

# Reinforcement Learning

## CS6960 MultiModal LLM Agents

---

Kenneth Marino

# Quick Announcements

- Add/Drop Course
  - Class is pretty much at capacity
  - If you already know you will drop, please do that ASAP so people can get in off the waiting list
- HW0 due next week
  - It's really short and easy
  - Basically just asking you to read the intro unit to the HF Agents tutorial and write some boilerplate code

# Recommended Readings

- Reinforcement Learning an Introduction - Sutton & Barto
  - It's all good but for introduction to RL Ch1&3 are helpful
  - <http://incompleteideas.net/book/RLbook2020.pdf>
- Reinforcement Learning Courses
  - Stanford Deep RL Class: <https://cs224r.stanford.edu/>
  - Berkeley Deep RL Course: <https://rail.eecs.berkeley.edu/deeprlcourse/>
  - If you need to understand particular topic/algorithm, notes from these courses are really useful

# Warning

- I can't possibly cover everything about RL in a single lecture
- Goal: give the high-level overview of RL and how it can be used in agents
- If you took e.g. Daniel Brown's course, this will be mostly review
- You probably don't need to know everything about RL to understand VLM agents
- For papers/project, if you need to know more details
  - E.g. details of algorithms like PPO
  - See recommended readings/resources

**Any Questions**

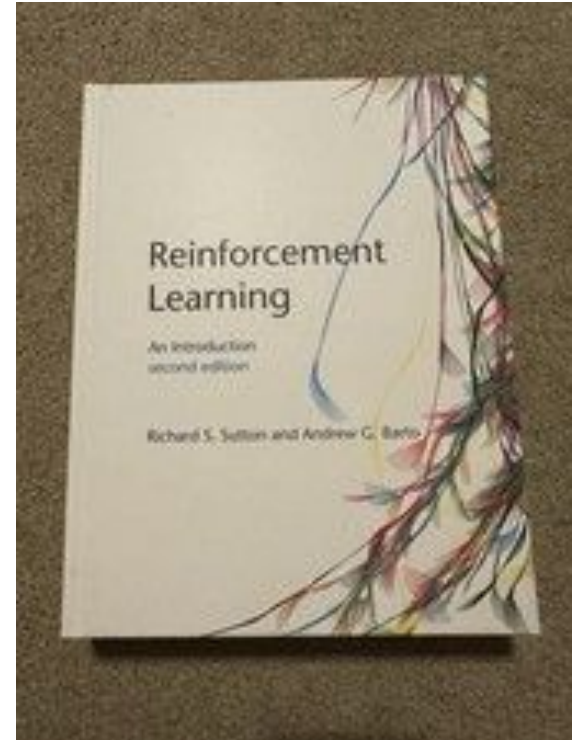
# Recall: What are Agents?

- VLM Agents

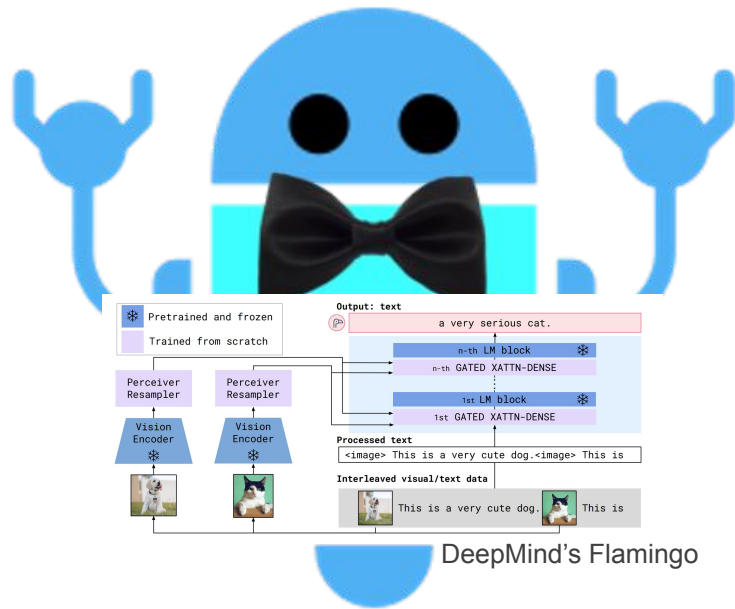
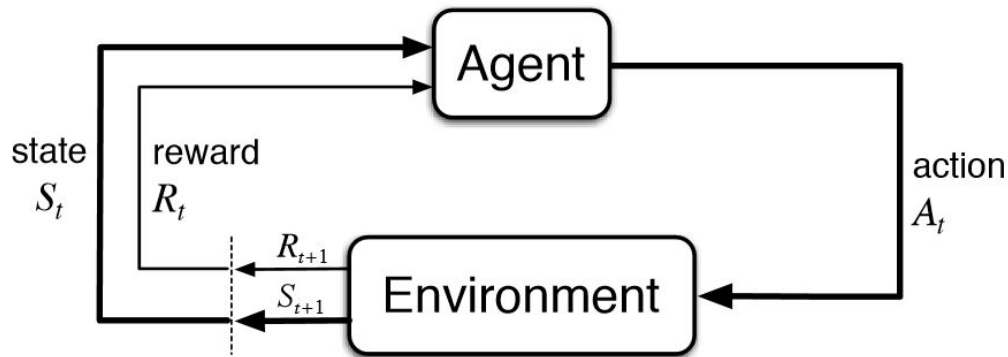
What Even is an Agent?

# Recall: Classical Definition

- Sutton and Barto: "The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment."

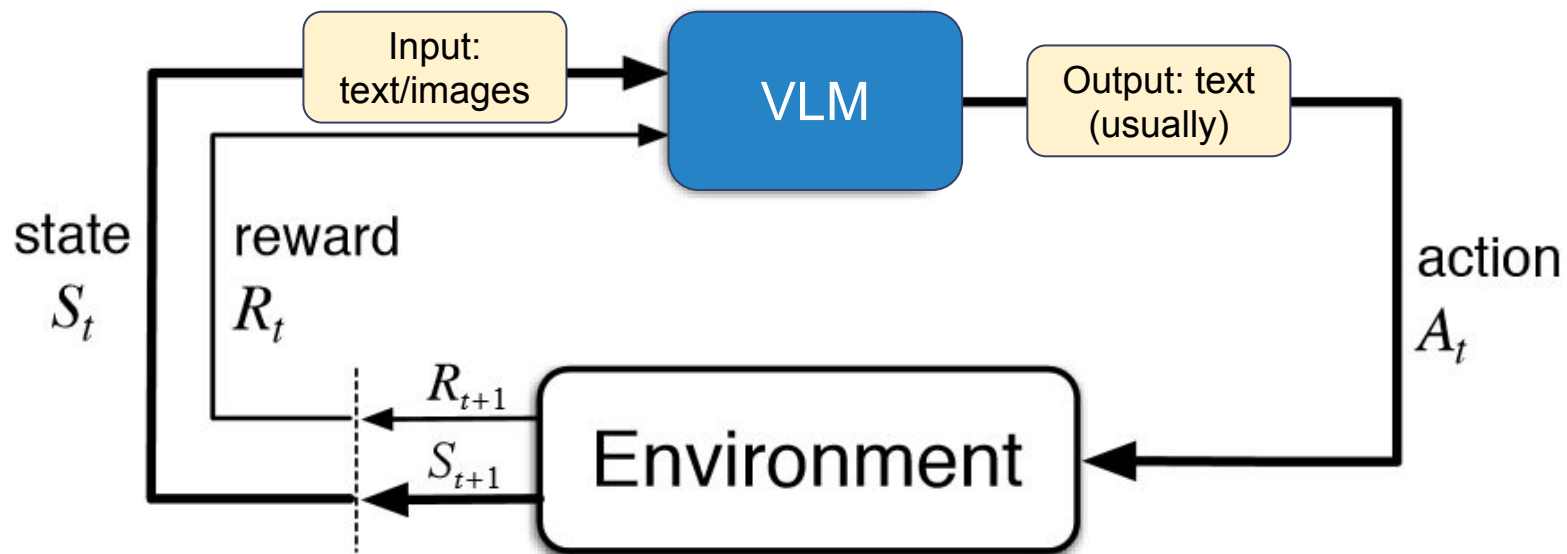


# Recall: VLM Agents

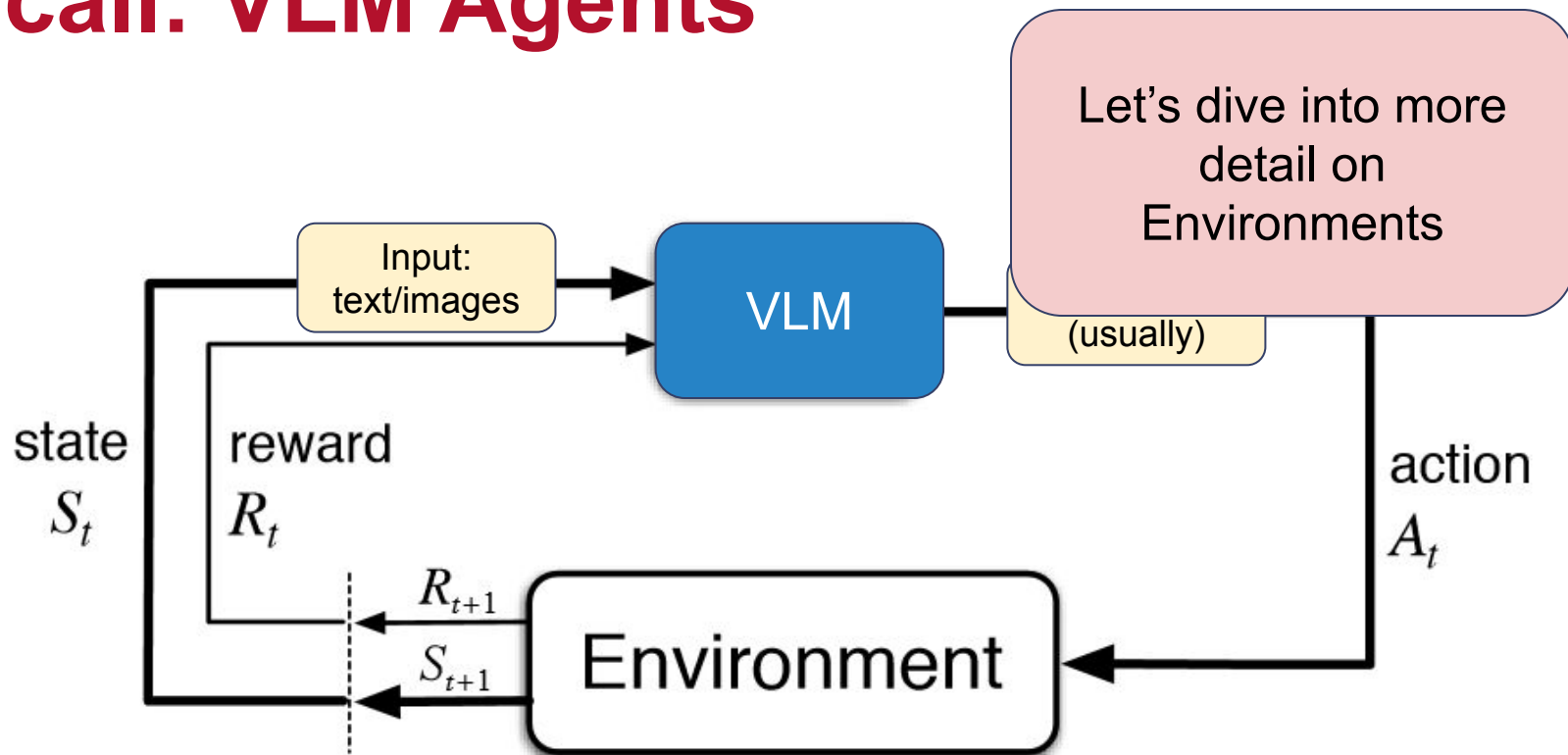


Agent

# Recall: VLM Agents



# Recall: VLM Agents

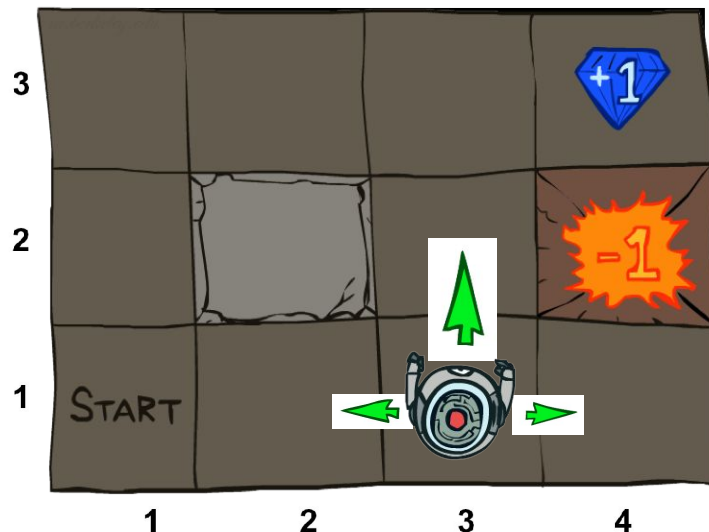


# Clarifying Terms

- When I say RL sometimes I mean
  - The science of understanding decision making agents in environments
- Sometime it means, a specific set of algorithms for maximizing reward from experience
  - E.g. REINFORCE, Q-Learning, PPO are RL algorithms
  - Behavior cloning is not technically an RL algorithm, but we often talk about it in RL classes because it's an algorithm for training decision making agents
- VLM Agents require an understanding of the first (it is about decision making)
- VLM Agents may not use RL algorithms (can use BC or even just prompting)

# Environments Defined by MDPs

- An MDP is defined by:
  - A **set of states**  $s \in S$
  - A **set of actions**  $a \in A$
  - A **transition function**  $T(s, a, s')$ 
    - Probability that  $a$  from  $s$  leads to  $s'$ , i.e.,  $P(s' | s, a)$
    - Also called the model or the dynamics
  - A **reward function**  $R(s, a, s')$ 
    - Sometimes just  $R(s)$ ,  $R(s,a)$ , or  $R(s')$
  - A **start state**
  - Maybe a **terminal state**



Slide credit: Daniel Brown

# What is Markov about MDPs?

- “Markov” generally means that given the present state, the future and the past are independent. **Conditional Independence!**
- For Markov decision processes, “Markov” means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0)$$

=

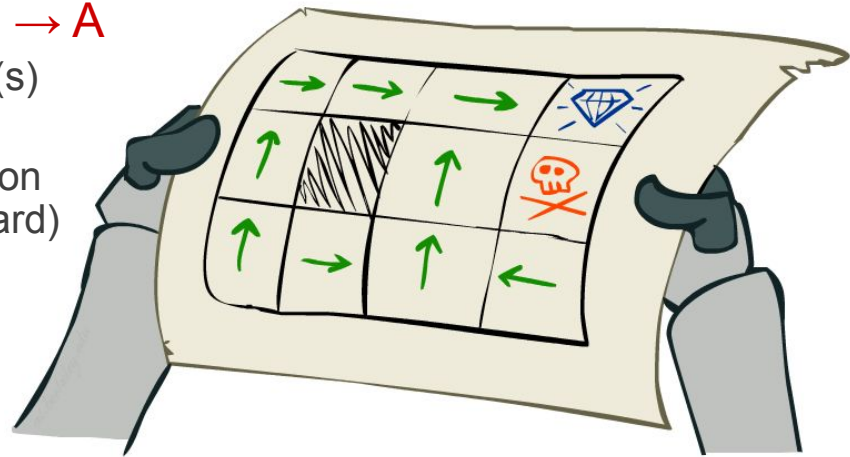
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$



Andrey Markov  
(1856-1922)

# Policies

- For MDPs, we want an optimal policy  $\pi^*: S \rightarrow A$ 
  - A policy  $\pi$  gives an action for each state:  $a = \pi^*(s)$
  - Depends only on the current state (not past states)
  - In an optimal policy, at every state, taking the action given by our policy leads to the most utility (reward)
- How do we learn this policy?



Optimal policy when  $R(s, a, s') = -0.03$   
for all non-terminals  $s$

# How do we learn a policy?

- Old-school Tabular RL
  - Given the full MDP and a discrete state/action space, calculate the optimal actions/values at each state
  - Essentially a DP problem
  - Sutton/Barto book goes into a lot of details about this



Try to figure out what each state "is worth"



# How do we learn a policy?

- Old-school Tabular RL
  - Given the full MDP and a discrete state/action space, calculate the optimal actions/values at each state
  - Essentially a DP problem
  - Sutton/Barto book goes into a lot of details about this
- Model-Free Methods
  - No access to underlying MDP, can't solve it with just computation
  - State/action spaces not necessarily discrete/finite
  - You needed to actually act to figure it out

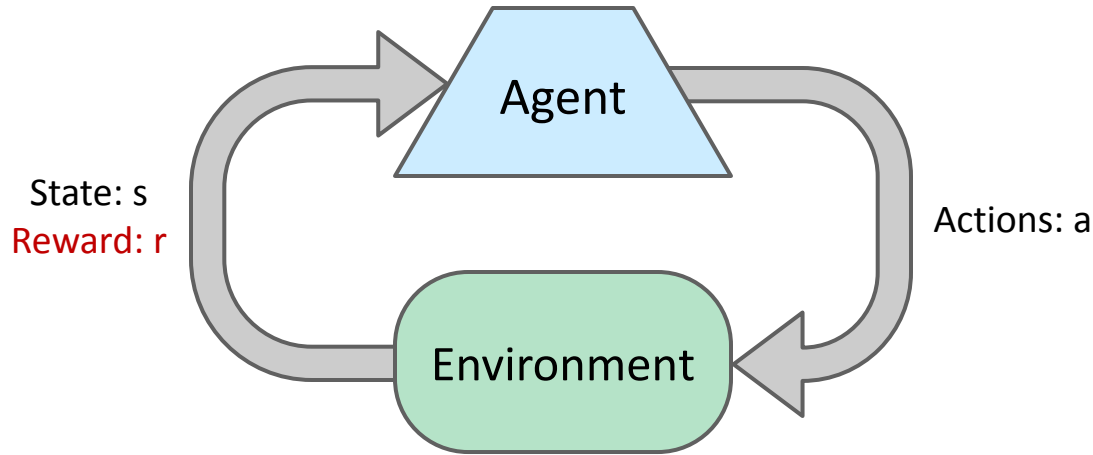


# How do we learn a policy?

- Old-school Tabular RL
  - Given the full MDP and a discrete state/action space, calculate the optimal actions/values at each state
  - Essentially a DP problem
  - Sutton/Barto book goes into a lot of details about this
- Model-Free Methods
  - No access to underlying MDP, can't solve it with just computation
  - State/action spaces not necessarily discrete/finite
  - You needed to actually act to figure it out
- Important ideas in reinforcement learning that come up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP



# Reinforcement Learning



- Basic idea:
  - 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**
  - All learning is based on observed samples of outcomes!

# Goal: Maximize Return/Utility

- Return: sum of all future rewards  $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$
- Often we care about \*discounted Return
  - Idea, reward later isn't as good as reward sooner
  - Want to get to goal/reward sooner ideally
  - Discount factor  $0 > \gamma \geq 1$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Utility (sometimes used interchangeably with return) is just expected utility

$$E[G_t] = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right]$$

# Goal: Maximize Return/Utility

- Return: sum of all future rewards  $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$

- Often we care about \*discounted Return

- Idea, reward
- Want to get
- Discount

In agents, utility is often just 0/1 at the end if we completed a task\*

\* But not always. E.g. you might want your agent to minimize things like cost, or give negative rewards for unsafe behavior

- Utility (sometimes)

$R_{t+k+1}$

y

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

# Example: Learning to Walk



Initial

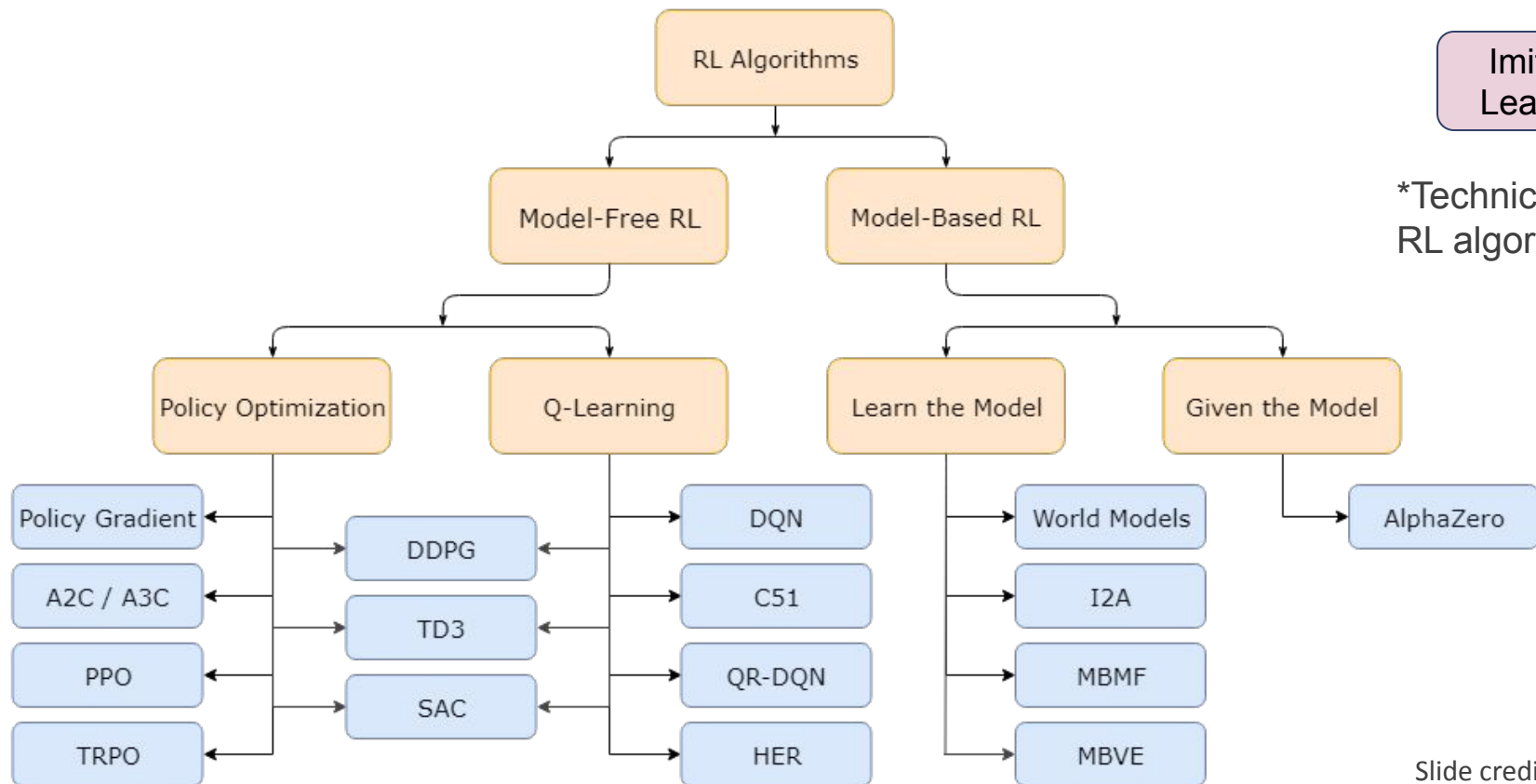


A Learning Trial



After Learning [1K Trials]

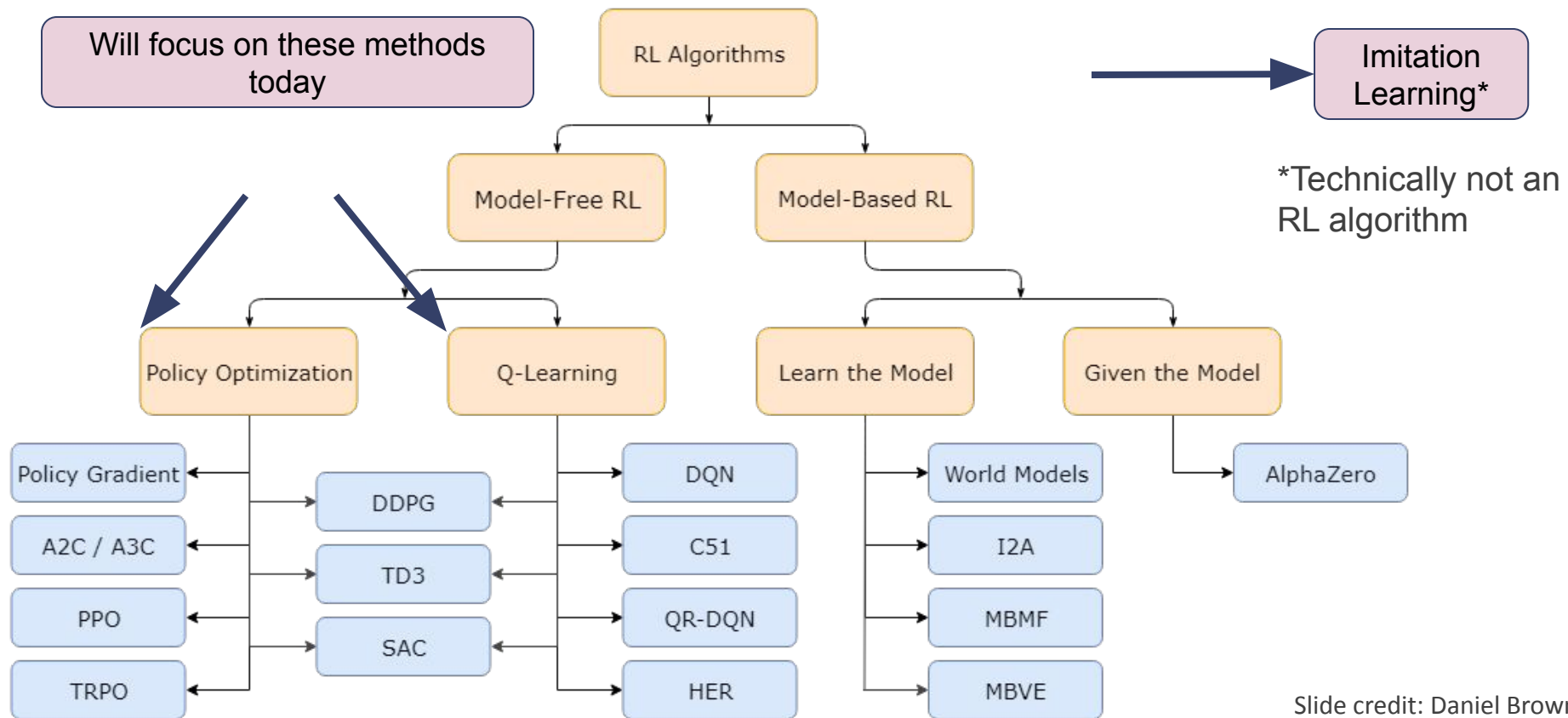
# Rough Taxonomy of RL Algorithms



Imitation Learning\*

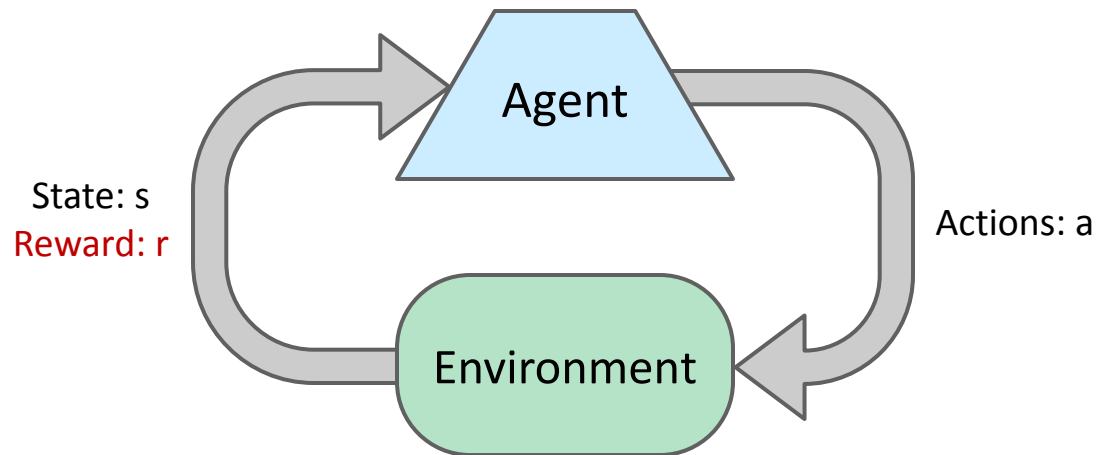
\*Technically not an RL algorithm

# Rough Taxonomy of RL Algorithms



# Reinforcement Learning

- Still assume there exists a Markov decision process (MDP):
  - A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s')$
  - A reward function  $R(s,a,s')$
- Still looking for a policy  $\pi(s)$



# Option 1: Imitation Learning

Common algorithm: Behavioral Cloning



Slide credit: Chelsea Finn

# Goal of Imitation Learning

**Data:** Given trajectories collected by an expert

“demonstrations”  $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T)\}$



(sampled from some unknown policy  $\pi_{\text{expert}}$ )

**Goal:** Learn a policy  $\pi_{\theta}$  that performs at the level of the expert policy, by mimicking it.

Example



- Dataset from human drivers
- Sensor readings + steering commands

# Imitation Learning - v1

0. Given demonstrations collected by an expert  $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T)\}$

1. For deterministic policy, supervised regression to the expert's actions

$$\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}, \mathbf{a}) \in \mathcal{D}} \|\mathbf{a} - \hat{\mathbf{a}}\|^2 \quad \text{where } \hat{\mathbf{a}} = \pi_{\theta}(\mathbf{s})$$

2. Deploy learned policy  $\pi_{\theta}$

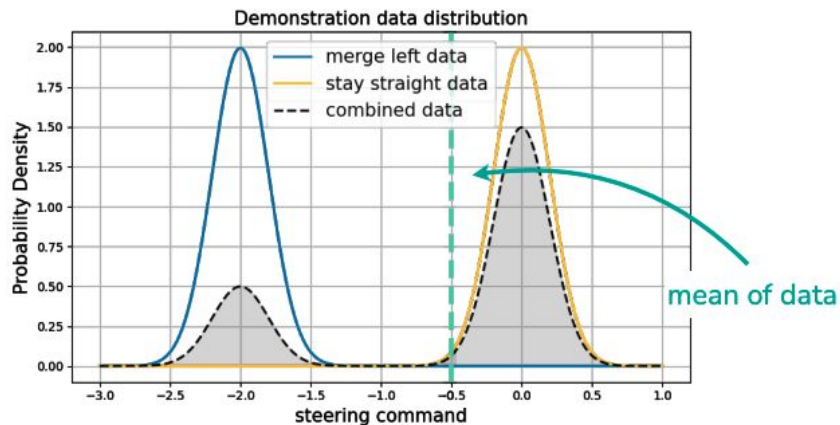
# What can go wrong?

## Example



- Dataset from human drivers
- Sensor readings + steering commands

Question: what might policy trained with  $\ell_2$ -regression do?



**How can we represent  
more than the mean?**

How often does this happen in practice? All the time!

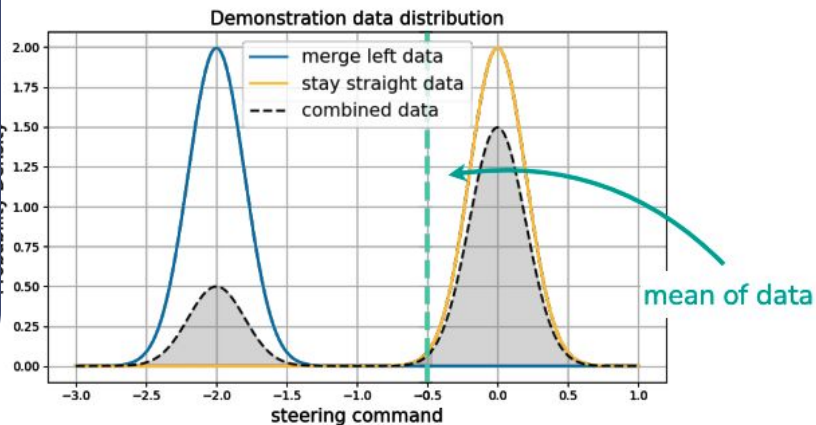
Esp. when data collected by multiple people.

# What can go wrong?

Confusingly for this class, often this is called a multimodal distribution (because it has multiple modes) But we often mean, multiple modes of input (e.g. vision and language) Hopefully from context its clear which one I mean

- sensor readings + steering commands

Q: what might policy trained with  $\ell_2$ -regression do?

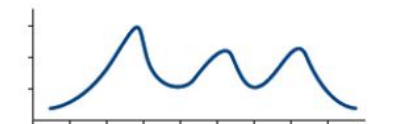


**How can we represent more than the mean?**

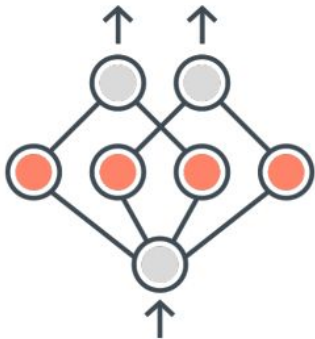
How often does this happen in practice? All the time!

Esp. when data collected by multiple people.

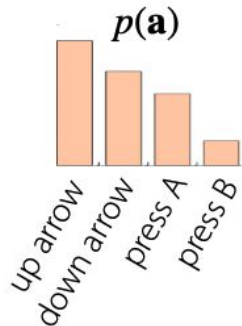
# Learning Distributions with NNs



parameters of distribution



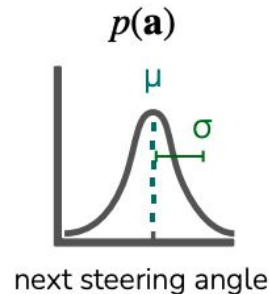
## 1D discrete actions



Neural net outputs  $p(\text{up}), p(\text{down}), \dots$  represent categorical distribution.

Maximally expressive

## Continuous actions



Neural net outputs  $\mu, \sigma$  to represent Gaussian distribution.

Not very expressive!

# Imitation Learning v2

Expressive policies

0. Given demonstrations collected by an expert  $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T)\}$

1. Train **generative model** of the expert's actions

$$\min_{\theta} - \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mathcal{D}} [\log \pi_{\theta}(\mathbf{a} | \mathbf{s})] \quad \text{with expressive distribution } \pi(\cdot | \mathbf{s})$$

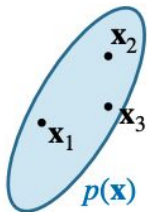
2. Deploy learned policy  $\pi_{\theta}$  maximize the **log probability** of the **demo actions** under the **policy**

# How this connects to agents

- v1
  - Works well for discrete actions
  - Works well for VLM agents where our action space is text
  - Most common way of learning policies in this course
  - Is actually just another way of doing SFT (we'll discuss how we do SFT on VLMs next lecture)
- v2
  - Matters most when output is something continuous like robotics actions
  - Will cover this more on Robotics unit

# What can still go wrong?

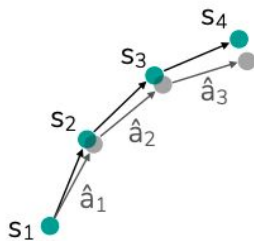
## Supervised learning



Inputs independent  
of predicted labels  $\hat{\mathbf{y}}$

## Compounding errors

## Supervised learning of behavior



Predicted actions affect  
next state.

Errors can lead to drift  
away from the data  
distribution!

Errors can then compound!

$$\underline{p_{expert}(\mathbf{s})} \neq \underline{p_{\pi}(\mathbf{s})}$$

states visited  
by expert

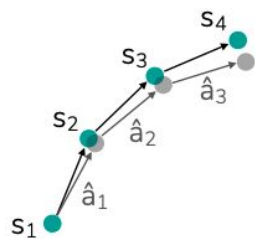
states visited by  
learned policy  $\pi$

“covariate shift”

# What can still go wrong?

## Compounding errors

### Supervised learning of behavior



Predicted actions affect next state.

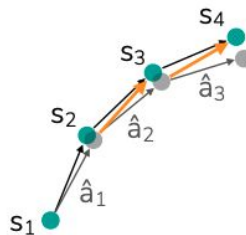
Errors can lead to drift away from the data distribution!

Errors can then compound!

$$p_{expert}(s) \neq p_{\pi}(s)$$

### Solutions?

1. Collect A LOT of demo data & hope for the best.
2. Collect **corrective behavior data**



# Other problems with IL

- Can only learn from positive data
  - Only have positive examples of what the agent should do
  - Can we learn from mistakes?
- Covariate drift
  - Discussed above (compounding errors, veering out of our data distribution)
  - Ideally we have some way to update our policy **online** (i.e. update the policy live in the actual environment)

# Other problems with IL

- Can only learn from positive data
  - Only have positive examples of what the agent should do
  - Can we learn from mistakes?
- Covariate drift
  - Discussed above (compounding errors, veering out of our data distribution)
  - Ideally we have some way to update our policy **online** (i.e. update the policy live in the actual environment)

Need true RL

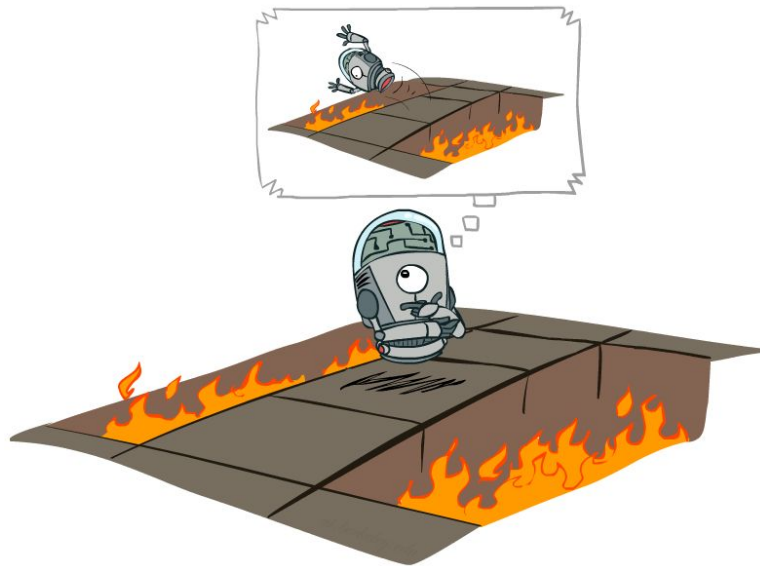
# Reinforcement Learning

- We still assume an MDP:
  - A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s')$
  - A reward function  $R(s,a,s')$
- Still looking for a policy  $\pi(s)$



# Idea: Learn how good an action is

- In state  $s$
- We want to know how good an action  $a$  is
- Let's write this as a function



$$\text{Value} = Q(s, a)$$

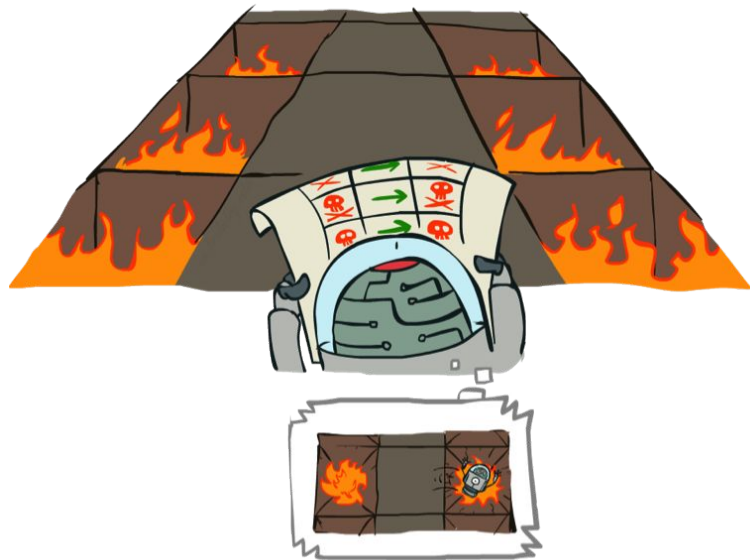
How much return we think we'll get

Given take action  $a$  in state  $s$

# Idea: Learn how good an action is

- In state  $s$
- We want to know how good an action  $a$  is
- Let's write this as a function

At each state  $s$ , choose action that gives best Q value



$$\text{Value} = Q(s, a)$$

How much return we think we'll get

Given take action  $a$  in state  $s$

# Idea: Learn how good an action is

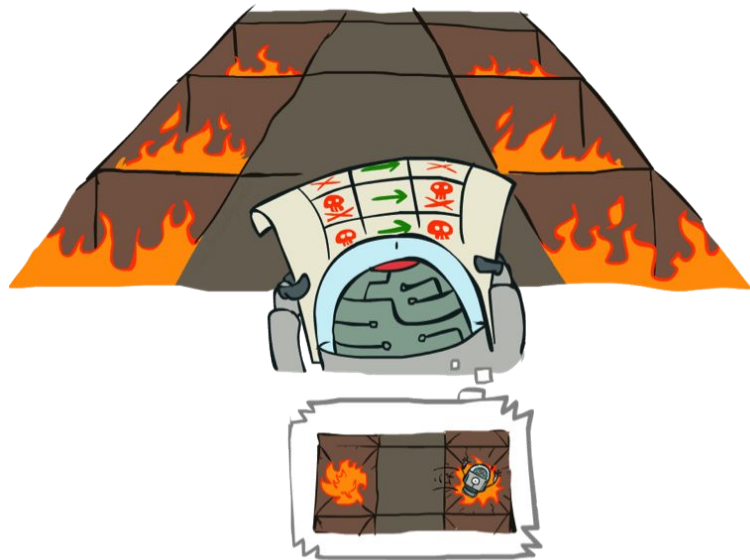
- In state  $s$
- We want to know how good an action  $a$  is
- Let's write this as a function

How do we learn  $Q$ ?

$$\text{Value} = Q(s, a)$$

How much return we think we'll get

Given take action  $a$  in state  $s$



# Tabular Q learning

- Make a table for all possible values  $s$  &  $a$  and set  $Q = 0$
- Update Q function using formula below (Bellman Equation)

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

- Tells you, what's the expected sum of rewards if you take action in state
- Guaranteed convergence
- See Sutton/Barto for more details
- Problems
  - Assumes we know world model (we usually don't)
  - Only works for discrete and small state/action space

# Updated Q learning

- Q function is now a neural network (like a VLM)
  - Takes in state/action as input
  - No longer constrained to tabular environments
- Approximate Q update

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$



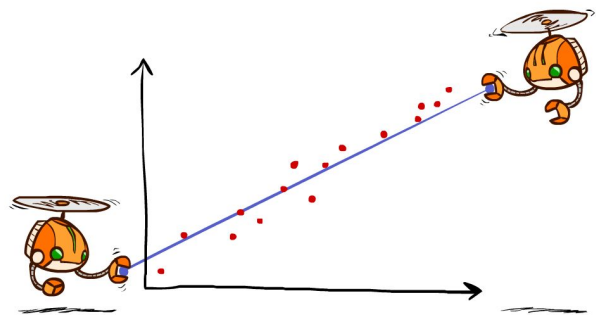
$$Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

- We can treat the Q-learning problem as a regression problem

# Update Q with gradient descent

Imagine we had only one point  $x$ , with features  $f(x)$ , target value  $y$ , and weights  $w$ :

$$\text{error}(w) = \frac{1}{2} \left( y - \sum_k w_k f_k(x) \right)^2$$
$$\frac{\partial \text{error}(w)}{\partial w_m} = - \left( y - \sum_k w_k f_k(x) \right) f_m(x)$$
$$w_m \leftarrow w_m + \alpha \left( y - \sum_k w_k f_k(x) \right) f_m(x)$$



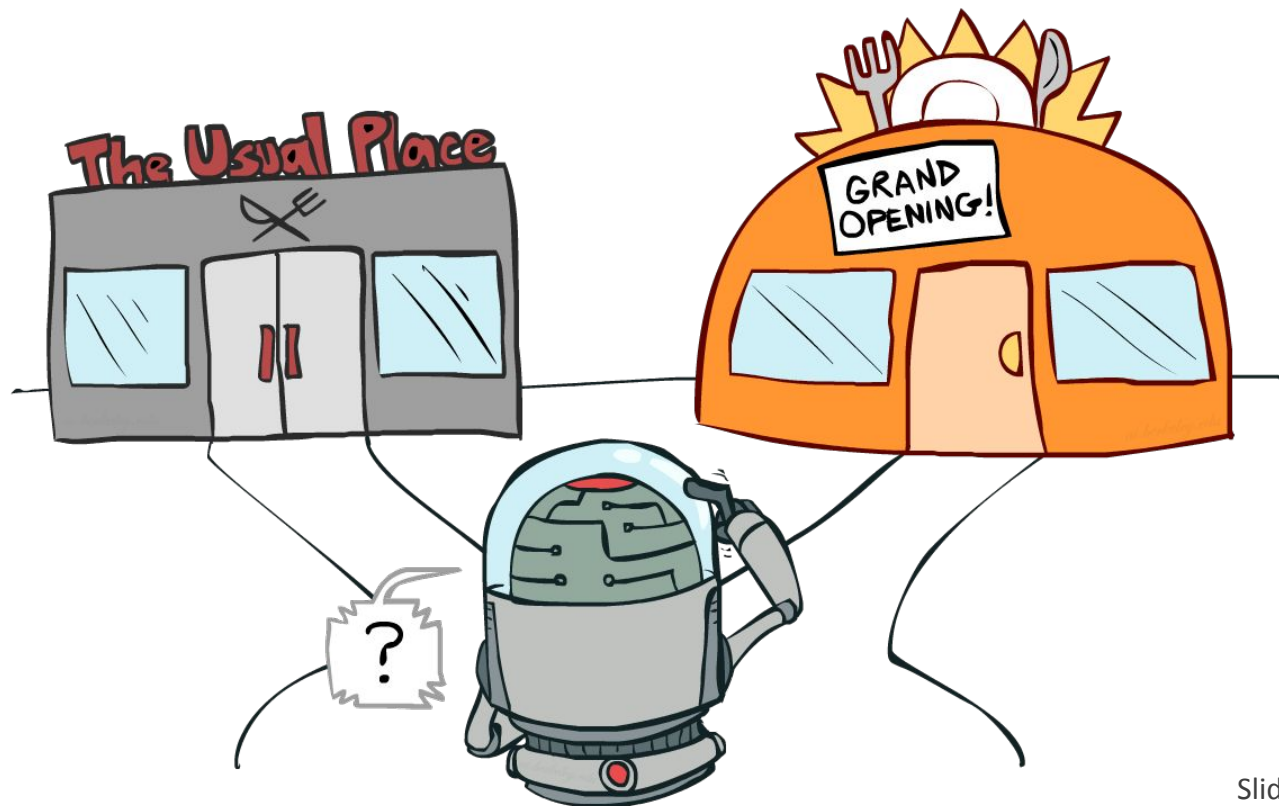
Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] f_m(s, a)$$

“target”

“prediction”

# Exploration vs. Exploitation



Slide credit: Daniel Brown

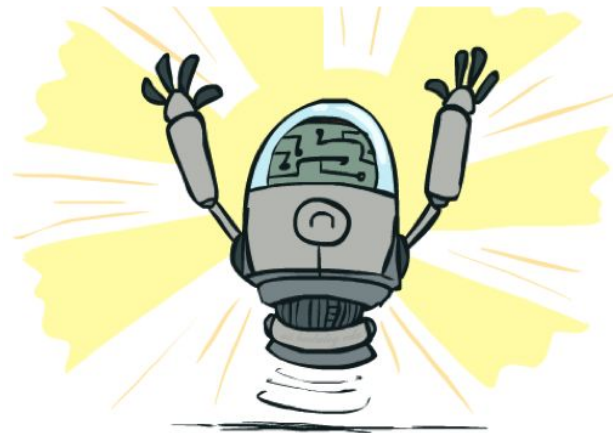
# How to Explore

- We want to visit many states/actions so we can learn a good Q function
- Several schemes for forcing exploration
  - Simplest: random actions ( $\epsilon$ -greedy)
    - Every time step, flip a coin
    - With (small) probability  $\epsilon$ , act randomly
    - With (large) probability  $1-\epsilon$ , act on current policy



# Q Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



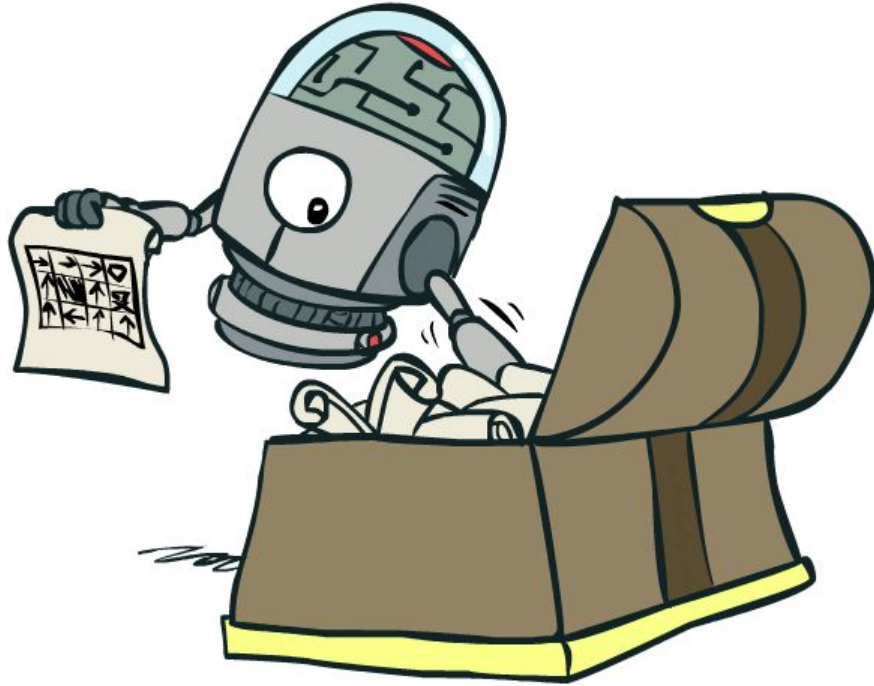
# Problems with Q Learning

- Only works for discrete action space
  - Useless for continuous actions like in e.g. robotics
  - Can also be tricky for very large action spaces (like in many agent problems)
- Off-policy
  - Great when we have a lot of off-policy data to use (e.g. trajectories collected from people)
  - Need to do a lot of random exploration to efficiently explore to find the optimal policy

# Two approaches to model-free RL

- Learn Q-Values
  - Train Q values to be consistent
  - Does not \*directly optimize performance
  - Uses an objective based on Bellman Equation
- Learn Policy Directly
  - Instead, just learn a parameterized function  $\pi_{\theta}$
  - Update weights of policy network to directly optimize policy

# Policy Search



# Preliminaries

- Trajectory (rollout, episode)  $\tau = (s_0, a_0, s_1, a_1, \dots)$

- $s_0 \sim \rho_0(\cdot)$ ,  $s_{t+1} \sim P(\cdot | s_t, a_t)$

- Rewards  $r_t = R(s_t, a_t, s_{t+1})$

- Finite-horizon undiscounted return of a trajectory

$$R(\tau) = \sum_{t=0}^T r_t$$

- Actions are sampled from a parameterized policy  $\pi_\theta$

- $a_t \sim \pi_\theta(\cdot | s_t)$

# Preliminaries

- Probability of a trajectory (rollout, episode)  $\tau = (s_0, a_0, s_1, a_1, \dots)$

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

- Expected Return of a policy  $J(\pi)$

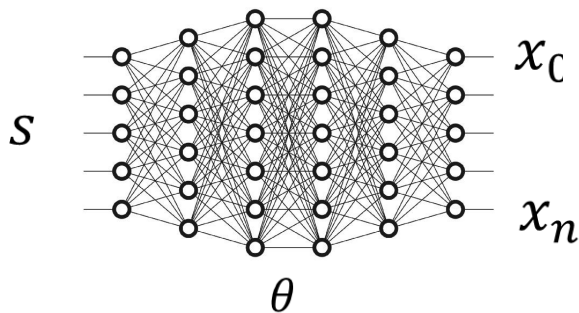
$$J(\pi) = \sum_{\tau} P(\tau|\pi) R(\tau) = E_{\tau \sim \pi}[R(\tau)]$$

- Goal of RL: Solve the following optimization problem

$$\pi^* = \operatorname{argmax}_{\pi} J(\pi)$$

# How should we parameterize our policy?

- We need to be able to do two things:
  - Sample actions  $a_t \sim \pi_\theta(\cdot | s_t)$
  - Compute log probabilities  $\log \pi_\theta(a_t | s_t)$
- Categorical (classifier over discrete actions)
  - Typically, you output a value  $x_i$  for each action (class) and then the probability is given by a softmax equation



$$\pi_\theta(a_i | s) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

# How should we parameterize our policy?

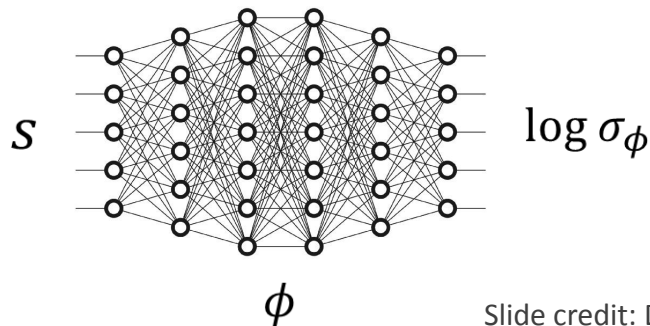
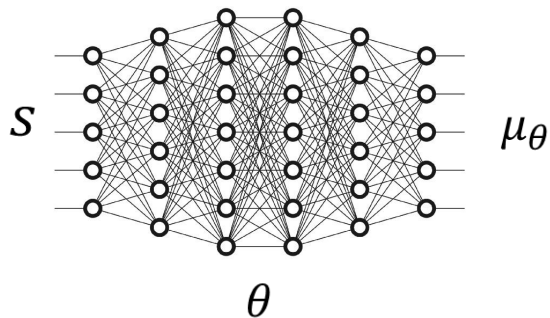
- Diagonal Gaussian (distribution over continuous actions)

$$a \sim N(\mu, \Sigma)$$

where  $\Sigma$  has non-zero elements only on the diagonal.

Thus, an action can be sampled as

$$a = \mu_{\theta}(s) + \sigma_{\phi}(s) \odot z, \quad z \sim N(0, I)$$



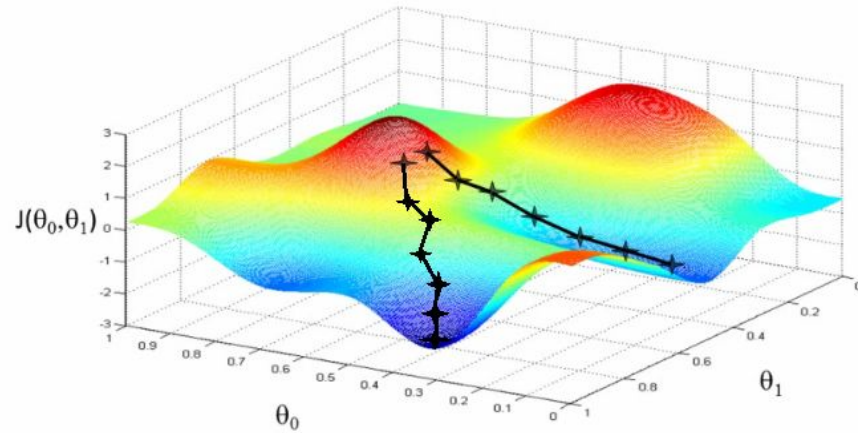
# Goal: Update Policy via Gradient Ascent

- We have a parameterized policy and we want to update it so that it maximizes the expected return.
- We want to find the gradient of the return with respect to the policy parameters and step in that direction.

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

Policy gradient

# Optimization: Max a function by “climbing” a hill

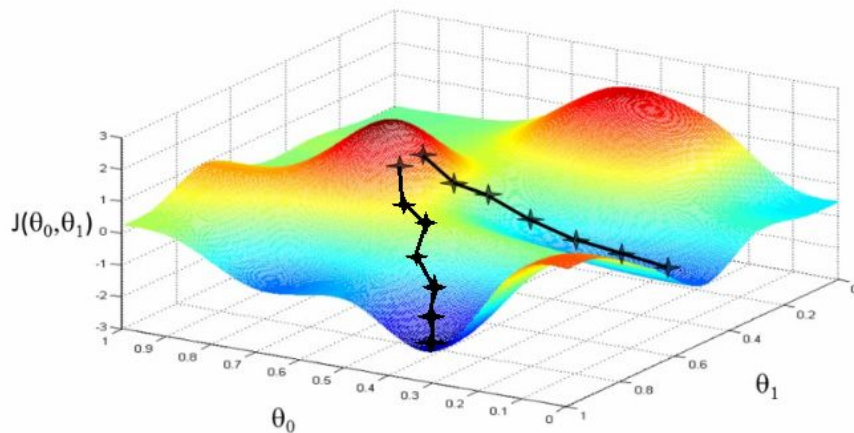


$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

Policy gradient

# Optimization: Max a function by “climbing” a hill

Use gradient to move in the direction of highest values



$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

Policy gradient

# Derivation of Policy Gradient

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \nabla_{\theta} E_{\tau \sim \pi_{\theta}} [R(\tau)] \\ &= \nabla_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau) \\ &= \sum_{\tau} \nabla_{\theta} P(\tau|\theta) R(\tau) \\ &= \sum_{\tau} P(\tau|\theta) \nabla_{\theta} \log P(\tau|\theta) R(\tau) \\ &= E_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log P(\tau|\theta) R(\tau)] \\ &= E_{\tau \sim \pi_{\theta}} [\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) R(\tau)]\end{aligned}$$

See <https://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes14.pdf>  
for details of derivation

Slide credit: Daniel Brown

# The Policy Gradient (REINFORCE)

- We can now perform gradient ascent to improve our policy!

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right]$$

Estimate with a sample mean over a set  $D$  of policy rollouts given current parameters

$$\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau)$$

# How would you implement this?

1. Start with random policy parameters  $\theta_0$
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

$$a_t \sim \pi_\theta(\cdot | s_t) \Rightarrow (s_0, a_0, r_0, s_1, a_1, r_1, \dots, r_T, s_{T+1})$$

$$D = \{\tau_1, \dots, \tau_N\}, \quad R = \{R(\tau_1), \dots, R(\tau_N)\}$$

3. Compute policy gradient

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) R(\tau) \right]$$

4. Update policy parameters

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \Big|_{\theta_k}$$

# Policy Gradient RL Algorithms

- We can directly update the policy to achieve high reward.
- Pros:
  - Directly optimize what we care about: Utility!
  - Naturally handles continuous action spaces!
  - Can learn specific probabilities for taking actions.
  - Often more stable than value-based methods (e.g. DQN).
- Cons:
  - On-Policy -> Sample-inefficient we need to collect a large set of new trajectories every time the policy parameters change.
  - Q-Learning methods are usually more data efficient since they can reuse data from any policy (Off-Policy) and can update per sample.

# Actor Critic

I rotate  
the piece



Really bad  
action

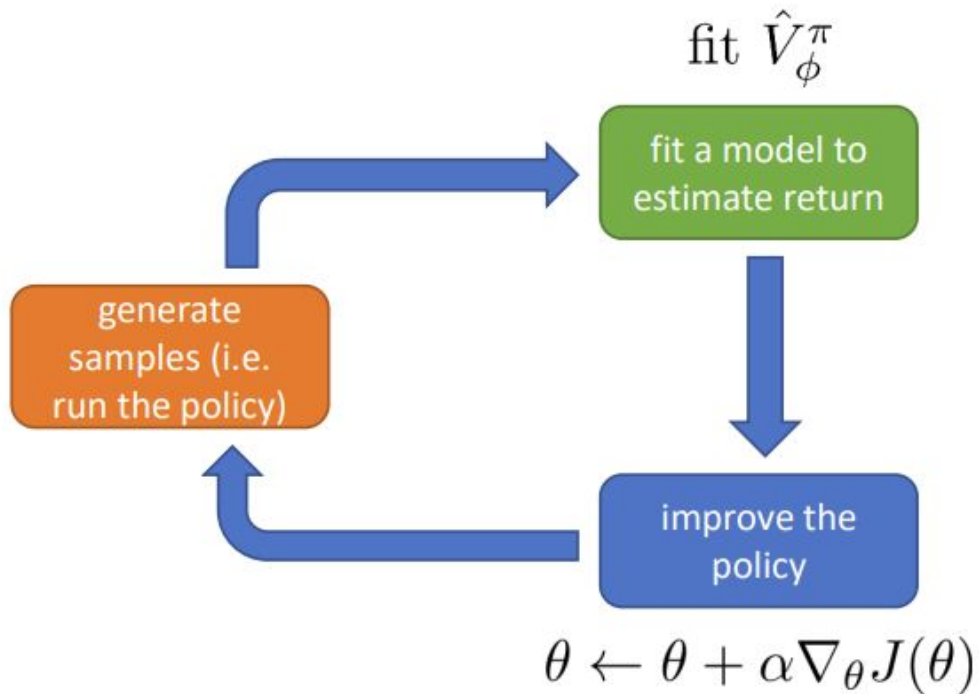


Actor



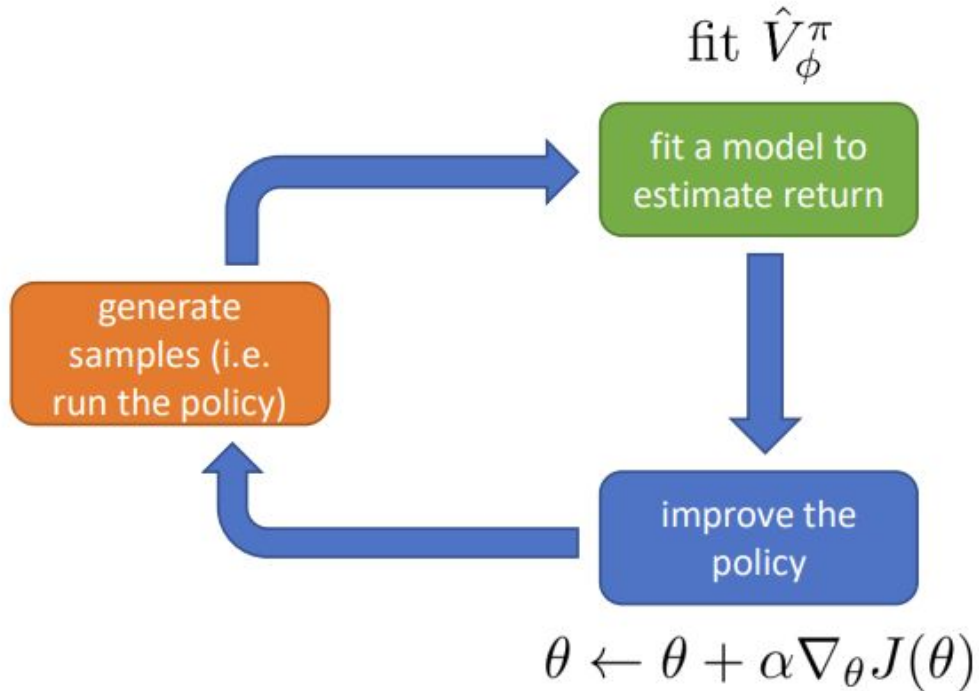
Critic

# Actor Critic



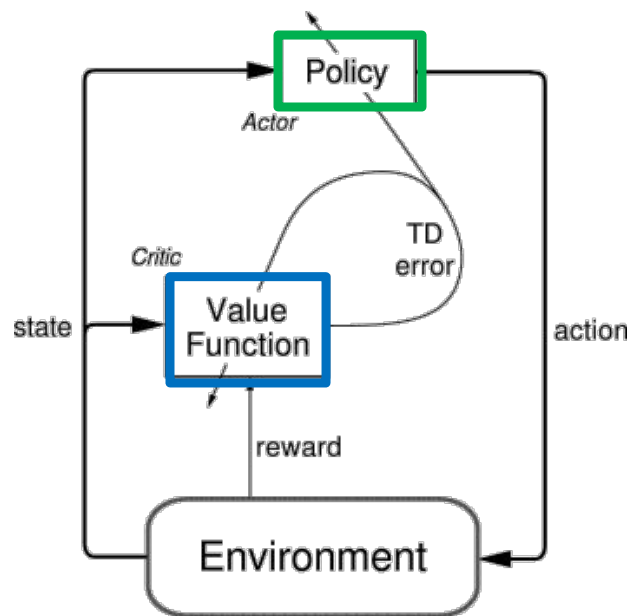
# Actor Critic

Combination of the two ideas of RL we've seen so far



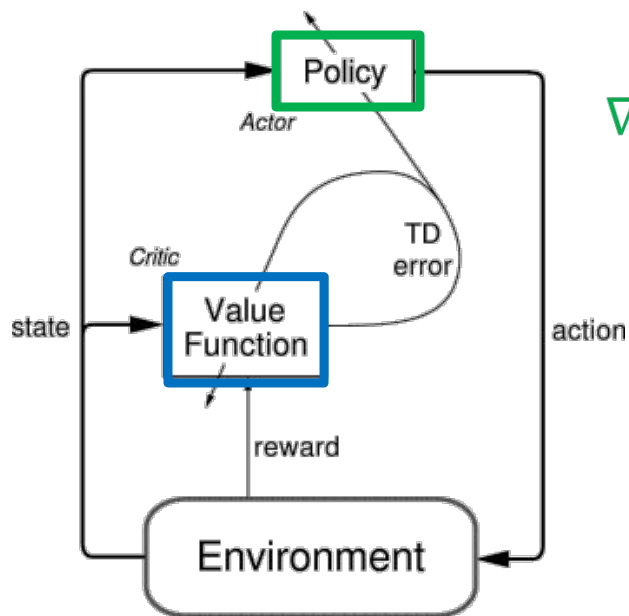
# Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



# Actor Critic Algorithms

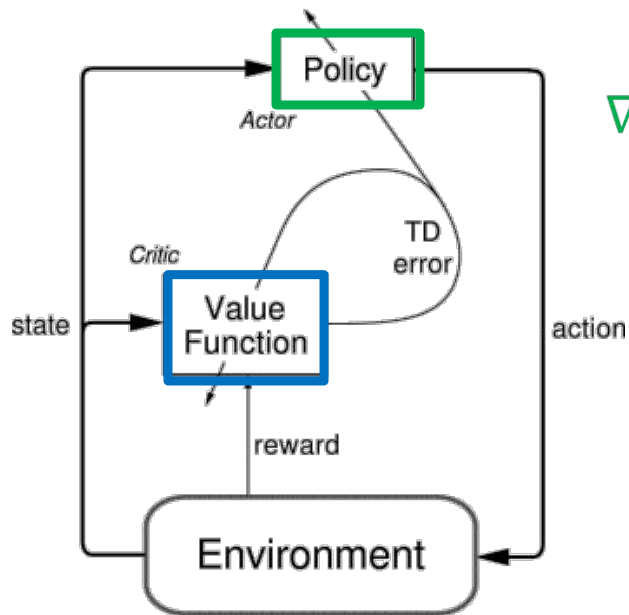
- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_w^{\pi_{\theta}}(s_t, a_t) \right]$$

# Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



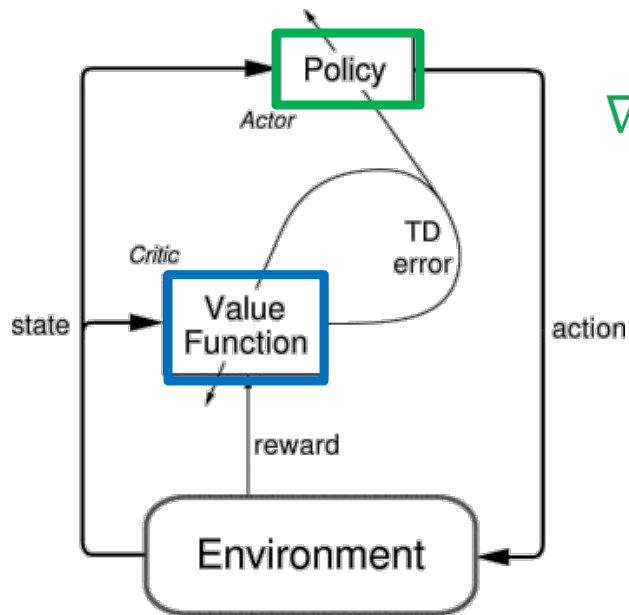
$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_w^{\pi_{\theta}}(s_t, a_t) \right]$$

Recall, before we directly used reward here

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right]$$

# Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function

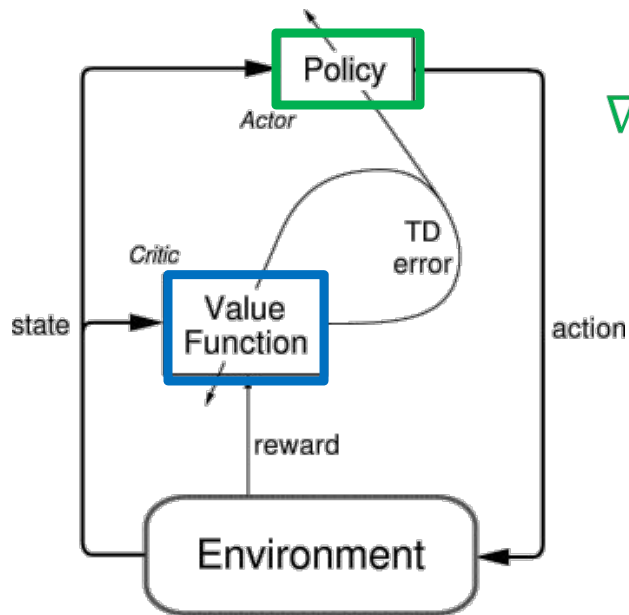


$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_w^{\pi_{\theta}}(s_t, a_t) \right]$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

# Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_w^{\pi_{\theta}}(s_t, a_t) \right]$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

$$\delta = (r_t + \gamma Q_w^{\pi_{\theta}}(s_{t+1}, a_{t+1}) - Q_w^{\pi_{\theta}}(s_t, a_t))$$

$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_{\theta} Q_w^{\pi_{\theta}}$$

Slide credit: Daniel Brown

# Q Actor Critic Algorithm Pseudo Code

---

## Algorithm 1 Q Actor Critic

---

Initialize parameters  $s, \theta, w$  and learning rates  $\alpha_\theta, \alpha_w$ ; sample  $a \sim \pi_\theta(a|s)$ .

**for**  $t = 1 \dots T$ : **do**

    Sample reward  $r_t \sim R(s, a)$  and next state  $s' \sim P(s'|s, a)$

    Then sample the next action  $a' \sim \pi_\theta(a'|s')$

    Update the policy parameters:  $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$ ; Compute the correction (TD error) for action-value at time  $t$ :

$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

    and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$

    Move to  $a \leftarrow a'$  and  $s \leftarrow s'$

**end for**

---

Adapted from Lilian Weng's post "Policy Gradient algorithms"

# Why Do This?

- Vanilla Policy Gradients has high variance
  - Directly uses return in updates which can have very high variance
  - Learning a value function instead greatly reduces noise in gradient updates
- Often faster convergence
  - Can update every few steps, not just at end of episodes when we get full return
- Separately estimating an estimate of how good a state/action pair is may help with policy learning

# Proximal Policy Optimization (PPO)

## Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov  
OpenAI

{joschu, filip, prafulla, alec, oleg}@openai.com

# Proximal Policy Optimization (PPO)

- One of the most popular deep RL algorithms
- Used to train ChatGPT and other LLMs

Motivation:

- Many Policy Gradient algorithms have stability problems.
- This can be avoided if we avoid making too big of a policy update.

---

**Algorithm 1** PPO-Clip

---

- 1: Input: initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.
- 4:   Compute rewards-to-go  $\hat{R}_t$ .
- 5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .
- 6:   Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

## Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.
- 4:   Compute rewards-to-go  $\hat{R}_t$ .
- 5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .
- 6:   Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**

Note: too many details to get into in such a short time

See

<https://huggingface.co/blog/dee-p-rl-ppo> for full details

# Lots of other tricks used

- Additional advantage normalization
- Early stopping with KL-divergence
- Etc.

The 37 Implementation Details of Proximal Policy Optimization:

<https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>

# Review: How to learn the policy

- Imitation Learning
  - Is especially effective / used in VLM agent problems
  - IL == BC == SFT
- Offline methods / Q learning
  - Learn the “goodness” of a state, action pair w/ Bellman updates
  - Follow highest Q values
- Policy Gradient Methods
  - Directly optimize policy network
  - Improve with actor and critic
  - Improve further with more tricks (PPO)
  - PPO is most common RL algorithm for LLMs

# Assumptions we've made

- Markov Assumption
  - For environment - only your current state matters, not how you got there
  - For policy - you only need to look at your current state
- Fully Observable
  - Everything about the environment is full observed and captured in your state  $s$

# Example: Computer Use

The image displays a web browser window on the left and a terminal window on the right. The browser shows the 'One Stop Market' website with a search bar and a grid of products. The terminal window shows the output of a Python script that uses GPT-4 for vision analysis to interact with the website.

**Browser Window (Left):**

- URL: [metis.lli.cs.cmu.edu:7770](https://metis.lli.cs.cmu.edu:7770)
- Page Title: One Stop Market
- Search Bar: Search entire store here...
- Navigation: My Account, My Wish List, Sign Out
- Categories: Personal Care, Sports & Outdoors, Clothing, Shoes & Jewelry, Home & Kitchen, Office Products, Tools & Home Improvement, Household, Patio, Lawn & Garden, Electronics, Cell Phones & Accessories, Video Games, Grocery & Gourmet Food
- Product Grid:

  - Red Gingerbread House Kit Pack, 17 oz., Pack of 2, Total 1 Review
  - V8 +Energy, Healthy Energy Drink, Steady Energy from Black and Green Tea, Pomegranate Blueberry, 8 Ounce Can, Pack of 24 (12 Reviews, \$14.47)
  - Elmwood Inn Fine Tea, Orange Vanilla Caffeine-free Fruit Infusion, 16-Ounce Pouch (4 Reviews, \$19.36)
  - Belle Of The Ball Princess Sprinkle Mix | Wedding Colorful Sprinkles | Cake Cupcake Cookie Sprinkles | Ice Cream Candy Sprinkles | Yellow Gold Red Royal Red Rose Icing Flowers Decorating Sprinkles, 8OZ (12 Reviews, \$23.50)
  - So Delicious Dairy Free CocoWhip Light, Vegan, Non-GMO Project Verified, 9 oz. Tub (12 Reviews, \$15.62)

**Terminal Window (Right):**

```
visualwebarena_mem - node - python run_eval.py --instruction_path
Predicted Next State: The next state will show the comment section with a
led "Interested" containing the text "Do you have a USB-C cable?".
Action: In summary, the next action I will perform is '''type [20] [Interes
owned by '''type [21] [Do you have a USB-C cable?]' and then '''click [22]
Running gpt-4-vision-preview...
Plan: The image of the Seagate 1tb external hard drive shows a cable, but I
are if the cable is USB-C or another type. Since the objective requires con
a USB-C cable, and the previous actions show that I have already asked the
cy have a USB-C cable. I will wait for the seller's response before proced
comment has been approved, no further action is required at this moment.
Summary: The task of inquiring about the USB-C cable has been completed, an
for the seller's response.
Predicted Next State: The next state will remain the same until the seller
the comment.
Action: In summary, the next action I will perform is '''stop [1]'''
2024-03-20 02:47:47,765 - INFO - [Result] (PASS) config_files/test_classifi
Saving jsons...
Iteration 3/5
2024-03-20 02:47:48,186 - INFO - [Config file]: config_files/test_classific
2024-03-20 02:47:48,186 - INFO - [Intent]: What is the email of the seller
palette on this page?
Running gpt-4-vision-preview...
Plan: The objective is to find the email of the seller of the red palette.
STARLIGHT Palette listed for $10 seems to be the red palette mentioned. The
ID [24] corresponds to this palette. To get the seller's email, I should cl
image or the associated link with ID [25]. By doing so, I expect to be take
led view where the seller's contact information is typically provided.
Summary: Clicking on the image or link of the Lancoex STARLIGHT Palette will
me to a detailed page where the seller's email can be found, thus progress
the objective.
Predicted Next State: After clicking on the listing for the "Lancoex STARLI
Brand How" the next state will display the detailed view of the listing,
images of the palette, a description, options to contact the seller or share
e and a section for comments. There will also be information about the sell
& their name and email address for contact purposes.
Action: In summary, the next action I will perform is '''click [25]'''
Running gpt-4-vision-preview...
Plan: The objective is to find the email of the seller of the red palette.
on this page. I have already clicked on the listing for the Lancoex STARLI
which is the red palette mentioned in the objective. The seller's contact i
available on the page, including the email address. Since obtaining the em
was the objective, I can now complete the task by issuing the stop action w
1 address as the answer.
Summary: The action retrieves the seller's email from the listing page of t
te, fulfilling the task's objective to obtain this contact information.
Predicted Next State: The episode will end after the stop action is issued
reaction provided will be returned to the user.
Action: In summary, the next action I will perform is '''stop [sofia_kumar@
Saving jsons...
Iteration 4/5
2024-03-20 02:48:52,240 - INFO - [Result] (PASS) config_files/test_classifi
Input Image: https://images.pexels.com/photos/5947093/pexels-photo-5947093
2024-03-20 02:48:55,759 - INFO - [Config file]: config_files/test_shopping
2024-03-20 02:48:55,759 - INFO - [Intent]: Find me powder to make the batter
the same as the picture.
]
```

# Example: Computer Use

The screenshot displays a web browser window with the URL `metis.lli.cs.cmu.edu:7770` and the page title "One Stop Market". The page shows a grid of products with their images, descriptions, prices, and "Add to Cart" buttons. The products include:

- Red Gingerbread House Kit Pack, 17 oz., Pack of 2, Total \$14.47
- V8 +Energy, Healthy Energy Drink, Steady Energy from Black and Green Tea, Pomegranate Blueberry, 8 Ounce Can, Pack of 24 \$19.36
- Elmwood Inn Fine Teas, Orange Vanilla Caffeine-free Fruit Infusion, 16-Ounce Pouch \$19.36
- Belle Of The Ball Princess Sprinkle Mix | Wedding Colorful Sprinkles | Cake Cupcake Cookie Sprinkles | Ice cream Candy Sprinkles | Yellow Gold Red Royal Red Rose Icing Flowers Decorating Sprinkles, 8OZ \$23.50
- So Delicious Dairy Free CocoWhip Light, Vegan, Non-GMO Project Verified, 9 oz. Tub \$15.62

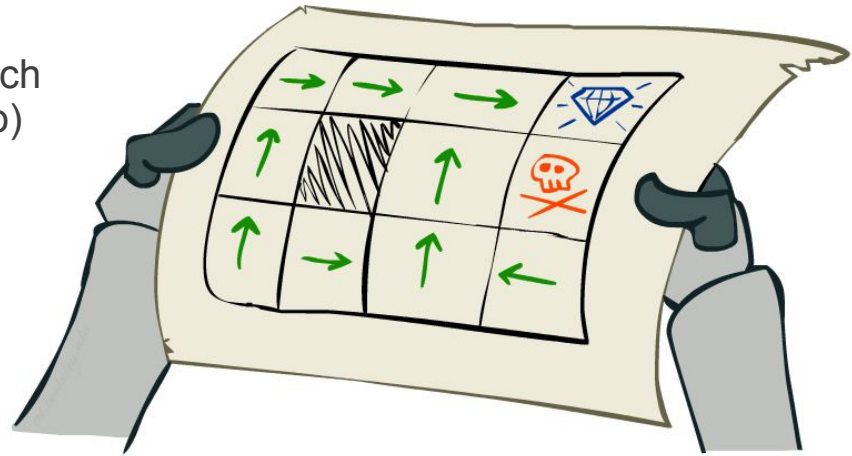
On the right side, a terminal window shows the following text:

```
Predicted Next State: The next state will show the content section with a red "interested" containing the text "do you have a USB-C cable?".
Action: In summary, the next action I will perform is ""type [20] [Interested owned by ""type [21] [Do you have a USB-C cable?]" and then ""click [22] [Running gpt-4-vision-preview...
Summary: The action retrieves the seller's email from the listing page of the te, fulfilling the task's objective to obtain this contact information.
Predicted Next State: The episode will end after the stop action is issued reaction provided will be returned to the user.
Action: In summary, the next action I will perform is ""stop [sofia_kumar@
2024-03-20 02:48:52,240 - INFO - [Result] (PASS) config_files/test_classifff
Saving jsons...
Iteration 4/5
Input Image: https://images.pexels.com/photos/5947093/pexels-photo-5947093-
2024-03-20 02:48:55,759 - INFO - [Config file] config_files/test_shopping
2024-03-20 02:48:55,759 - INFO - [Intent]: Find me powder to make the butter
the same as the picture.
]
```

A pink callout box overlaid on the terminal contains the text: "Fully Observable Assumption is clearly not correct (need to find information) Need to recall previous States for Agent to succeed".

# LLM Policies In Practice

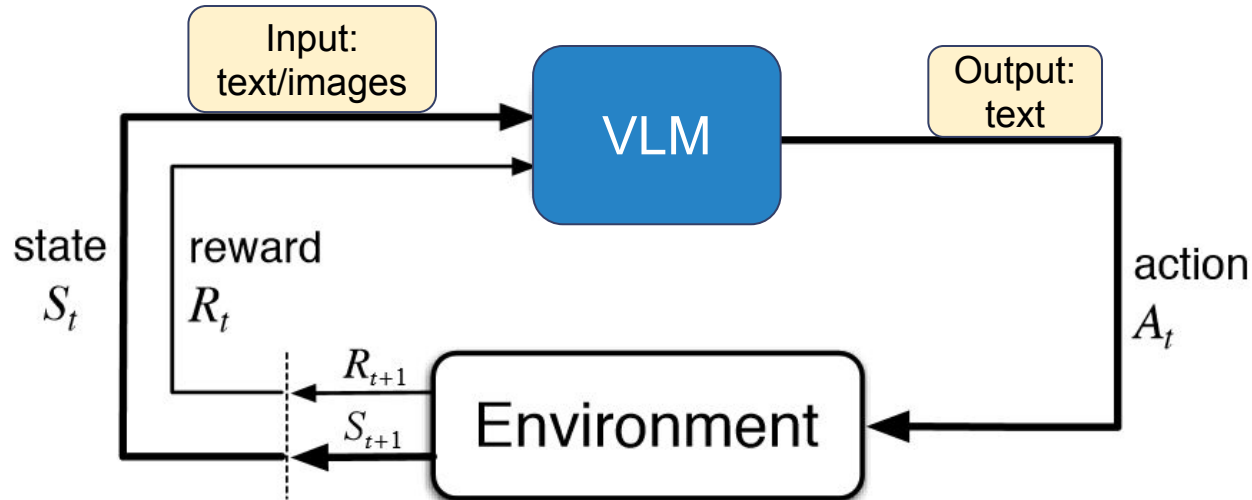
- $\pi(o_t, \text{history}(o_{0:t-1}, a_{0:t-1}), i)$ 
  - VLM policy takes the current observation  $o_t$  (which doesn't necessarily contain all relevant state info)
  - Some (usually condensed) history of past observations and actions the agent took
  - Possibly other information (may depend on your problem)
- Fortunately
  - VLMs are transformer based and quite good at handling sequential information



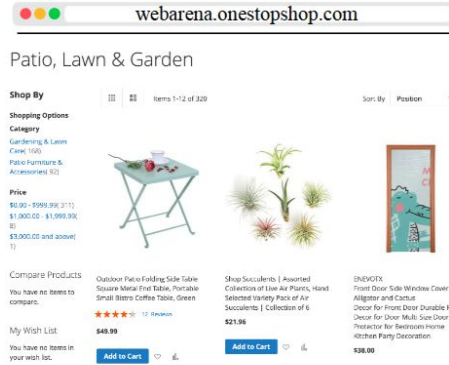
Optimal policy when  $R(s, a, s') = -0.03$   
for all non-terminals  $s$

# What this means for VLM Agents

- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)



# Recall: Obs/Action Space of CU Agent



```

</li>
<div>
  <a href="#"...></a>
  <div class>
    <a href="#"...>Outdoor Patio ...
  </a>
  <div>
    <span>Rating:</span>
    <div>
      <span>82%</span>
    </div>
    <a href="#"...#reviews">12
  <span>Reviews</span></a>






```

RootWebArea 'Patio, Lawn ..'  
 link 'Image'  
 img 'Image'  
 link 'Outdoor Patio..'  
 LayoutTable ''  
 StaticText 'Rating:'  
 generic '82%'  
 link '12 Reviews'  
 StaticText '\$49.99'  
 button 'Add to Cart' focusable: True  
 button 'Wish List' focusable: ...  
 button 'Compare' focusable: ...

Postmail Forums Wiki

**/f/food**

Submissions Comments 0 Hot

-  [homemade] Obligatory Halloween Pumpkin Loaf!
-  [ate] Maple Pecan Croissant
-  [Homemade] Margherita pizza
-  [Homemade] Sichuanese Spicy Beef Noods!
-  [ate] Sushi platter

**Webpage with SoM of Interactable Elements**

...

[7] [A] [Comments]

[8] [BUTTON] [Hot]

[9] [IMG] [description: picture of a pumpkin]

[10] [A] [kneechalice]

...

**SoM Elements and Text Content**

Action Type <i>a</i>	Description
click [elem]	Click on element elem.
hover [elem]	Hover on element elem.
type [elem] [text]	Type text on element elem.
press [key_comb]	Press a key combination.
new_tab	Open a new tab.
tab_focus [index]	Focus on the i-th tab.
tab_close	Close current tab.
goto [url]	Open url.
go_back	Click the back button.
go_forward	Click the forward button.
scroll [up down]	Scroll up or down the page.
stop [answer]	End the task with an output.

# What this means for VLM Agents

- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)
- Most useful algorithms to know
  - Imitation learning probably most common
    - When we talk about LLMs, this is exactly the same as Supervised Fine Tuning (SFT)
  - When we do RL with agents, PPO is most commonly used

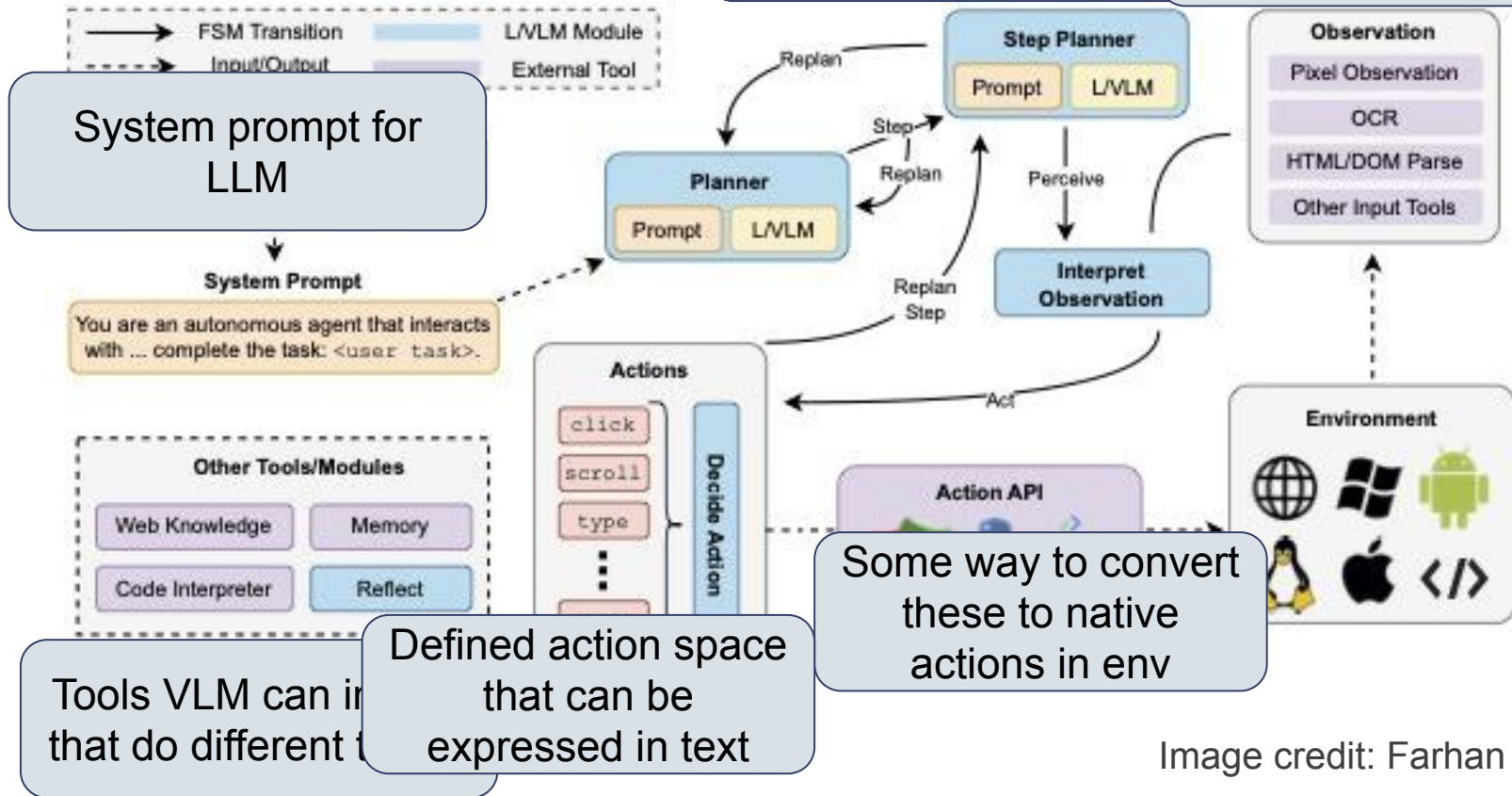
# What this means for VLM Agents

- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)
- Most useful algorithms to know
  - Imitation learning probably most common
    - When we talk about LLMs, this is exactly the same as Supervised Fine Tuning (SFT)
  - When we do RL with agents, PPO is most commonly used
- VLM agents also often work without any training!
  - Through prompting frameworks and other methods
  - Just relies on general ability of the base models!

# Recall: Firmament

Frameworks for LLMs  
be better at actions  
(reflect, plan, replan, e

Hand-crafted  
observations useful  
for VLM



System prompt for LLM

System Prompt  
You are an autonomous agent that interacts with ... complete the task: <user task>.

Tools VLM can interact with that do different things

Defined action space that can be expressed in text

Some way to convert these to native actions in env

# What this means for VLM Agents

- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)
- Most useful algorithms to know
  - Imitation learning probably most common
    - When we talk about LLMs, this is exactly the same as Supervised Fine Tuning (SFT)
  - When we do RL with agents, PPO is most commonly used
- VLM agents also often work without any training!
  - Through prompting frameworks and other methods
  - Just relies on general ability of the base models!

# What this means for VLM Agents

- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)
- Most useful algorithms to know
  - Imitation learning probably most common
    - When we talk about LLMs, this is exactly the same as Supervised Fine Tuning (SFT)
  - When we do RL with agents, PPO is most commonly used
- VLM agents also often work without any training!
  - Through prompting frameworks and other methods
  - Just relies on general ability of the base models!
  - **There will always be a limit on how good this will be / no way to make it better**

# Recommended Readings

- Reinforcement Learning an Introduction - Sutton & Barto
  - It's all good but for introduction to RL Ch1&3 are helpful
  - <http://incompleteideas.net/book/RLbook2020.pdf>
- Reinforcement Learning Courses
  - Stanford Deep RL Class: <https://cs224r.stanford.edu/>
  - Berkeley Deep RL Course: <https://rail.eecs.berkeley.edu/deeprlcourse/>
  - If you need to understand particular topic/algorithm, notes from these courses are really useful
- This was a very brief overview of topics
  - Some/none of details might matter depending on your project/application
  - Goal: get a 10,000 foot view so that when an RL topic comes up again, you know where to start looking

# Any Questions



Questions