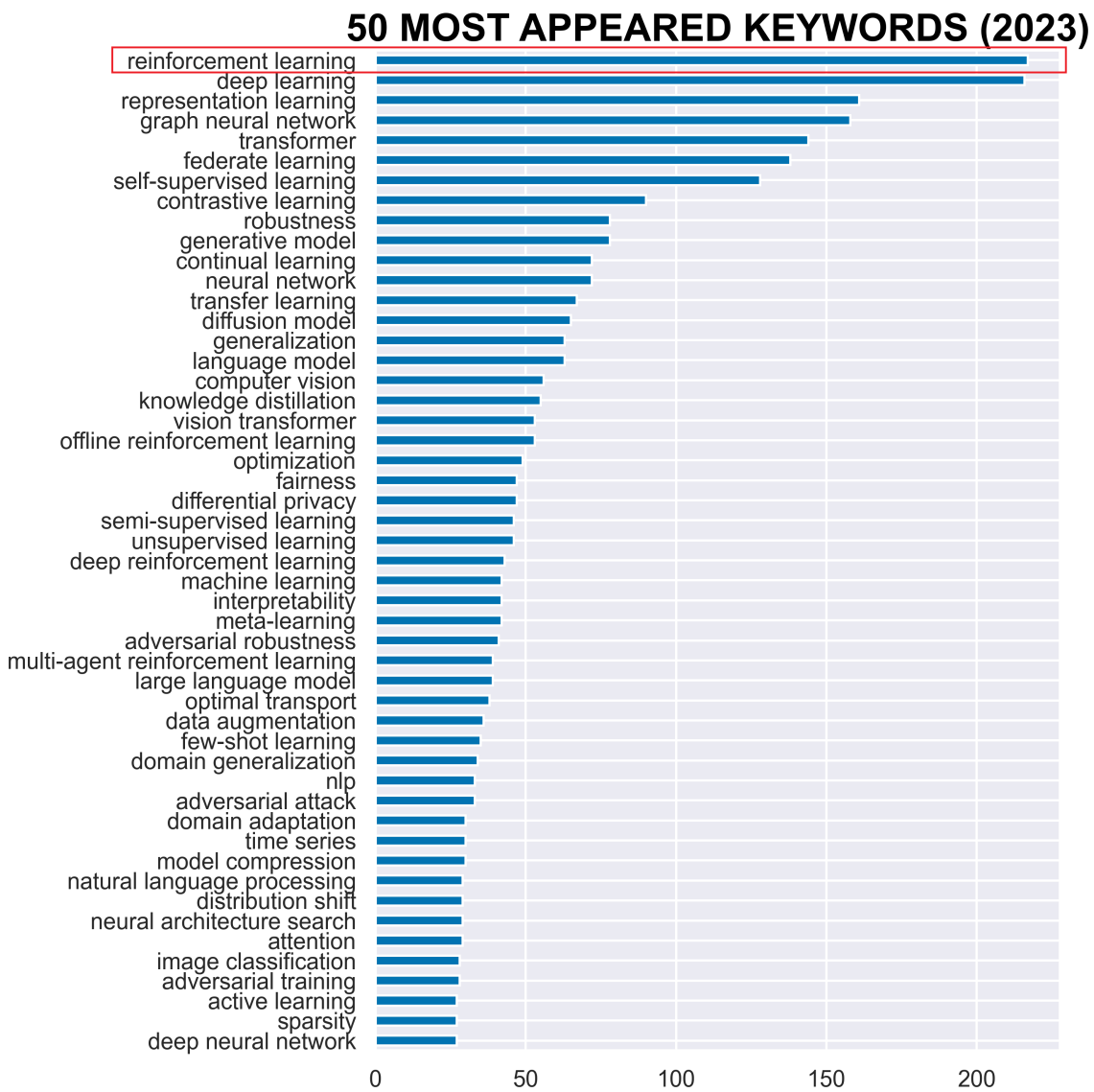
action a_t

transition dynamics $p(s_{t+1} \mid s_t, a_t)$

reward $p(r \mid s_t, a_t)$

The probability problems involved are formidable. . . [but] the theory of sequential design will be of the greatest importance to mathematical statistics and to science... – Robins 1952

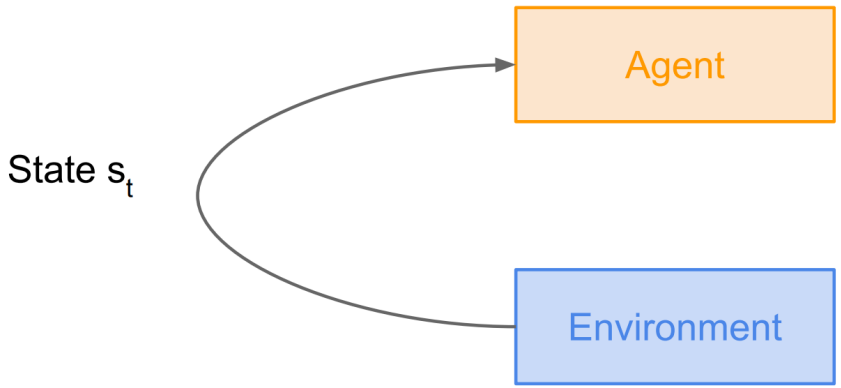
Reinforcement Learning - ICLR 2023



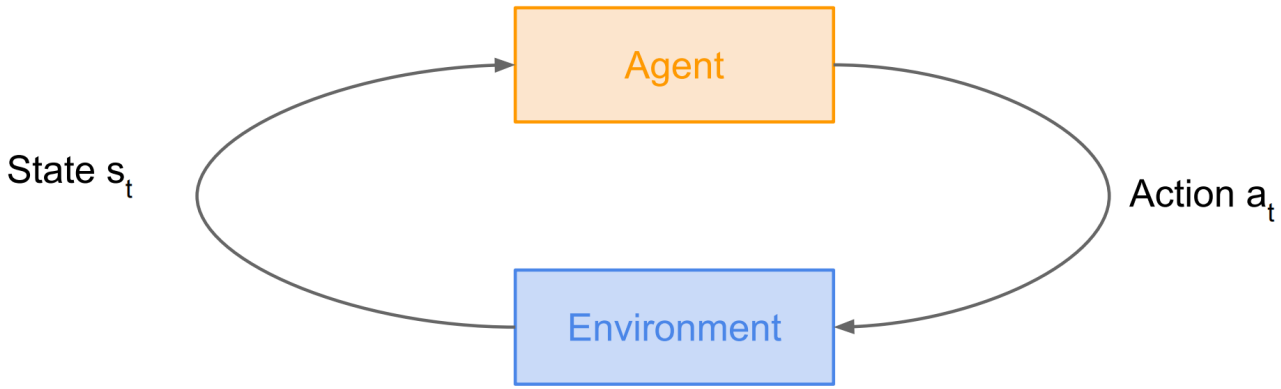
[0]https://github.com/EdisonLeeeee/ICLR2023-OpenReviewData

What is Reinforcement Learning?

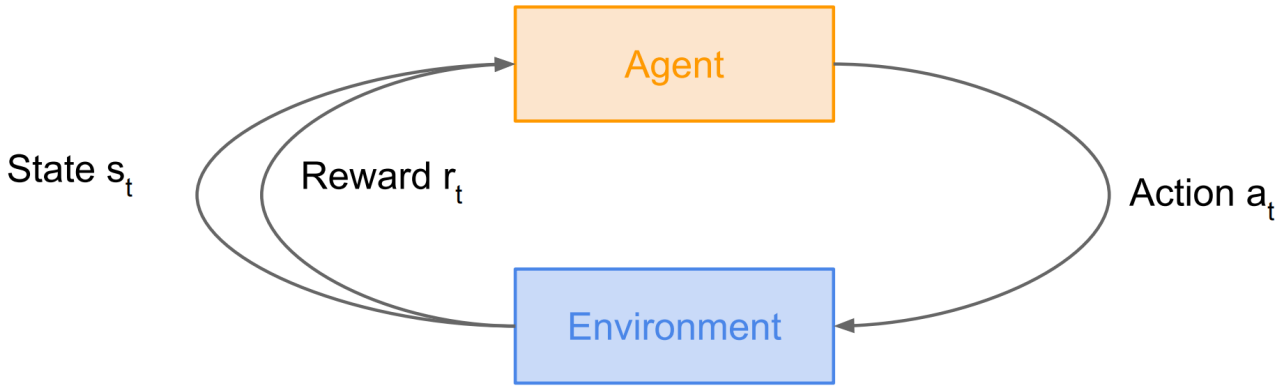
What is Reinforcement Learning? (cont.)



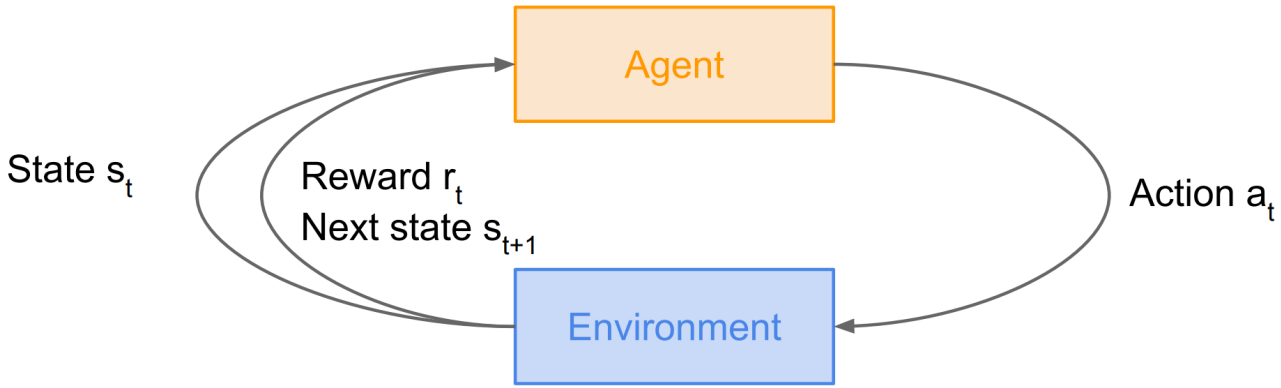
What is Reinforcement Learning? (cont.)

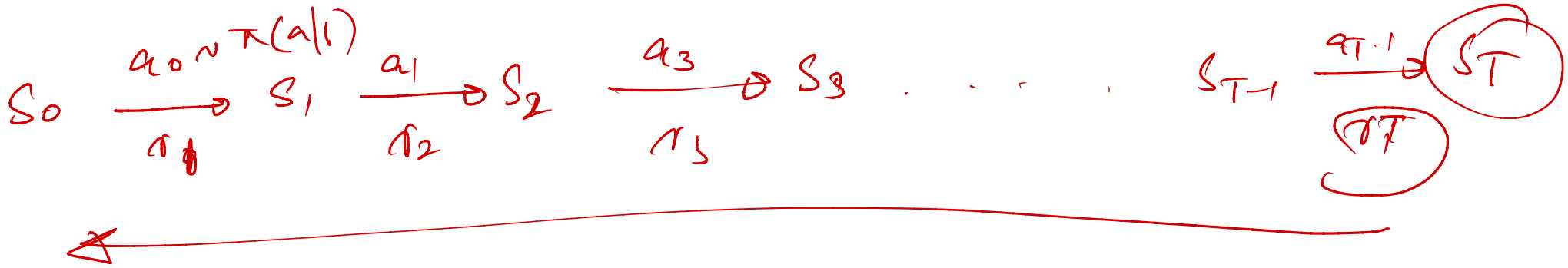


What is Reinforcement Learning? (cont.)



What is Reinforcement Learning? (cont.)



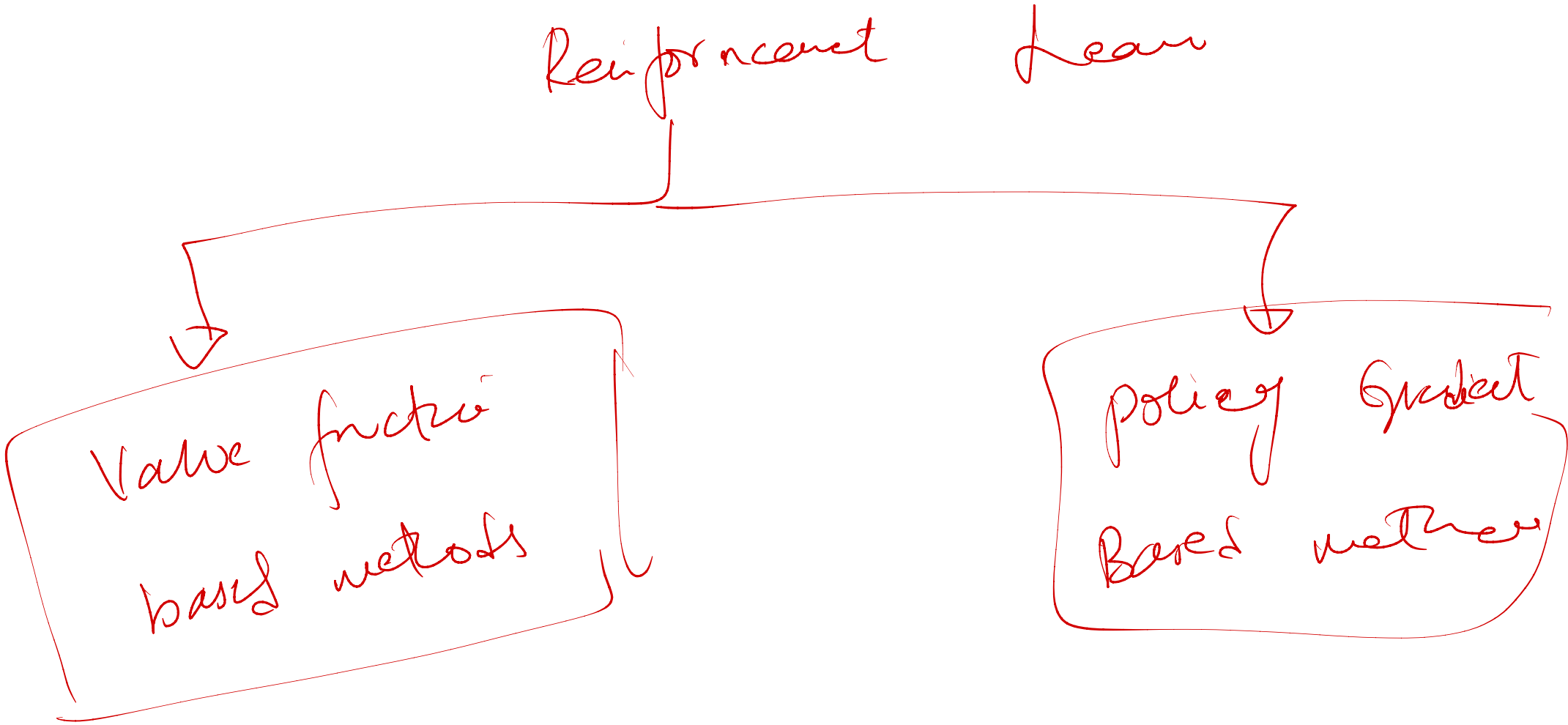


trajectory

$$\zeta = \sum_{t=0}^{T} \gamma^t (r_t)$$

maximize ζ

Expected return


Reinforcement Learn
Value functi
based methods
policy Gradient
Based methods

Supervised Learning vs. Reinforcement Learning

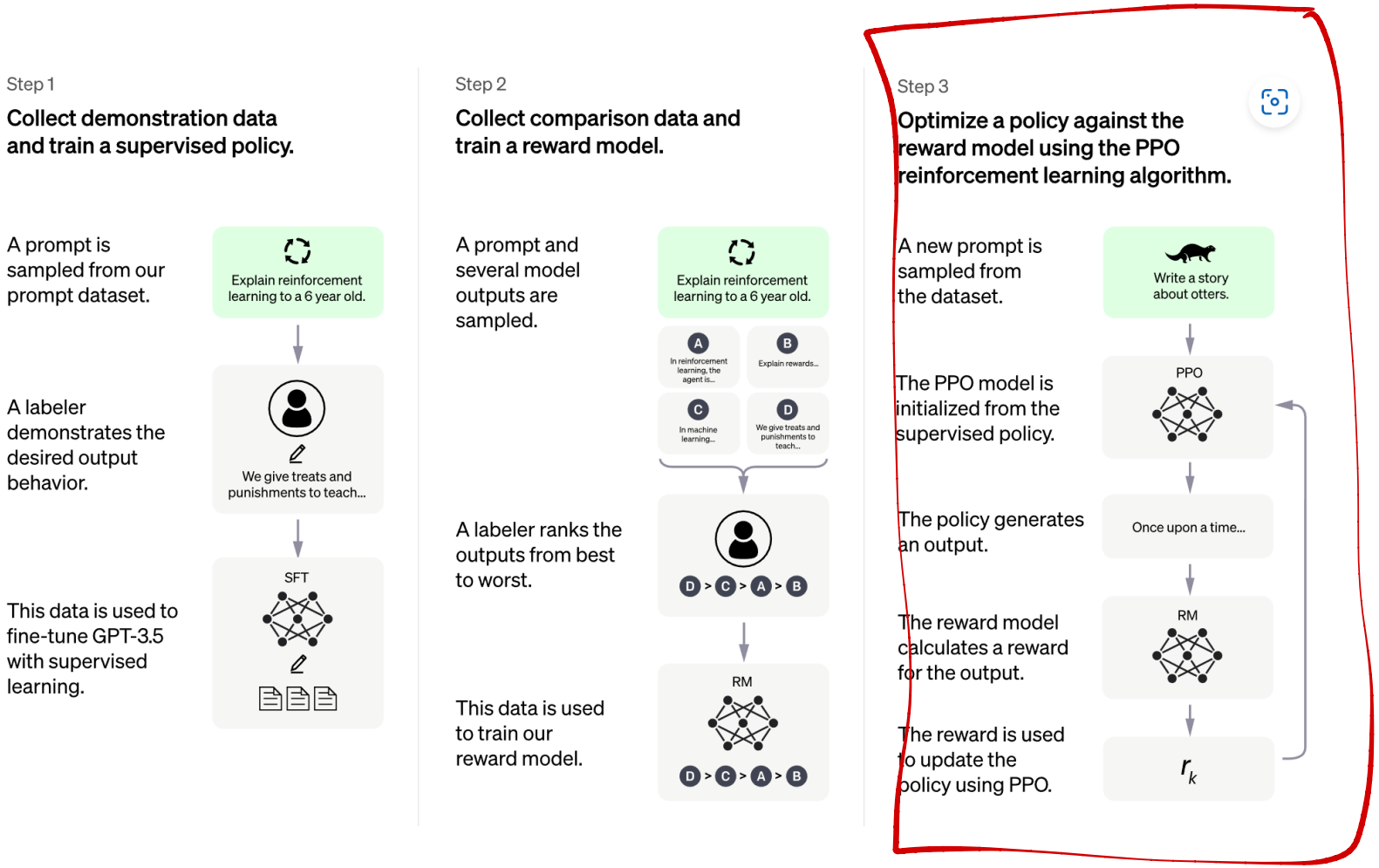
Supervised Learning

- Given labeled data: $\{(x_i, y_i)\}$ learn $f(x) \approx y$
- Directly told what to output
- Inputs x are independently, identically distributed (i.i.d.)

Reinforcement Learning

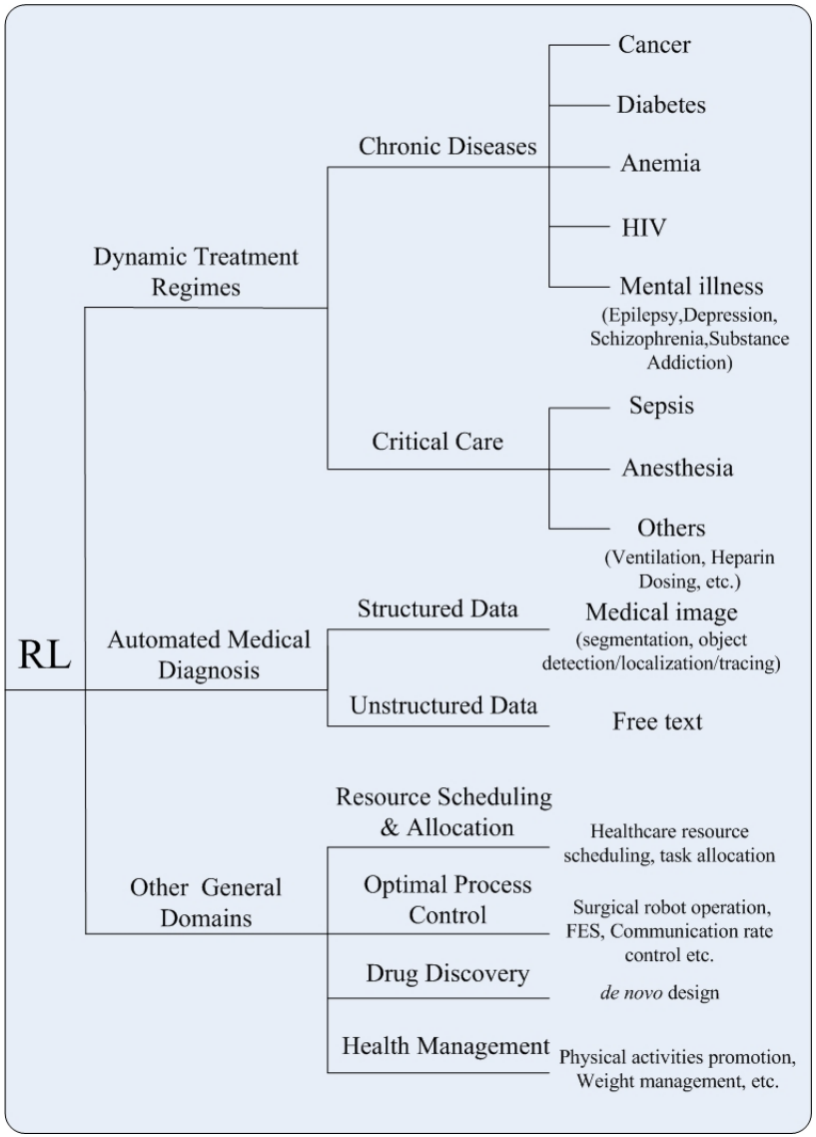
- Learn behavior $\pi(a|s)$
- from experience, indirect feedback
- data not i.i.d.: actions affect the future observations

Some Examples - ChatGPT



[0]https://openai.com/blog/chatgpt

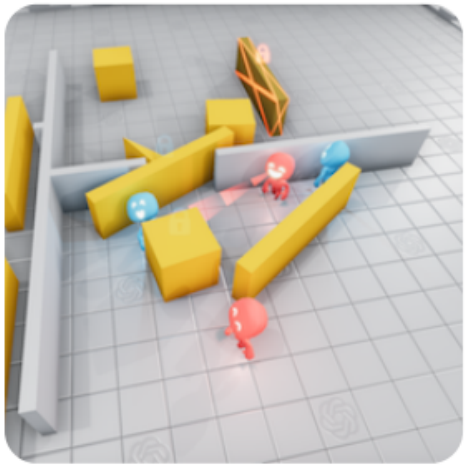
Some Examples - Healthcare



[0] Yu, et al. Reinforcement Learning in Healthcare: A Survey

Some Examples - Multiagent Hide & Seek

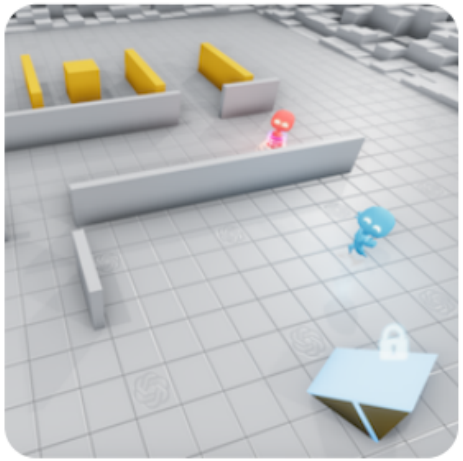
(a) Running and Chasing



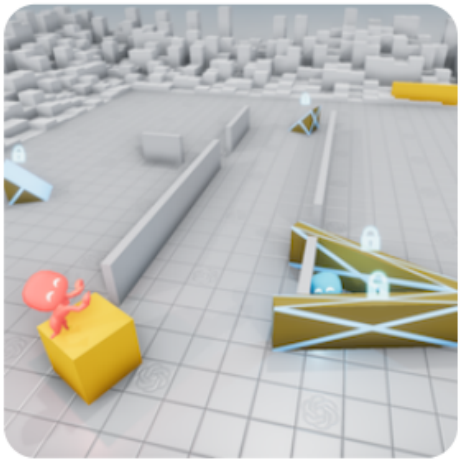
(b) Fort Building

(c) Ramp Use

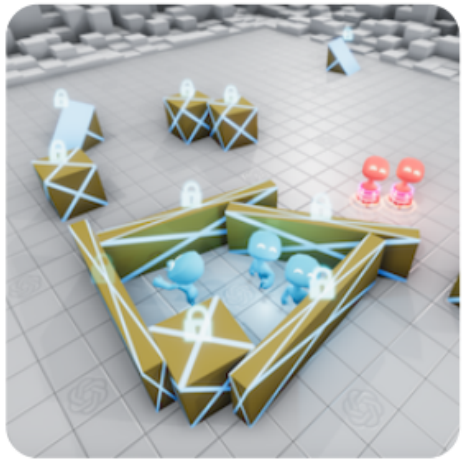
Some Examples - Multiagent Hide & Seek (cont.)



(d) Ramp Defense



(e) Box Surfing



(f) Surf Defense

Some Examples - Multiagent Hide & Seek (cont.)

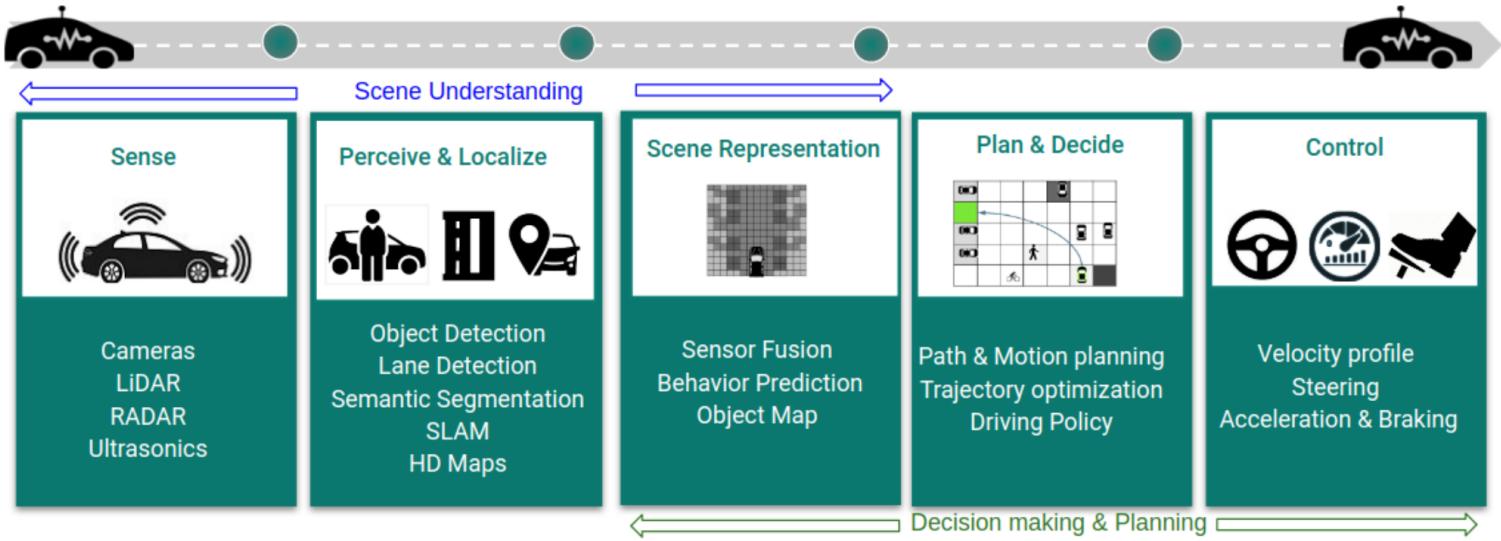
https://www.youtube.com/watch?v=kopoLzvh5jY

[0] Baker, et al. Emergent Tool Use From Multi-Agent Autocurricula

Some Examples - Grand Turismo GTSophie

[0] https://www.gran-turismo.com/us/gran-turismo-sophy/technology/

Some Examples - Autonomous Driving



[0] Kiran, et al. Deep Reinforcement Learning for Autonomous Driving: A Survey

Some Examples - AlphaGo

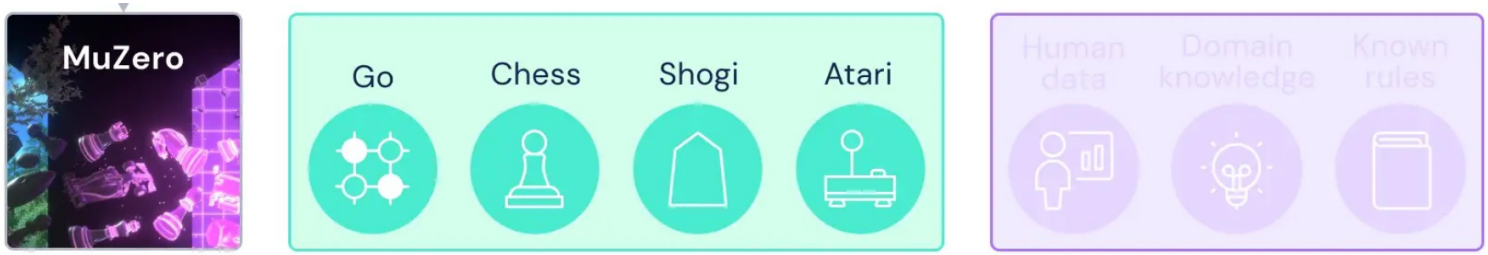
'AlphaGo' AI Scores Narrow Win Against Ke Jie, World's Top 'Go' Player

By Lucian Armasu published May 23, 2017

Comments (4)

[0]https://www.tomshardware.com/news/alphago-narrow-win-ke-jie,34486.html

Some Examples - Muzero



MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics.
(Dec 2020, Nature)

[0]https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules

Some Examples - Minecraft

AI learns how to play Minecraft by watching videos

Open AI has trained a neural network to play Minecraft by Video PreTraining (VPT) on a massive unlabeled video dataset of human Minecraft play, while using just a small amount of labeled contractor data.

With a bit of fine-tuning, the AI research and deployment company is confident that its model can learn to craft diamond tools, a task that usually takes proficient humans over 20 minutes (24,000 actions). Its model uses the native human interface of keypresses and mouse...

29 June 2022 | Artificial Intelligence

[0]https://www.artificialintelligence-news.com/2022/06/29/ai-learns-how-to-play-minecraft-by-watching-videos/

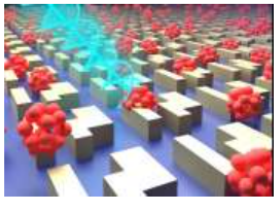
Some Examples

Reinforcement learning: From board games to protein design

Scientists have successfully applied reinforcement learning to a challenge in molecular biology. The team of researchers developed powerful new protein design software adapted from a strategy proven adept at board games like ...

BIOTECHNOLOGY

APR 20, 2023 0 37



Chiral detection of biomolecules based on reinforcement learning

As one of the basic physical properties, chirality plays an important role in many fields. Especially in biomedical chemistry, the discrimination of enantiomers is a very important research subject. Most biomolecules exhibit ...

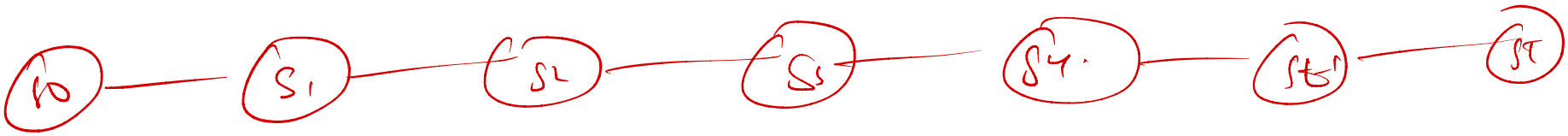
OPTICS & PHOTONICS

FEB 22, 2023 0 4

[0]https://phys.org/tags/reinforcement+learning/

Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world i.e., future is independent of the past given the present
- Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$
  - $\mathcal{S}$: 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



$$p(s_t / s_{t-1}) = p(s_t / s_{t-1}, s_{t-2} \ldots s_0)$$

↓ Markov process

Markov process

$$p(s_{t+1} / s_t, a_t) = p(s_{t+1} / s_t, a_t, s_{t-1}, a_{t-1} \ldots s_0 a_0)$$

Markov Decision Process (cont.)

- At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(.|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(.|s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}
- A policy π is a function from S to A that specifies what action to take in each state
- **Objective**: Find policy $\pi^\star$ that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

Markov Decision Process (cont.)

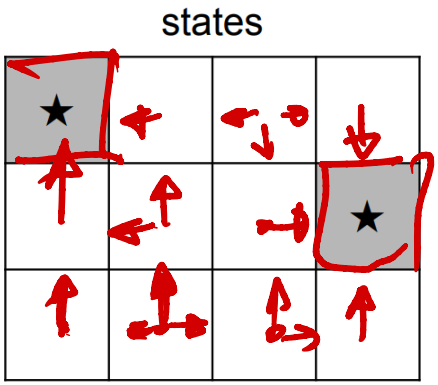
- The Agent and the Environment interact in a time-sequenced loop
- Agent responds to [State, Reward] by taking an Action
- Environment responds by producing next step's State
- Environment also produces a scalar denoted as Reward
- Each State is assumed to have the Markov Property, meaning:
  - Next State/Reward depends only on Current State (for a given Action)
  - Current State captures all relevant information from History
  - Current State is a sufficient statistic of the future (for a given Action)
- Goal of Agent is to maximize Expected Sum of all future Rewards
- By controlling the (Policy : State $\rightarrow$ Action) function
- This is a dynamic (time-sequenced control) system under uncertainty

A simple MDP: Grid World

actions = {

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

}

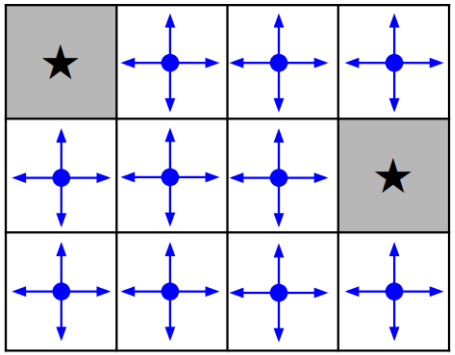


Set a negative "reward" for each transition (e.g. $r = -1$)

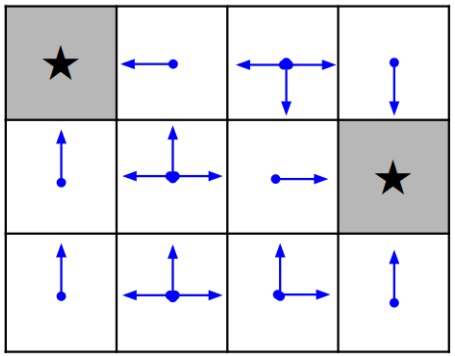
Objective: reach one of terminal states (greyed out) in least number of actions

$$p(\omega_t \mid \omega_1 \ldots \omega_{t-1})$$

A simple MDP: Grid World (cont.)



Random Policy



Optimal Policy

Real World Problems that fit MDP Framework

- Self-driving vehicle (speed/steering to optimize safety/time)
- Game of Chess (Boolean Reward at end of game)
- Complex Logistical Operations (eg: movements in a Warehouse)
- Make a humanoid robot walk/run on difficult terrains
- Manage an investment portfolio
- Control a power station
- Optimal decisions during a football game
- Strategy to win an election (high-complexity MDP)

Why are these problems hard?

- State space can be large or complex (involving many variables)
- Sometimes, Action space is also large or complex
- No direct feedback on "correct" Actions (only feedback is Reward)
- Time-sequenced complexity (Actions influence future States/Actions)
- Actions can have delayed consequences (late Rewards)
- Agent often doesn't know the Model of the Environment
- "Model" refers to probabilities of state-transitions and rewards
- So, Agent has to learn the Model AND solve for the Optimal Policy
- Agent Actions need to tradeoff between "explore" and "exploit"

Policy

- Policy π determines how the agent chooses actions
- $\pi : S \rightarrow A$, mapping from states to actions
- Deterministic Policy:

$$\pi(s) = a$$

- Stochastic Policy:

$$\pi(a|s) = Pr(a_t = a|s_t = s)$$

The Optimal Policy

- We want to find optimal policy $\pi^\star$ that maximizes the sum of reward
- But, how do we handle the randomness (initial state, transition probability...)?

The Optimal Policy

- We want to find optimal policy $\pi^\star$ that maximizes the sum of reward
- But, how do we handle the randomness (initial state, transition probability...)?
- **Solution**: Maximize the expected sum of rewards!
- Formally,

$$\pi^\star = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right] \text{ with } s_0 \sim p(s_0), a_t \sim \pi(.|s_t), s_{t+1} \sim p(.|s_t, a_t)$$

Value Function

- How good is a state?

$$V^{\pi}(s) = E[G] = E\left[\sum_{t} \beta^{t} r_{t}\right]$$

$a_t \sim \pi(a_t|s_t)$

$s_0 = s$

Bellman Equations

$$\hat{r} = \mathbb{E}[r] \qquad r \sim p(r_i | s_k, a)$$

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{s'}\Big[r(s, a \sim \pi(a|s)) + \beta V(s')\Big]$$

V functions

$$\boxed{V^{\pi}(s) = \hat{r}(s, a \sim \pi(a|s)) + \beta \mathbb{E}_{s' \sim P(s'|s,a)}\left(V^{\pi}(s')\right)}$$

θ function

$$V^{\pi}(s) = E\left[\sum_{t \geq 0} \beta^{t} r_t \;\middle|\; s_0 = s,\ a_t \sim \pi(a_t | s_t)\right]$$

$$Q^{\pi}(s,a) = r(s,a) + \beta \, \mathbb{E}_{s' \sim p(s'|s,a)} \left[V^{\pi}(s') \right]$$

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \, Q(s,a)$$

$$\theta^{\pi}(s,a) = \hat{r}(s,a) + \gamma\left(\beta \sum_{s' \sim p(s'|s,a)} \pi(a'|s') \cdot \theta'(s', a')\right)$$

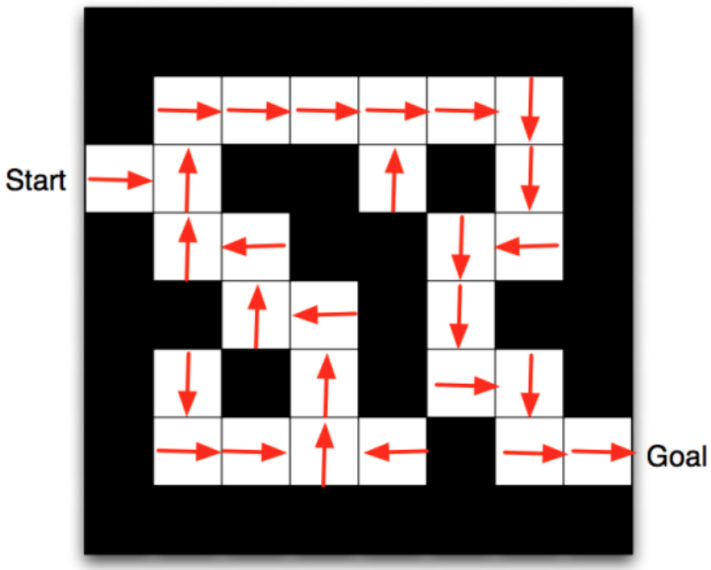
Value Function

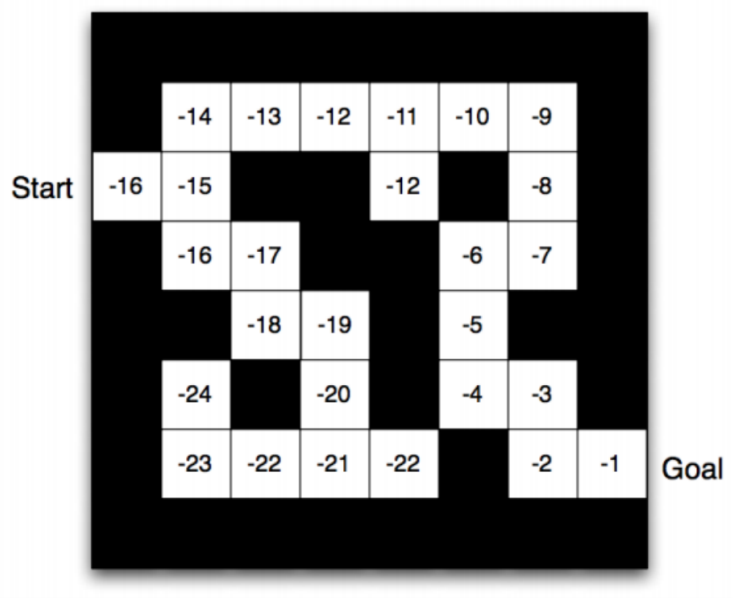
- How good is a state?
- The **value function** at state s, is the expected cumulative reward from following the policy from state s:

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

- Computing the value function is generally impractical, but we can try to approximate (learn) it
- The benefit is credit assignment: see directly how an action affects future returns rather than wait for rollouts

Value Function





Q-Value Function

- Can we use a value function to choose actions?

$$\arg\max_a r(s_t, a) + \gamma \mathbb{E}_{p(s_{t+1}|s_t,a_t)} \left[v^\pi(s_{t+1}) \right]$$

Q-Value Function

- Can we use a value function to choose actions?

$$\arg\max_a r(s_t, a) + \gamma \mathbb{E}_{p(s_{t+1}|s_t,a_t)} \left[v^\pi(s_{t+1}) \right]$$

- **Problem:** this requires taking the expectation with respect to the environment's dynamics, which we don't have direct access to!
- Instead, learn a Q-value function.

Q-Value Function

- 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:

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

- Relationship:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a)$$

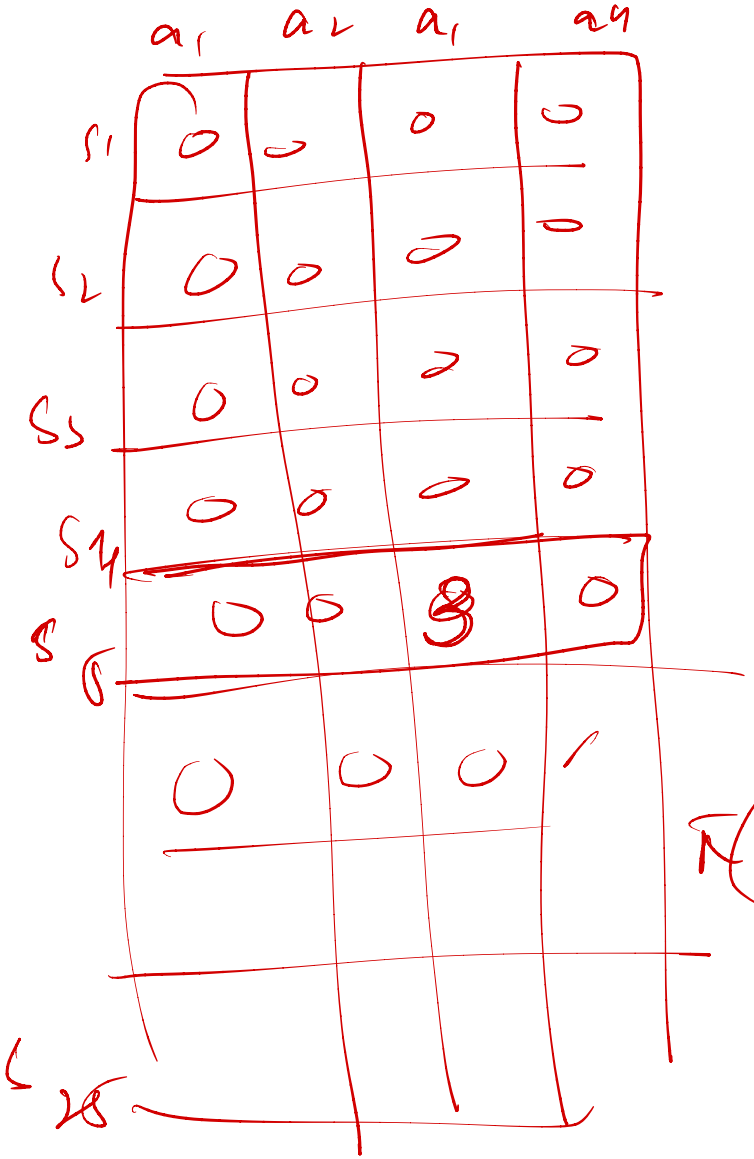
- Optimal Action:

$$\arg\max_{a} Q^{\pi}(s,a)$$

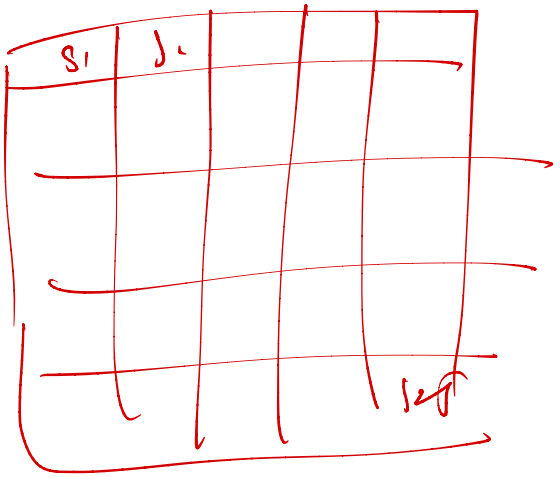
$$Q^{\pi}(s,a) = r(s,a) + \beta \, \mathop{E}_{s' \sim p(s'|s,a)} \left[V^{\pi}(s') \right]$$

$$Q^{\pi}(s,a) = r(s,a) + \beta \, \mathop{E}_{s' \sim p(s'|s,a)} \left[\mathop{E}_{a' \sim \pi(a'|s')} \left[\tilde{Q}^{\pi}(s',a') \right] \right]$$

$$Q^{\pi}(s,a) = r(s,a) + \beta \, \mathop{E}_{s' \sim p(s'|s,a)} \left[\sum_{a' \in A(s')} \pi(a'|s') \, \tilde{Q}^{\pi}(s',a') \right]$$



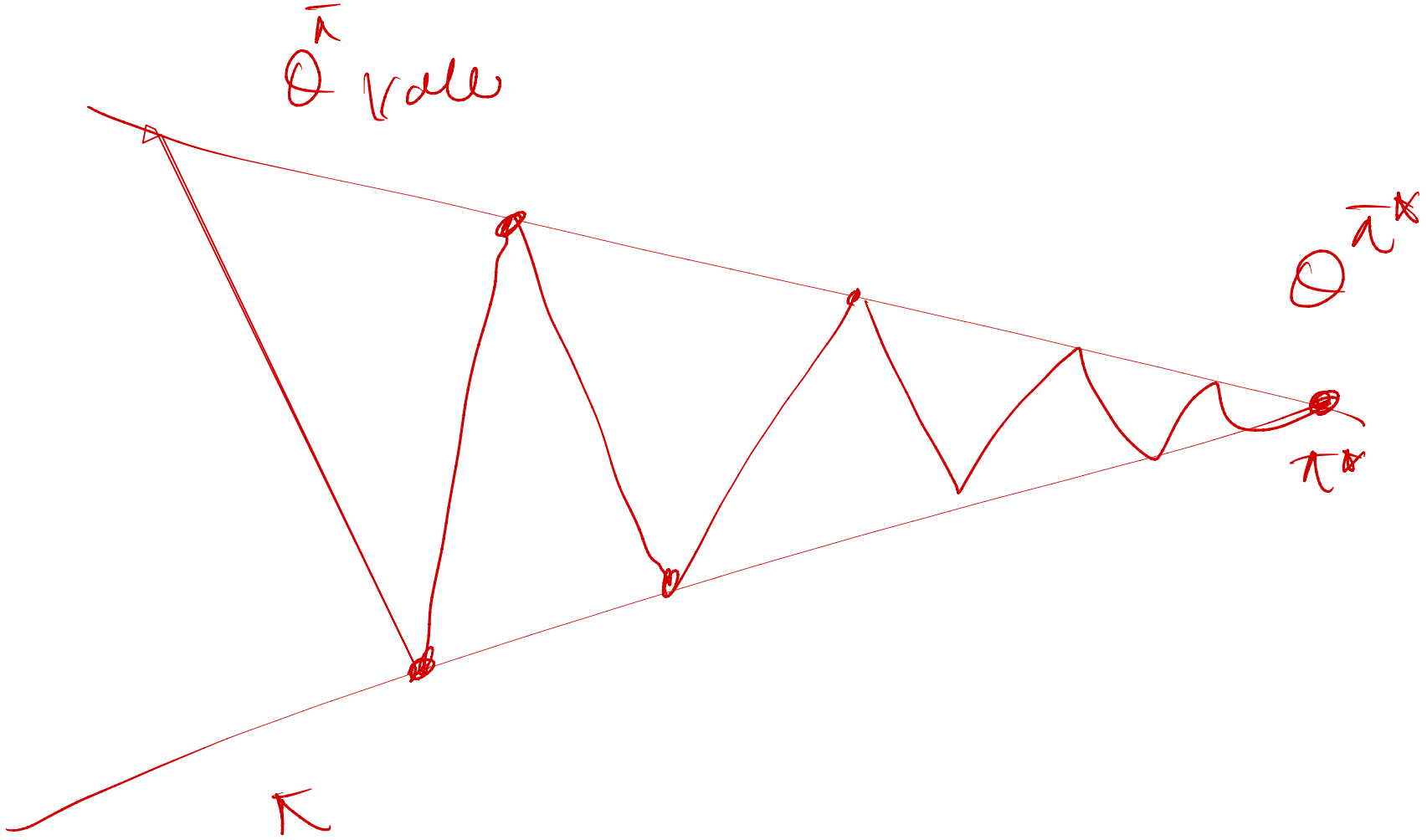
$Q(s, a)$



$s_0 \sim p(s_0)$

$$\pi(a_t | s_t) = \max_a Q(s_t, a)$$

$$\pi(a_t | s_t) = \begin{cases} 1 & a = \arg\max_a Q(s_t, a) \\ 0 & \text{otherwise} \end{cases}$$


$\bar{\theta}$ Valle
θ^{λ^*}
λ^*
λ

Q learning Algorithm

π — uniform Random

Repeat until convergence

calculate $Q(s,a)$

optimize $\pi(a_t|s_t) = \begin{cases} 1-\epsilon \text{ if } a_t = \underset{a}{\text{argmax}}\, Q(s,a) \\ \epsilon / |A(s)| \text{ otherwise} \end{cases}$

$$Q^{\pi}(s,a) = r(s,a) + \underbrace{\beta \; \mathop{E}_{s' \sim p(s'|s,a)} \left[V^{\pi}(s') \right]}$$

$$Q^{\pi}(s,a) = r(s,a) + \beta \mathop{E}_{s' \sim p(s'|s,a)} \left[\sum_{a' \sim \pi(a'|s')} \pi(a'|s') Q^{\pi}(s',a') \right]$$

$$\theta^t(s,a) = \underbrace{\theta^{t-1}(s,a)(1-\alpha)} + \underbrace{\alpha\,\theta^{\pi^*}(s,a)}$$

Bellman Equation

- The Bellman Equation is a recursive formula for the Q-value function:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)\pi(a'|s')}[Q^{\pi}(s',a')]$$

- There are various Bellman equations, and most RL algorithms are based on repeatedly applying one of them.

Optimal Bellman Equation

- The optimal policy $\pi^\star$ is the one that maximizes the expected discounted reward, and the optimal Q-value function $Q^\star$ is the Q-value function for $\pi^\star$
- The Optimal Bellman Equation gives a recursive formula for $Q^\star$:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)} \left[\max_{a'} Q^\pi(s_{t+1}, a') | s_t = s, a_t = a \right]$$

- This system of equations characterizes the optimal Q-value function. So maybe we can approximate $Q^\star$ by trying to solve the optimal Bellman equation!
- **Intuition**: If the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r(s,a) + \gamma Q^\pi(s', a')$

Q-Learning

- Let Q be an action-value function which hopefully approximates $Q^\star$
- Q-learning is an algorithm that repeatedly adjusts Q to minimize the Bellman error
- Each time we sample consecutive states and actions $(s_t, a_t, s_{t+1}$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{\left[r(s, a) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]}_{\text{Bellman Error}}$$

Exploration-Exploitation Tradeoff

- Notice: Q-learning only learns about the states and actions it visits.
- Exploration-exploitation tradeoff: the agent should sometimes pick suboptimal actions in order to visit new states and actions.
- Simple solution: ϵ-greedy policy
  - With probability $1 - \epsilon$, choose the optimal action according to Q
  - With probability ϵ, choose a random action
- ϵ-greedy policy still widely used today regardless of simplicity

Q-Learning Algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

$$Q(S,A) = Q(S,A)(1-\alpha) + \alpha\left[R + \gamma \max_a Q(S',a)\right]$$

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Solution**: use a function approximator to estimate $Q(s, a)$. E.g. a neural network!

$$Q(s, a, ; \theta) \approx Q^{\star}(s, a)$$

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Solution**: use a function approximator to estimate $Q(s, a)$. E.g. a neural network!

$$Q(s, a, ; \theta) \approx Q^{\star}(s, a)$$

- If the function approximator is a deep neural network $\Rightarrow$ Deep Q-Learning (DQN)

Deep Q-Learning (DQN)

- Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^{\star}(s,a) = \mathbb{E}_{p(s'|s,a)}\left[r(s,a) + \gamma \max_{a'} Q^{\star}(s',a')|s,a\right]$$

- Forward Pass:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a\sim p(.)}\left[(y - Q(s,a;\theta))^2\right]$$

$$\text{where} \quad y = \mathbb{E}_{p(s'|s,a)}\left[r(s,a) + \gamma \max_{a'} Q(s',a';\theta)|s,a\right]$$

- Backward Pass:
  - Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta}\mathcal{L}(\theta) = \mathbb{E}_{s,a\sim p(.),p(s'|s,a)}\left[(r(s,a) + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta))\nabla_{\theta} Q(s,a;\theta)\right]$$

Training the Q-network: Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated $\Rightarrow$ inefficient learning
  - Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) $\Rightarrow$ can lead to bad feedback loops

Training the Q-network: Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated $\Rightarrow$ inefficient learning
  - Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) $\Rightarrow$ can lead to bad feedback loops
- Address these problems using experience replay
  - Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
  - Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

DQN Algorithm

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

DQN with Target Network

- DQN, as stated above, will have learning problems because the target and the prediction are not independant as they both rely on the same network.
- It's like a dog chasing it's own tail.

DQN with Target Network

- DQN, as stated above, will have learning problems because the target and the prediction are not independant as they both rely on the same network.
- It's like a dog chasing it's own tail.
- **Solution:** Use two separate Q-value estimators, each of which is used to update the other.
- The target values are calculated using a target Q-network. The target Q-network's parameters are updated to the current networks every C time steps.
- Target network prevents the network from spiraling around.

DQN with Target Network Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

[0] Minh et al. https://www.nature.com/articles/nature14236

Double DQN

- Q-learning suffers from a maximization bias.
- This is because the update target is $r + \max_a Q^\star(s, a)$. If Q-value is slightly overestimated then this error gets compounded.

Double DQN

- Q-learning suffers from a maximization bias.
- This is because the update target is $r + \max_a Q^\star(s, a)$. If Q-value is slightly overestimated then this error gets compounded.
- **Solution:** Decompose the max operation in the target into action selection and action evaluation.
- Use the current network to select the max action for the next state and then use the target network to get the target Q-value for that action.
- Using these independent estimators, we can have unbiased Q-value estimates of the actions selected using the opposite estimator.
- We can thus avoid maximization bias by disentangling our updates from biased estimates.

DDQN Algorithm

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_\mathcal{B}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$

$$y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$$

    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

[0] https://leejungi.github.io/posts/Dueling-DQN/

On-Policy vs. Off-Policy Algorithms

- **On Policy**: In this, the learning agent learns the value function according to the current action derived from the policy currently being used.
- **Off Policy**: In this, the learning agent learns the value function according to the action derived from another policy.

SARSA

- SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning technique is an Off-Policy technique and uses the greedy approach to learn the Q-value. SARSA technique, on the other hand, is an On-Policy and uses the action performed by the current policy to learn the Q-value.

- SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning technique is an Off-Policy technique and uses the greedy approach to learn the Q-value. SARSA technique, on the other hand, is an On-Policy and uses the action performed by the current policy to learn the Q-value.

Q-Learning: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s, a) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$

- SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning technique is an Off-Policy technique and uses the greedy approach to learn the Q-value. SARSA technique, on the other hand, is an On-Policy and uses the action performed by the current policy to learn the Q-value.

Q-Learning: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s, a) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$

SARSA: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s, a) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$

- The update equation for SARSA depends on the current state, current action, reward obtained, next state and next action.
- This observation lead to the naming of the learning technique as **SARSA** stands for **State Action Reward State Action** which symbolizes the tuple (s, a, r, s', a').

SARSA

- The update equation for SARSA depends on the current state, current action, reward obtained, next state and next action.
- This observation lead to the naming of the learning technique as **SARSA** stands for **State Action Reward State Action** which symbolizes the tuple (s, a, r, s', a').
- Just like DQN, we also have a Deep SARSA.

SARSA Algorithm

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s, a), α, γ, π to an arbitrary policy (non-greedy)

Output: Optimal action value Q(s, a) for each stste-action pair

while *True* do

  for ($i = 0$; $i \leq$ *# of episodes*; $i++$) do

    Initialize s

    Choose a from s, using policy derived from Q

    Repeat(for each step of episodes):

    Take action a; observe reward, r, and next state, s'

    Choose action a' from state s' using policy derived from Q

    $Q(s, a) \longleftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

    $s \longleftarrow s'$; $a \longleftarrow a'$;

    until s is terminal

  end

end

[0]Lockery & Peters, Adaptive learning by a target-tracking system

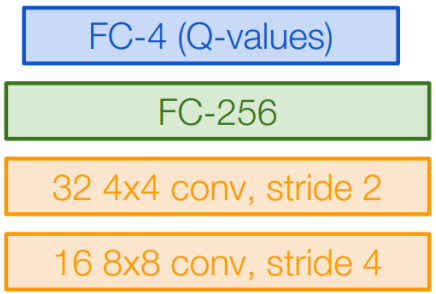
Case Study: Playing Atari Games

- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

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

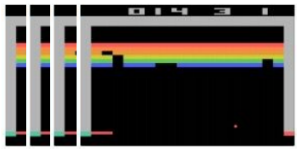
Case Study: Playing Atari Games

$Q(s, a; \theta)$: neural network with weights θ



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

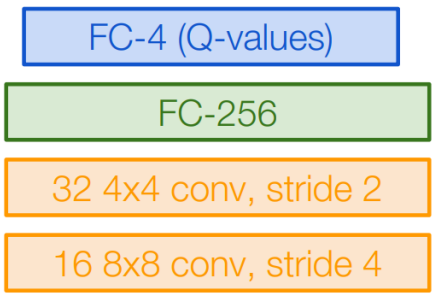


Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

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

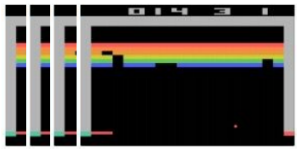
Case Study: Playing Atari Games

$Q(s, a; \theta)$:
neural network
with weights θ



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

https://www.youtube.com/watch?v=V1eYniJ0Rnk

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

Summary

- Markov Decision Process is the mathematical formulation of the Reinforcement Learning Problem defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$
- Each state satisfies the Markov Property i.e., future is independent of the past given the present.
- The Agent and the Environment act in a time sequenced loop. Policy π determines how the agent chooses actions
- Value function approximates how good a state is while Q-value function approximates the goodness of a state action pair.
- Bellman Equation is a recursive formula for the Q-value function
- Q-learning is an algorithm that repeatedly adjusts Q to minimize the Bellman error
- If the Q-value function approximater is Deep Neural Network, we get Deep Q-Learning
- SARSA is an on-policy variation of Q-learning

References

These slides have been adapted from

- Emma Brunskill, Stanford CS234: Reinforcement Learning
- Fei-Fei Li, Yunzhu Li& Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision
- Ashwin Rao, Stanford CME241: Foundations of Reinforcement Learning with Applications in Finance
- Jimmy Ba & Bo Wang, UofT CSC413/2516: Neural Networks and Deep Learning