Reinforcement Learning

CS6960 MultiModal LLM Agents

Kenneth Marino

Slide credits: Daniel Brown, Dan Klein and Pieter Abbeel, Chelsea Finn, others

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

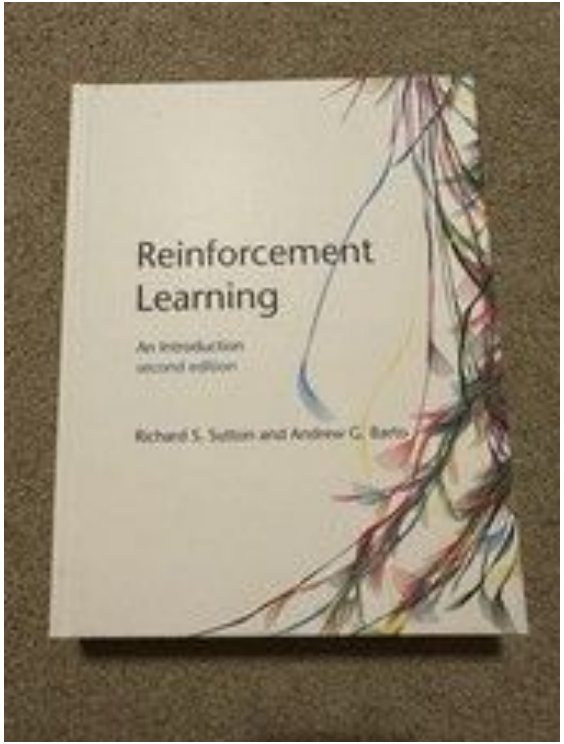
Recall: What are Agents?

- VLM Agents

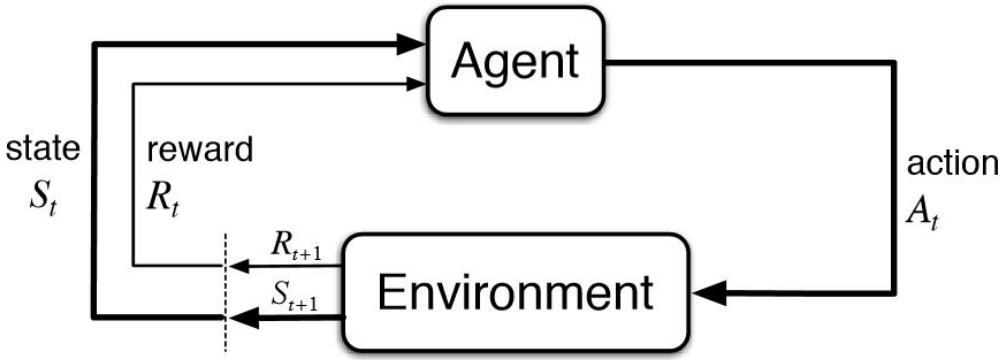
What Even is an Agent?

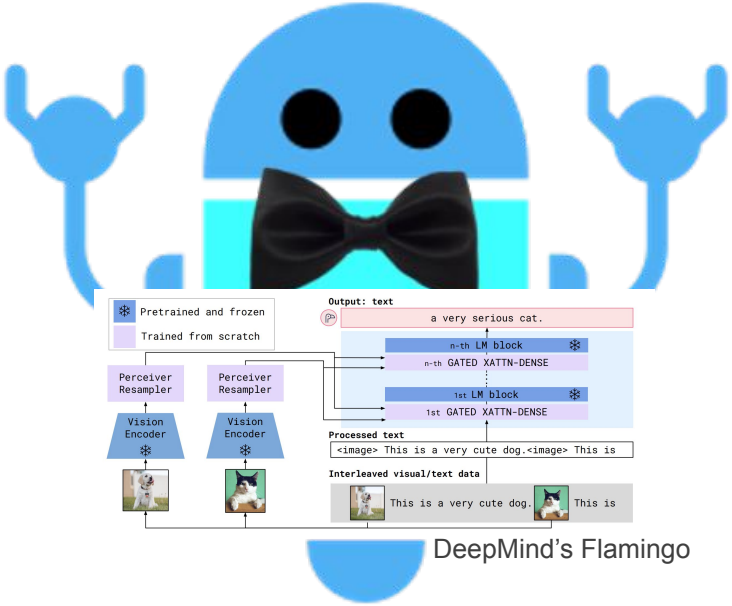
Recall: Classical Definition

- 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

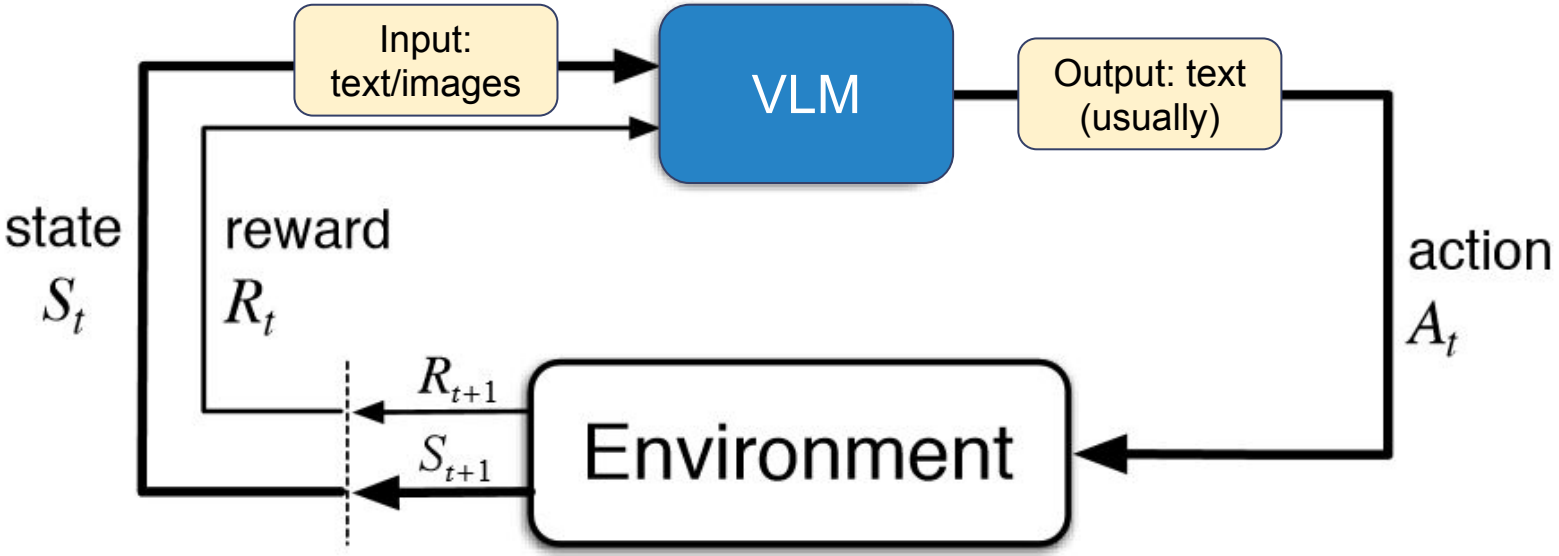




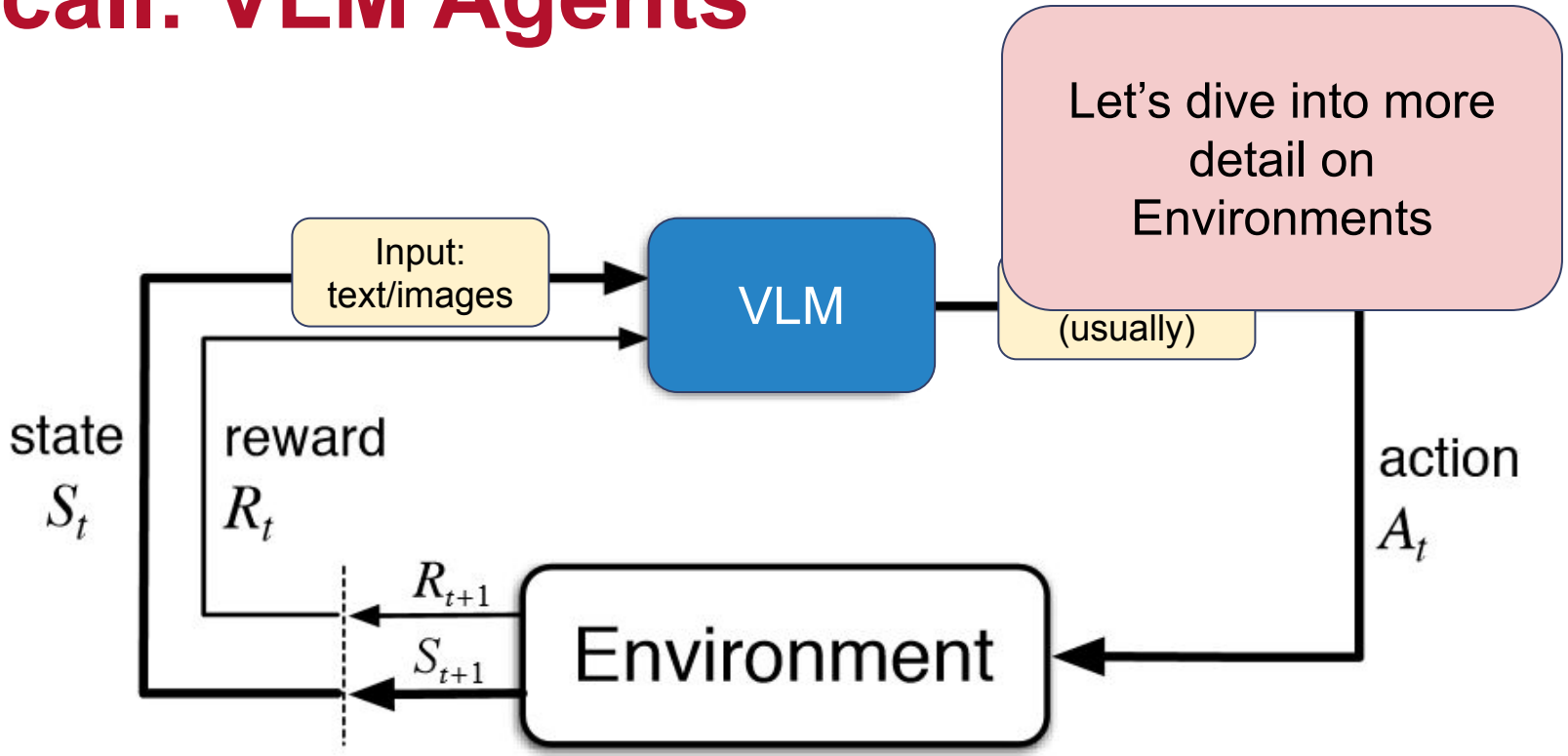
DeepMind's Flamingo

Agent

Recall: VLM Agents



Recall: VLM Agents



Clarifying Terms

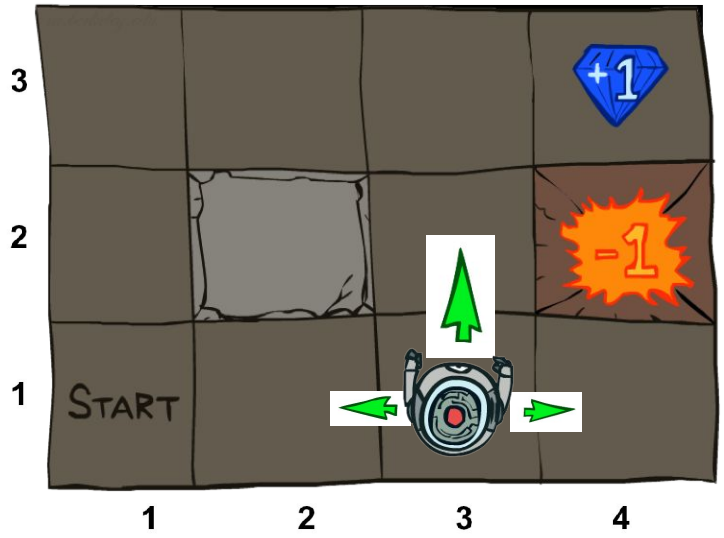
- When I say RL sometimes I mean
  - The science of understanding decision making agents in environments
- Sometime it means, a specific set of algorithms for maximizing reward from experience
  - E.g. REINFORCE, Q-Learning, PPO are RL algorithms
  - Behavior cloning is not technically an RL algorithm, 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)

Slide credit: Daniel Brown

[Demo – gridworld manual intro (L8D1)]

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

[Demo – gridworld manual intro (L8D1)]

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

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

$$=$$

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

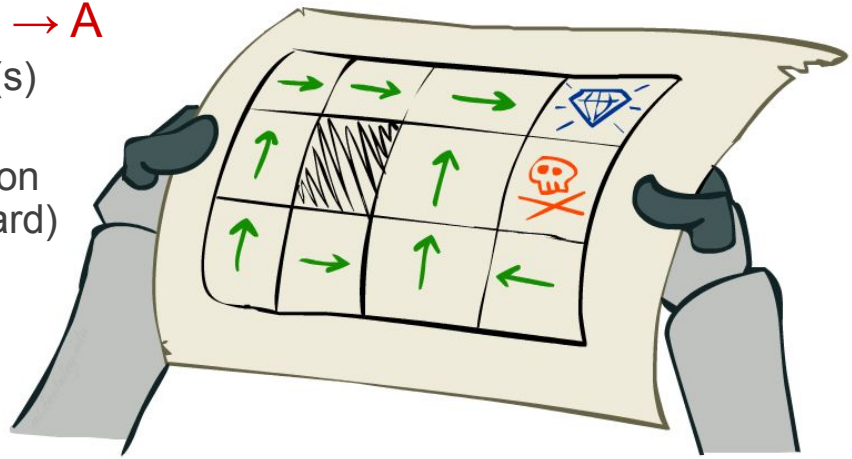
Andrey Markov
(1856-1922)

Slide credit: Daniel Brown

Policies

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π 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

Slide credit: Daniel Brown

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"

Slide credit: Daniel Brown

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

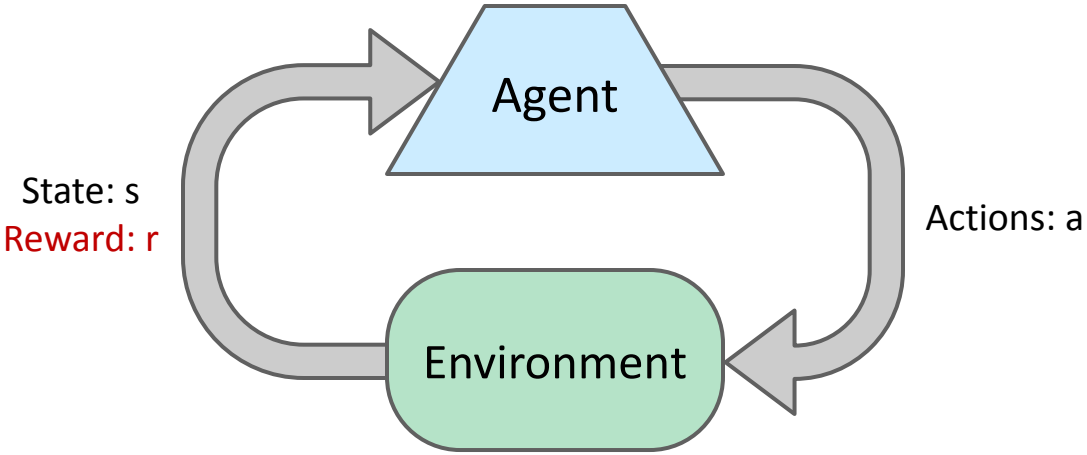
Slide credit: Daniel Brown

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

Slide credit: Daniel Brown

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!

Slide credit: Daniel Brown

Goal: Maximize Return/Utility

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

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

- 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 $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$
- Often we care about *discounted Return
  - Idea, rewa
  - Want to g
  - Discount
- Utility (sometir

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

R_{t+k+1}

y

$\left[\sum_{k=0}$

Example: Learning to Walk

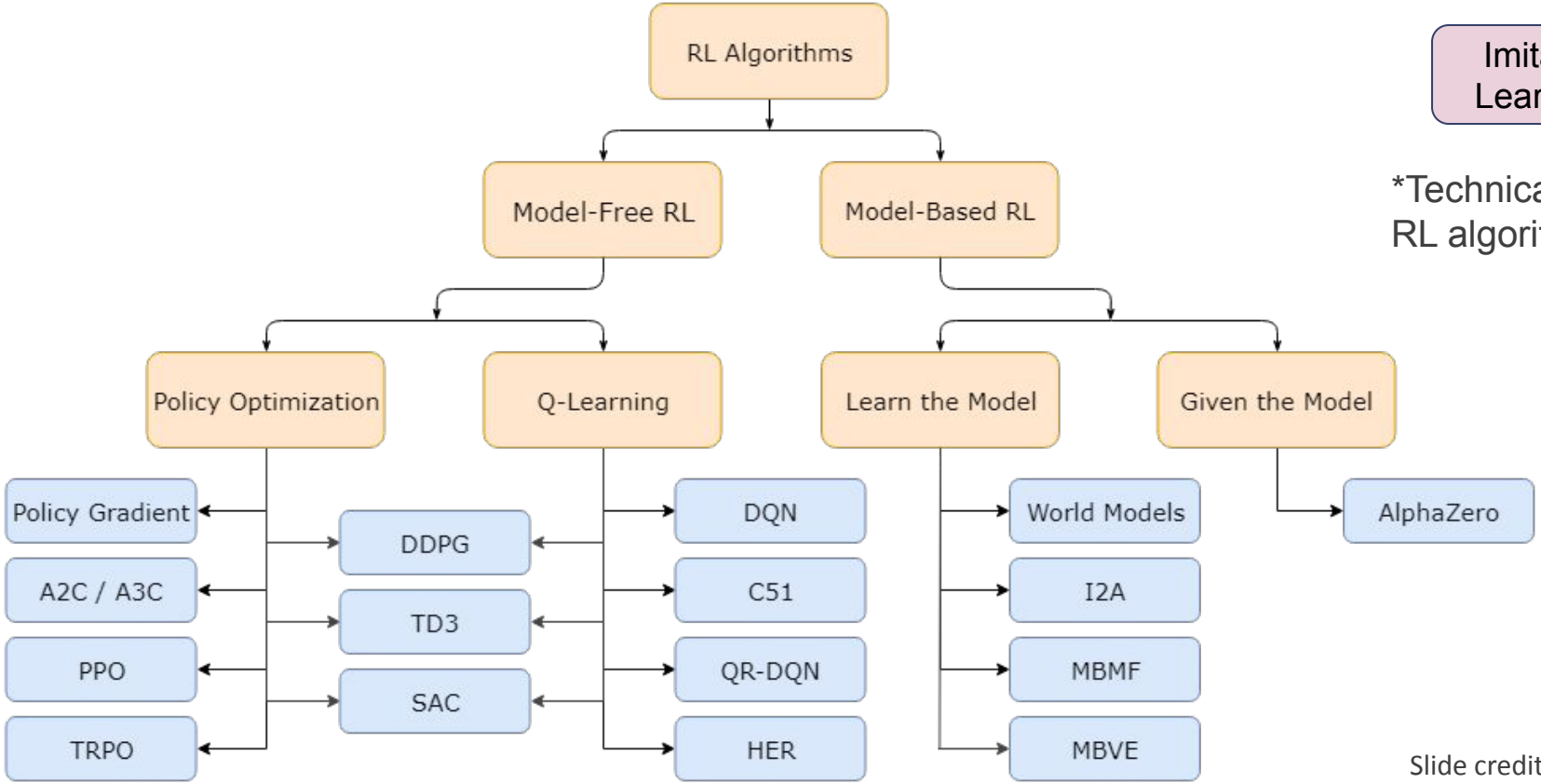
Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Rough Taxonomy of RL Algorithms

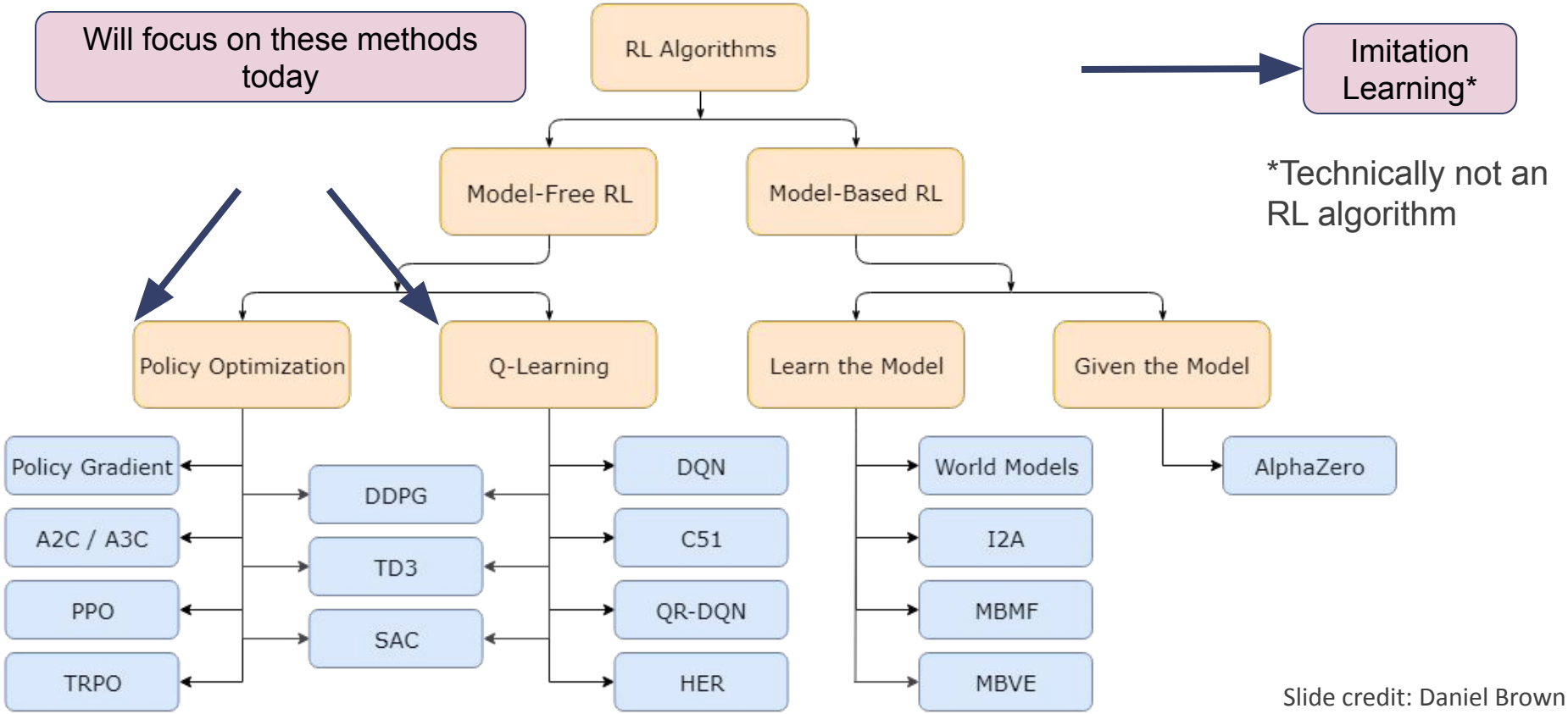


Imitation Learning*

*Technically not an RL algorithm

Slide credit: Daniel Brown

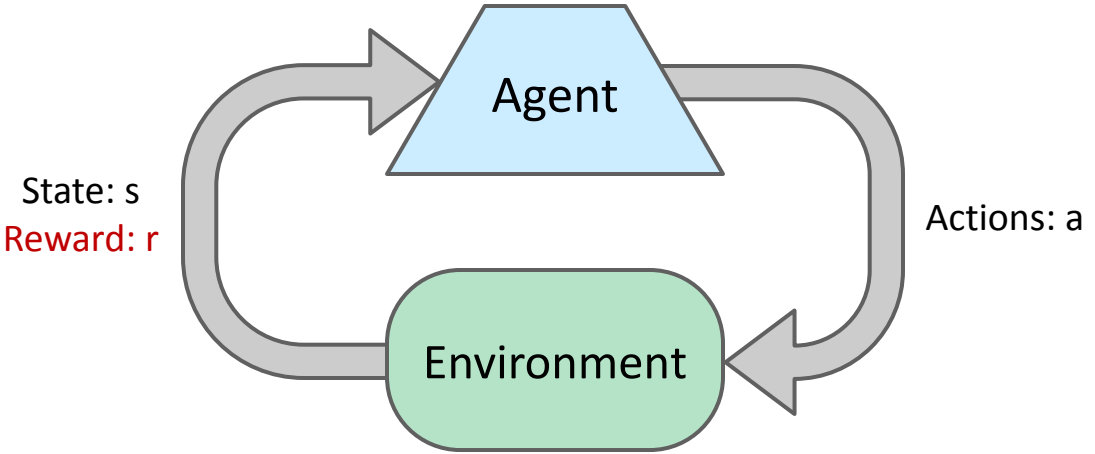
Rough Taxonomy of RL Algorithms



Slide credit: Daniel Brown

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 $\pi(s)$



Slide credit: Daniel Brown

Option 1: Imitation Learning

Common algorithm: Behavioral Cloning

Slide credit: Chelsea Finn

Goal of Imitation Learning

Data: Given trajectories collected by an expert

"*demonstrations*" $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T)\}$

(sampled from some unknown policy π_{expert})

Goal: Learn a policy π_θ that performs at the level of the expert policy, by mimicking it.

Example

- Dataset from human drivers
- Sensor readings + steering commands

Slide credit: Chelsea Finn

Imitation Learning - v1

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

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

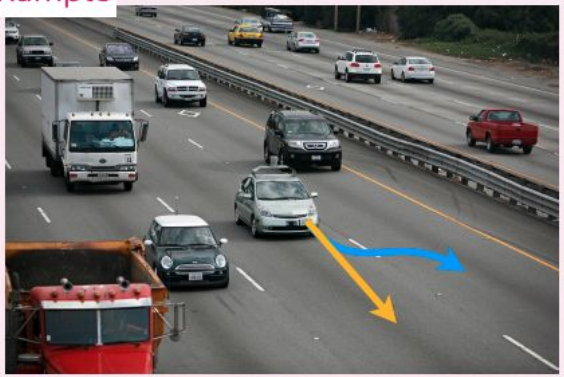
$$\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s},\mathbf{a})\in\mathcal{D}} \|\mathbf{a} - \hat{\mathbf{a}}\|^2 \qquad \text{where } \hat{\mathbf{a}} = \pi_\theta(\mathbf{s})$$

2. Deploy learned policy π_θ

Slide credit: Chelsea Finn

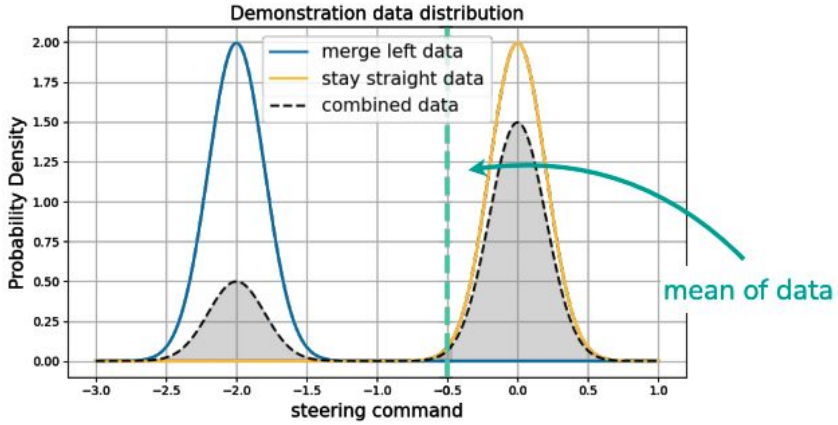
What can go wrong?

Example



- Dataset from human drivers
- Sensor readings + steering commands

Question: what might policy trained with ℓ_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.

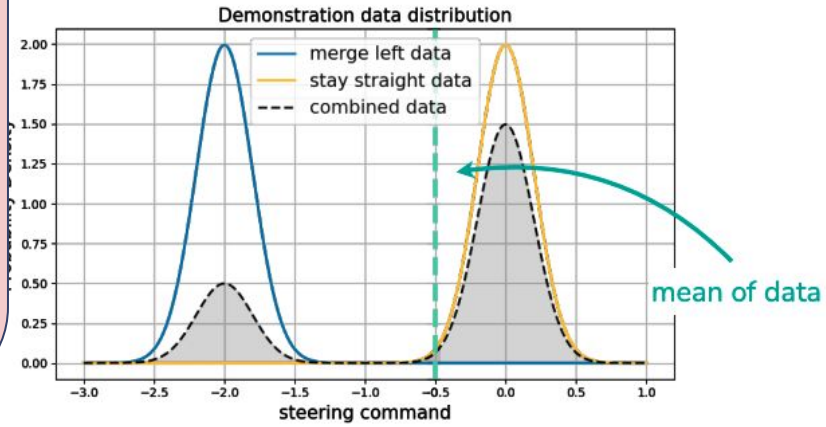
Slide credit: Chelsea Finn

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

n: what might policy trained with ℓ_2-regression do?



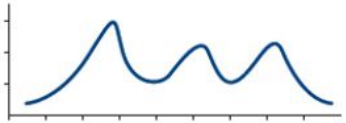
How can we represent more than the mean?

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

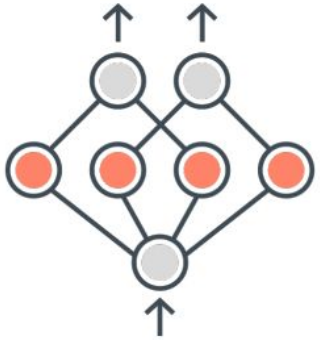
Esp. when data collected by multiple people.

Slide credit: Chelsea Finn

Learning Distributions with NNs



parameters of distribution

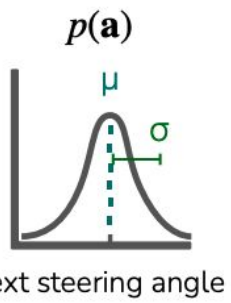


1D discrete actions

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

Maximally expressive

Continuous actions



Neural net outputs μ, σ to represent Gaussian distribution.

Not very expressive!

Slide credit: Chelsea Finn

Imitation Learning v2

Expressive policies

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

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

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

maximize the log probability of the demo actions under the policy

2. Deploy learned policy π_θ

Slide credit: Chelsea Finn

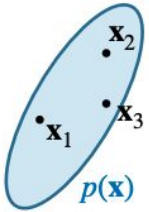
How this connects to agents

- 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?

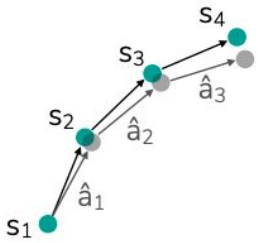
Compounding errors

Supervised learning



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

Supervised learning of behavior



Predicted actions affect next state.

Errors can lead to drift away from the data distribution!

Errors can then compound!

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

states visited by expert

states visited by learned policy π

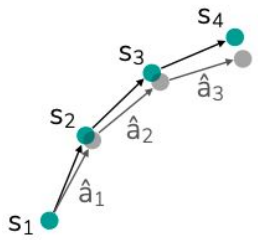
"covariate shift"

Slide credit: Chelsea Finn

What can still go wrong?

Compounding errors

Supervised learning of behavior



Predicted actions affect next state.

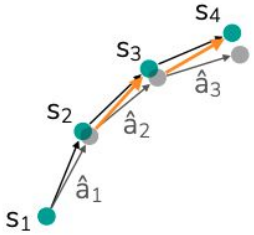
Errors can lead to drift away from the data distribution!

Errors can then compound!

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

Solutions?

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



Slide credit: Chelsea Finn

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

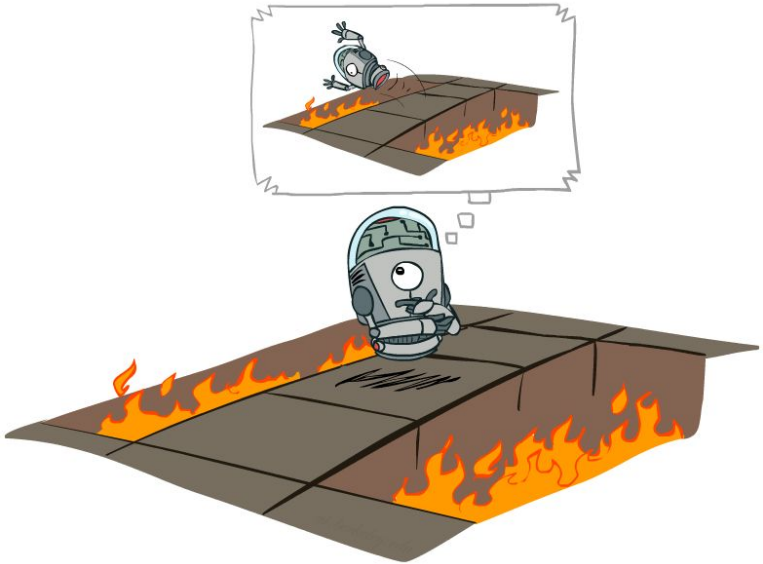
Reinforcement Learning

- 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 $\pi(s)$

Slide credit: Daniel Brown

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



Value = Q(s, a)

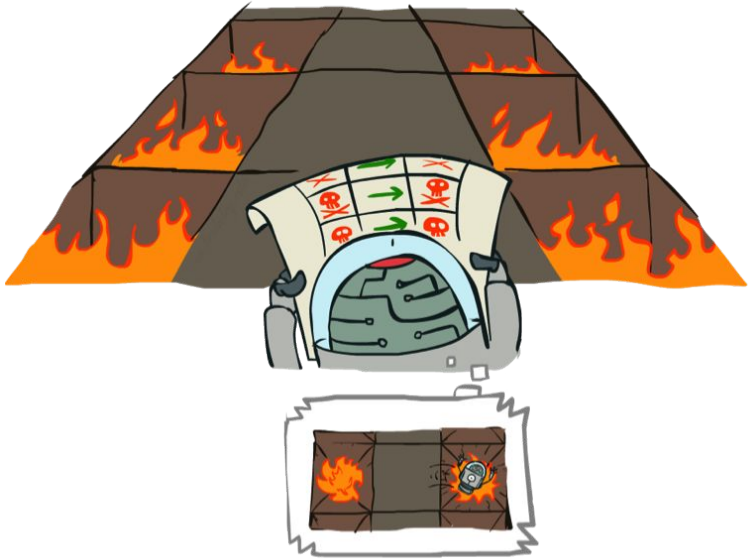
How much return we think we'll get

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



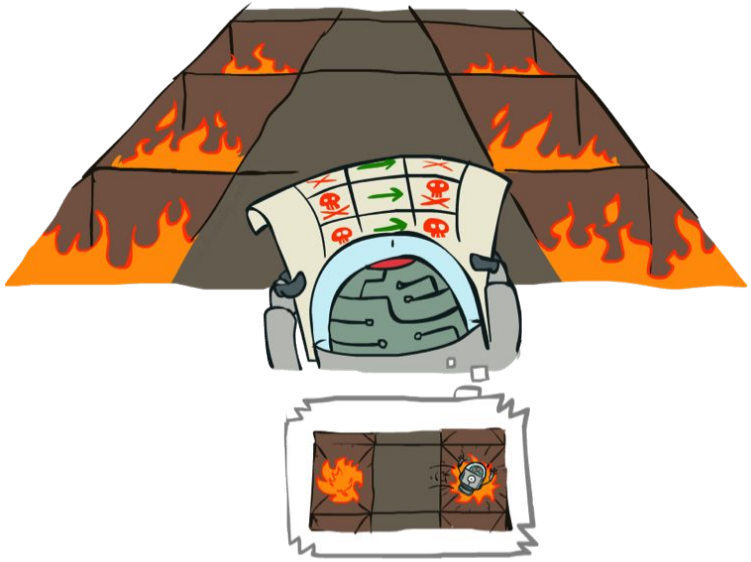
$$\text{Value} = Q(s, a)$$

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

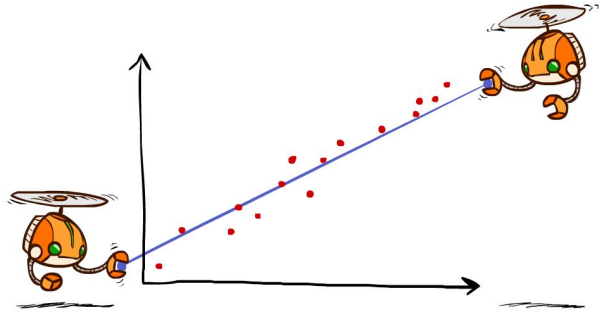
Update Q with gradient descent

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

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



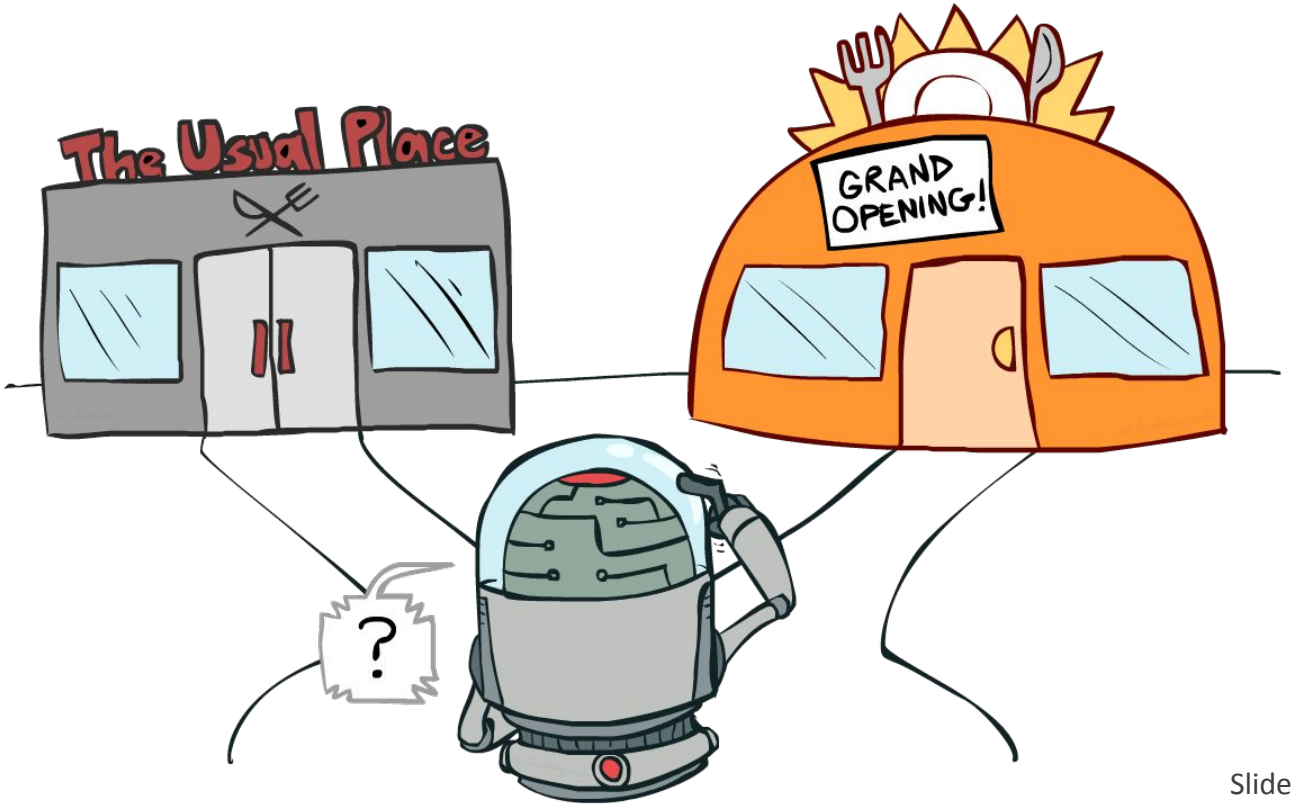
Approximate q update explained:

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

"target" "prediction"

Slide credit: Daniel Brown

Exploration vs. Exploitation



Slide credit: Daniel Brown

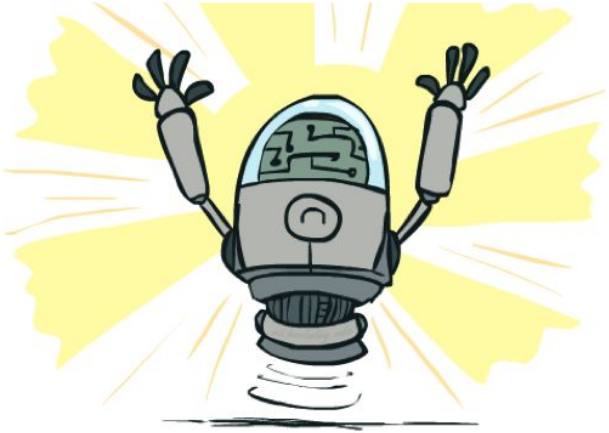
How to Explore

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

Slide credit: Daniel Brown

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 (!)



Slide credit: Daniel Brown

Problems with Q Learning

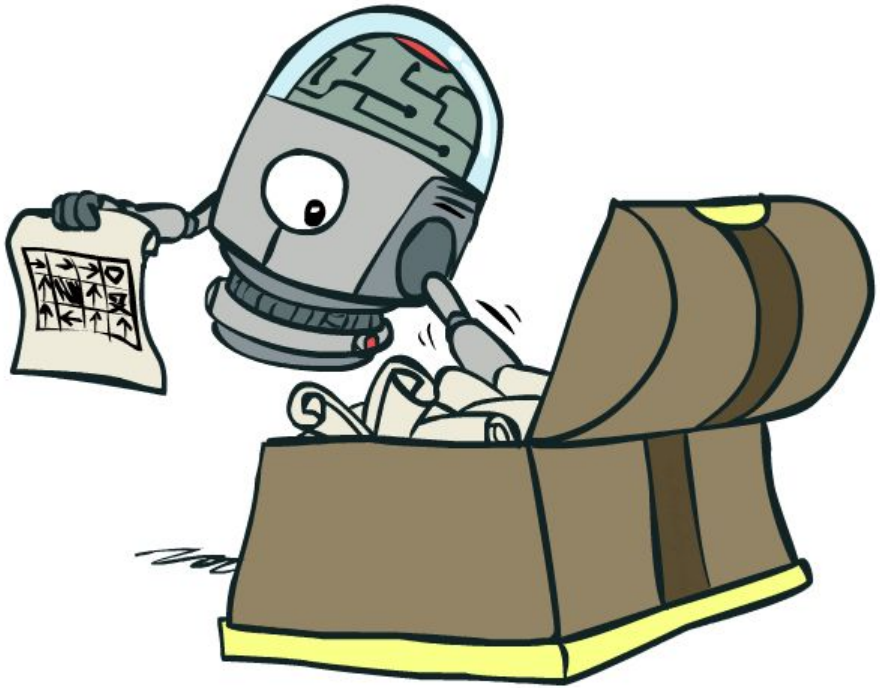
- Only works for discrete action space
  - Useless for continuous actions like in e.g. robotics
  - Can also be tricky for very large action spaces (like in many agent problems)
- Off-policy
  - Great when we have a lot of off-policy data to use (e.g. trajectories collected from people)
  - Need to do 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 π_θ
  - Update weights of policy network to directly optimize policy

Slide credit: Daniel Brown

Policy Search



Slide credit: Daniel Brown

Preliminaries

- Trajectory (rollout, episode) $\tau = (s_0, a_0, s_1, a_1, \dots)$
  - $s_0 \sim \rho_0(\cdot), \quad s_{t+1} \sim P(\cdot\,|s_t, a_t)$
- 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 π_θ

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

Slide credit: Daniel Brown

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 $\mathrm{J}(\pi)$

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

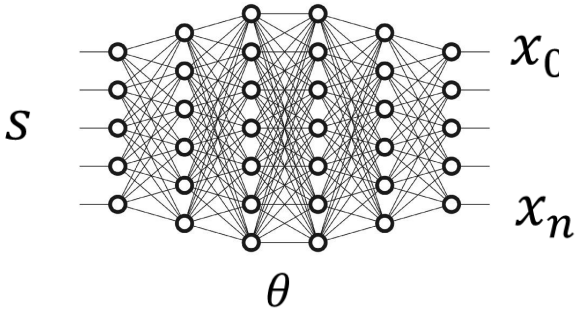
- Goal of RL: Solve the following optimization problem

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

Slide credit: Daniel Brown

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



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

Slide credit: Daniel Brown

How should we parameterize our policy?

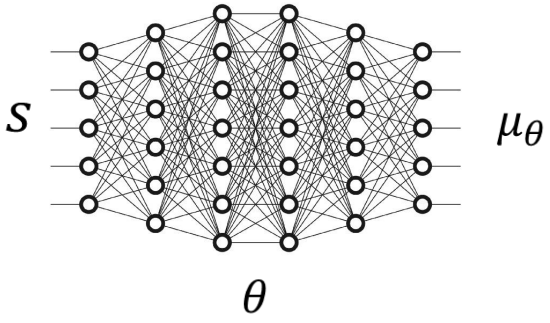
- Diagonal Gaussian (distribution over continuous actions)

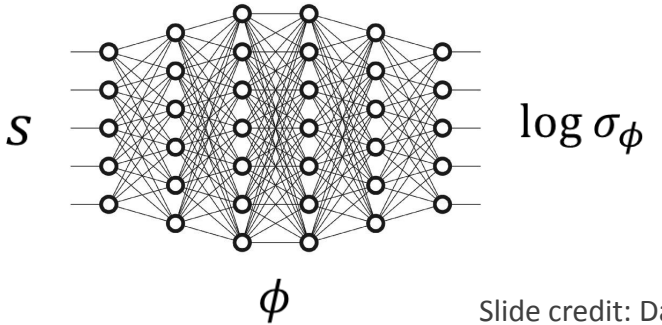
$$a \sim N(\mu, \Sigma)$$

where Σ has non-zero elements only on the diagonal.

Thus, an action can be sampled as

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





Slide credit: Daniel Brown

Goal: Update Policy via Gradient Ascent

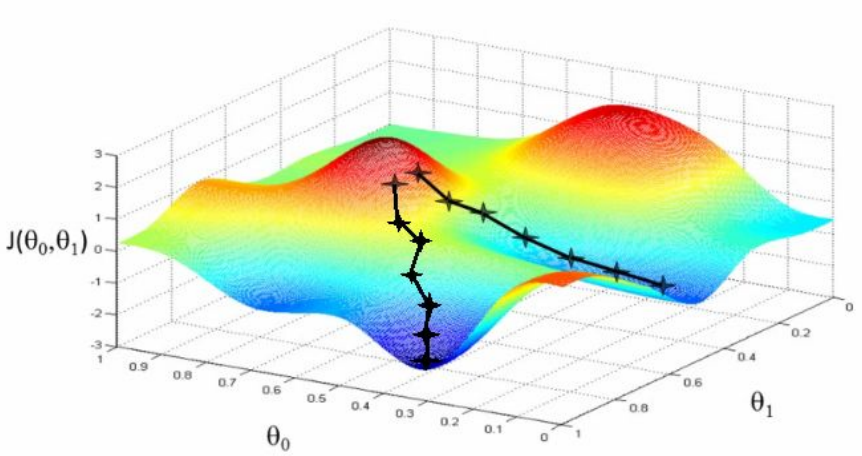
- We have a parameterized policy and we want to update it so that it maximizes the expected return.
- We want to find the gradient of the return with respect to the policy parameters and step in that direction.

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

Policy gradient

Slide credit: Daniel Brown

Optimization: Max a function by “climbing” a hill



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

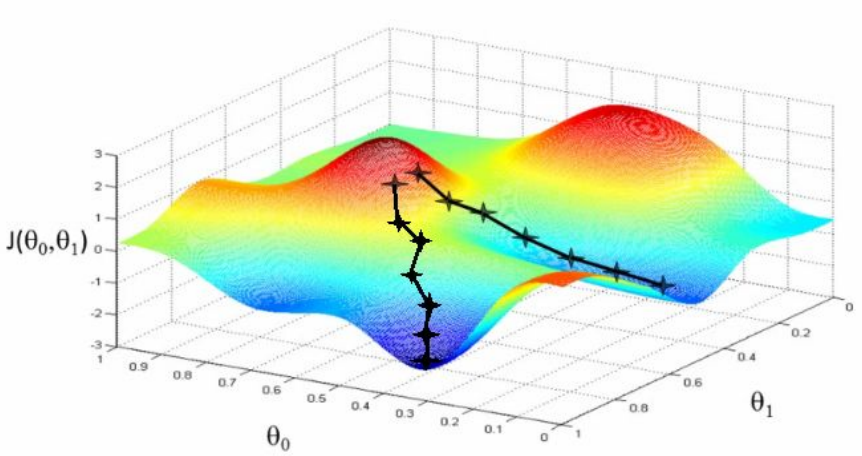
Policy gradient

Image via: https://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/

Slide credit: Daniel Brown

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

Image via: https://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/

Slide credit: Daniel Brown

Derivation of Policy Gradient

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

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

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

Slide credit: Daniel Brown

How would you implement this?

1. Start with random policy parameters θ_0
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

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

$$D = \{\tau_1, \ldots \tau_N\}, \qquad R = \{R(\tau_1), \ldots R(\tau_N)\}$$

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

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

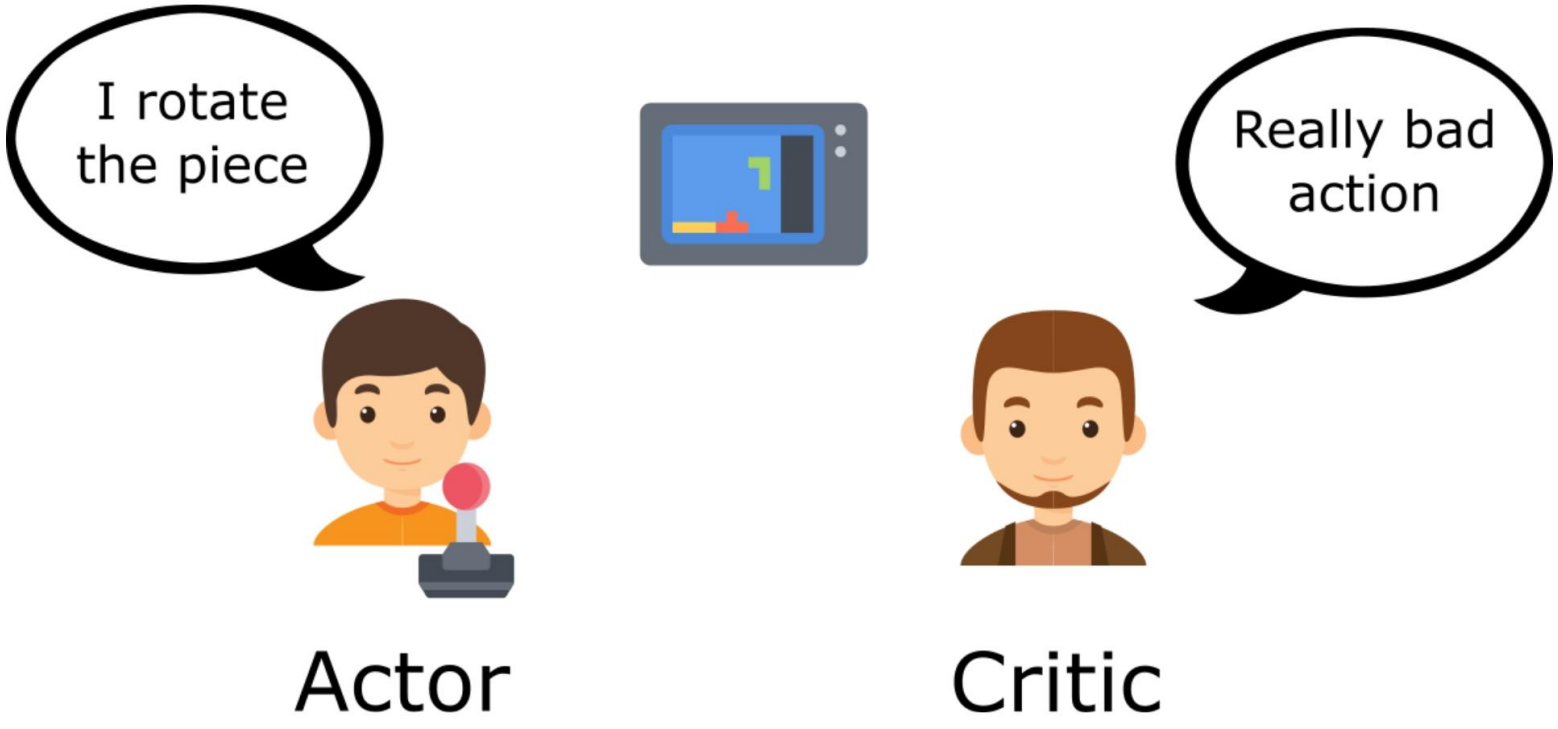
Slide credit: Daniel Brown

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.

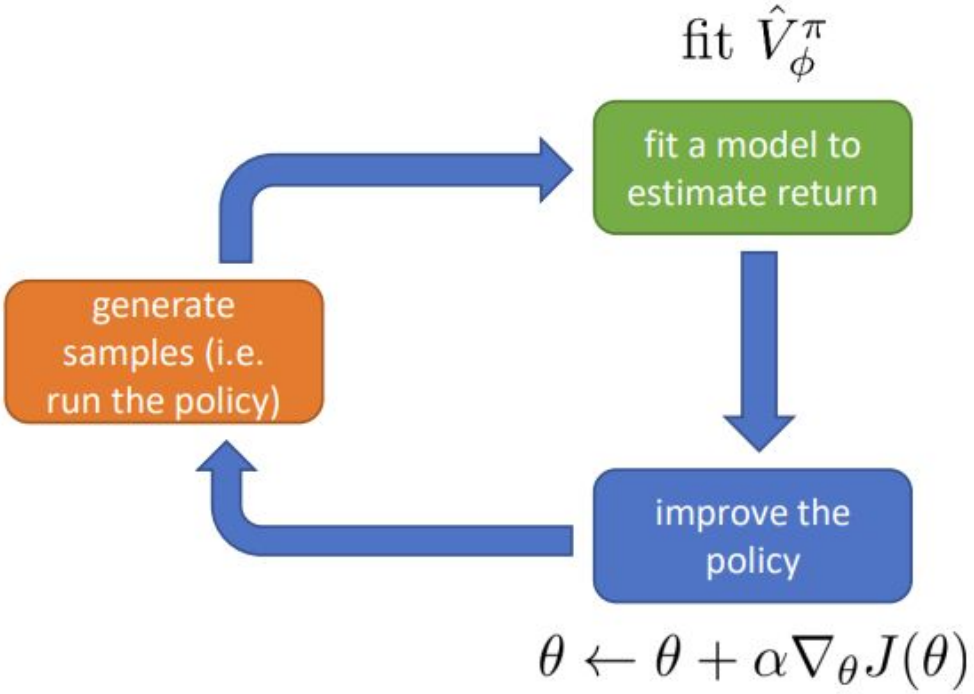
Slide credit: Daniel Brown

Actor Critic



Slide credit: Daniel Brown

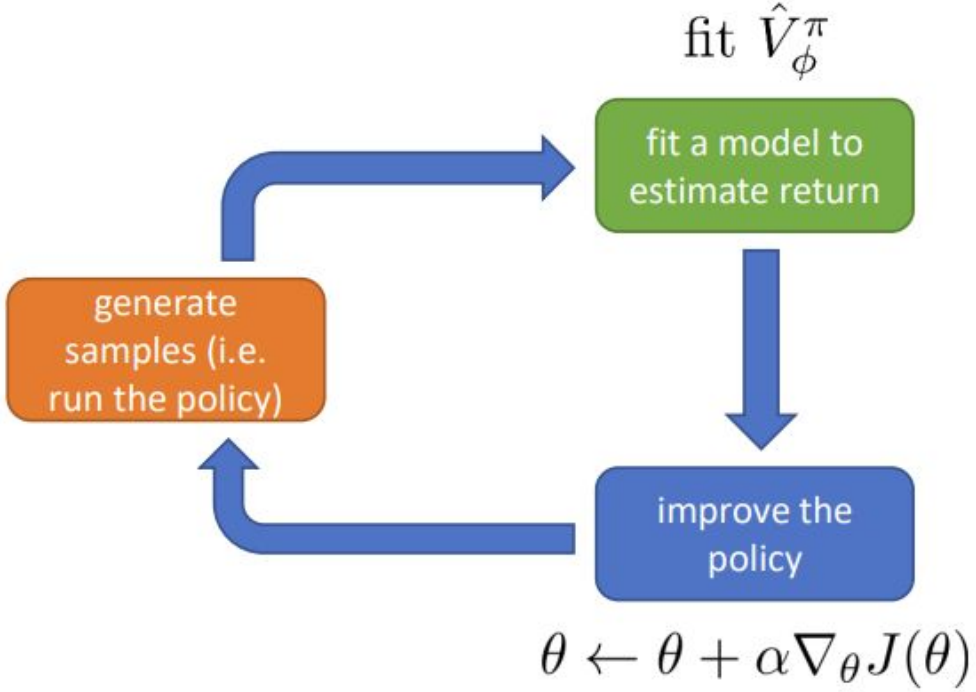
Actor Critic



Slide credit: Daniel Brown

Actor Critic

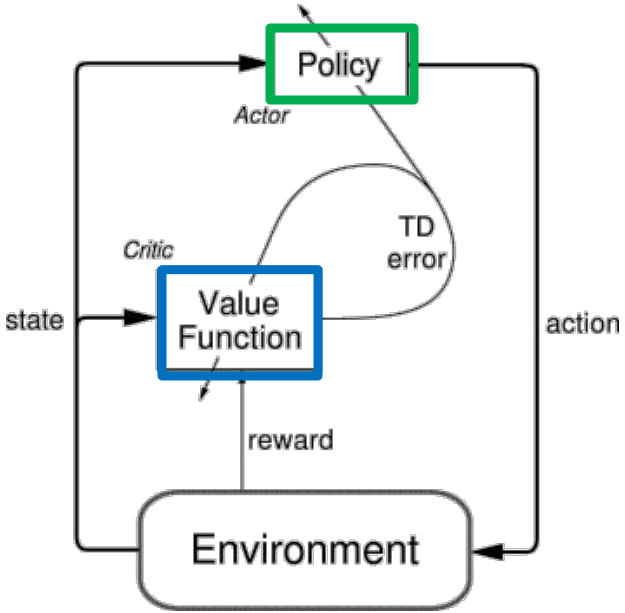
Combination of the two ideas of RL we've seen so far



Slide credit: Daniel Brown

Actor Critic Algorithms

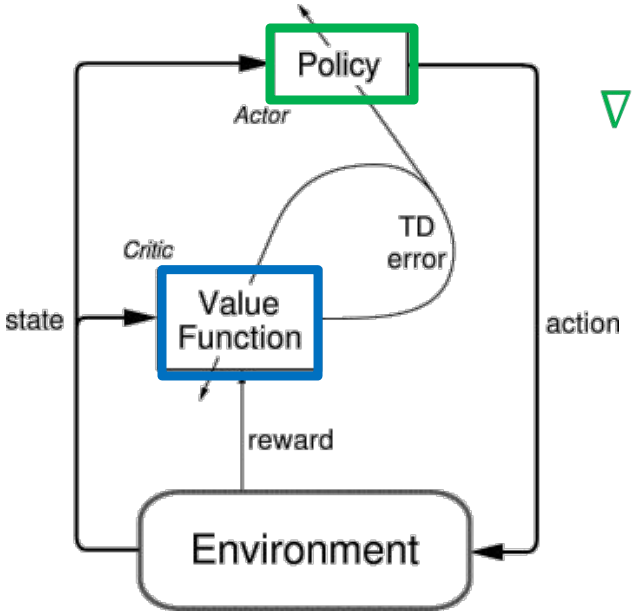
- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



Slide credit: Daniel Brown

Actor Critic Algorithms

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

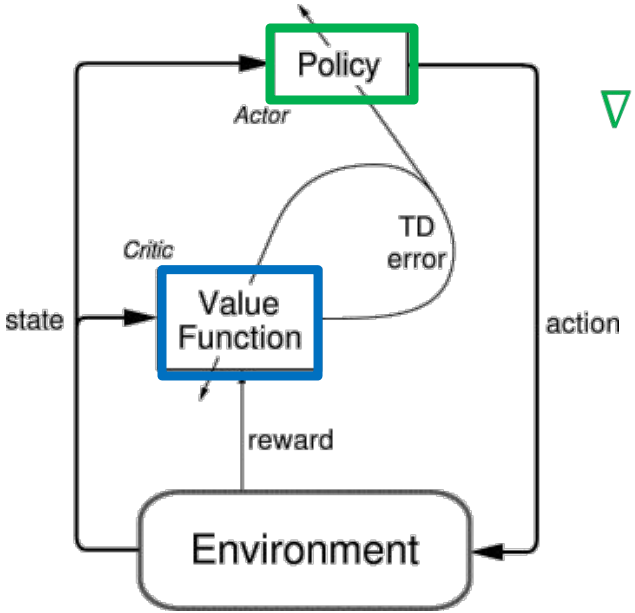


$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_w^{\pi_\theta}(s_t, a_t)\right]$$

Slide credit: Daniel Brown

Actor Critic Algorithms

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



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

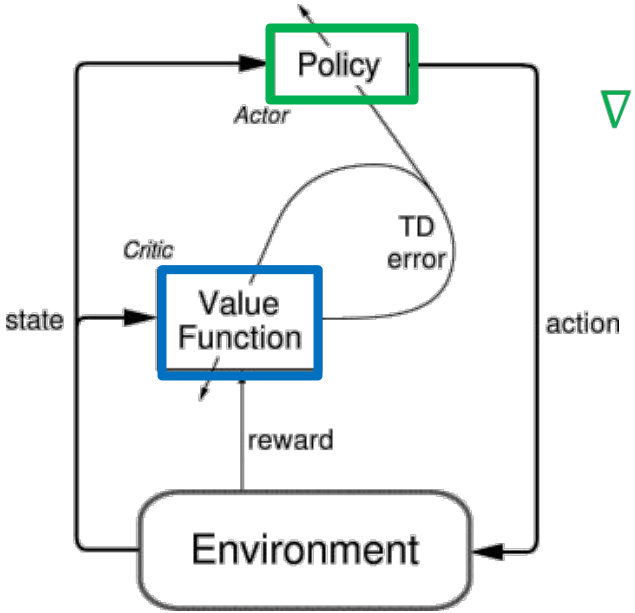
Recall, before we directly used reward here

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

Slide credit: Daniel Brown

Actor Critic Algorithms

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



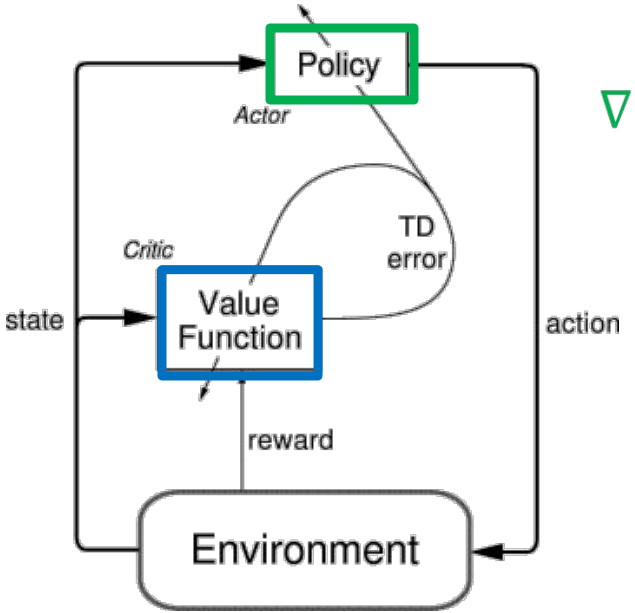
$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_w^{\pi_\theta}(s_t, a_t)\right]$$

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

Slide credit: Daniel Brown

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, θ, w and learning rates α_θ, α_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"

Slide credit: Daniel Brown

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

Slide credit: Daniel Brown

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.

https://huggingface.co/blog/deep-rl-ppo

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **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_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

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

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**

Slide credit: Daniel Brown

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **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_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

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

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/deep-rl-ppo for full details

Slide credit: Daniel Brown

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/

Slide credit: Daniel Brown

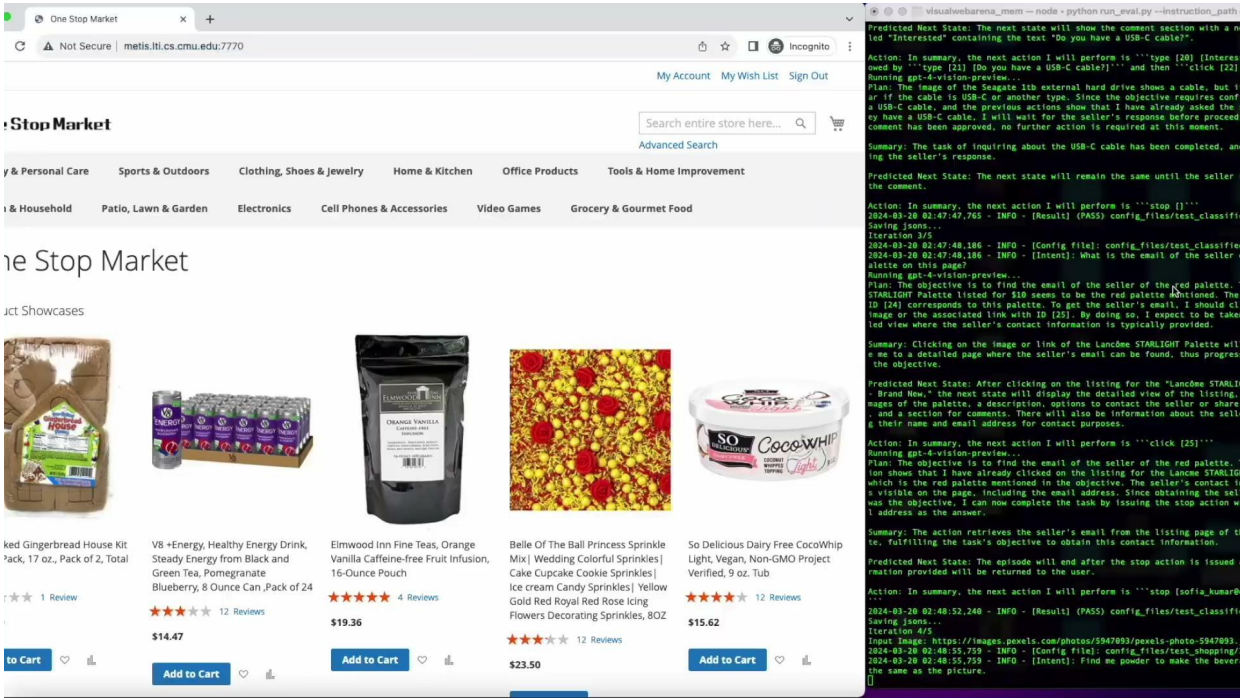
Review: How to learn the policy

- 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

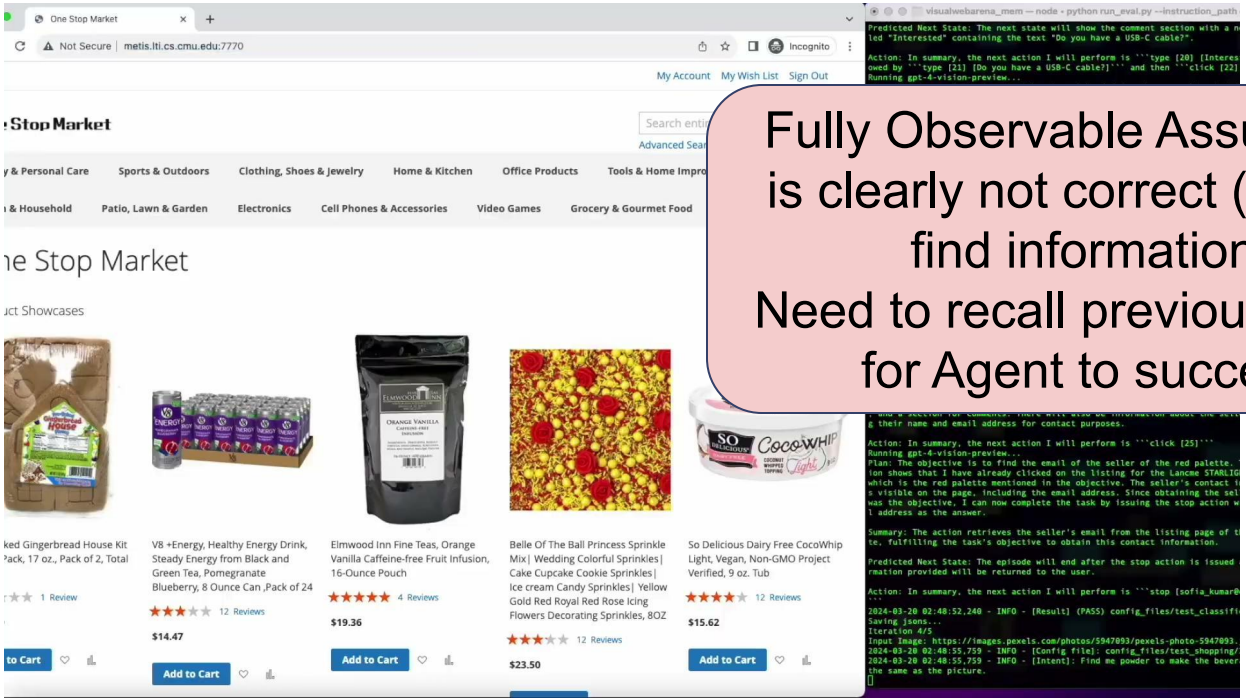
Assumptions we've made

- 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

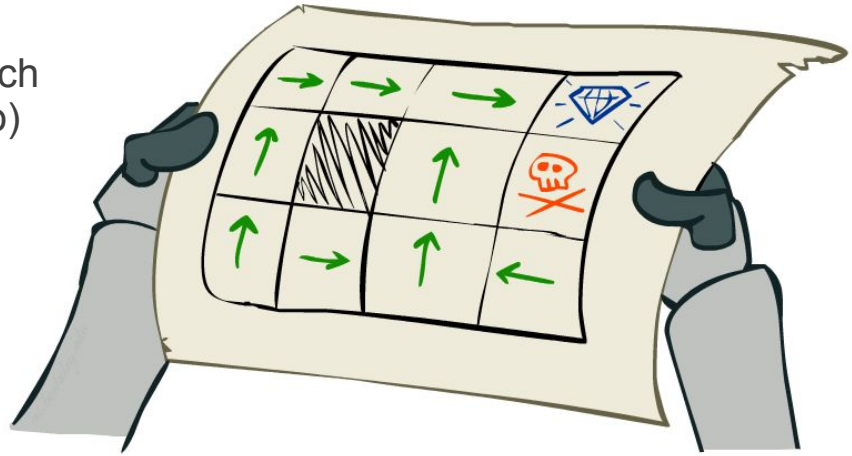


Example: Computer Use



LLM Policies In Practice

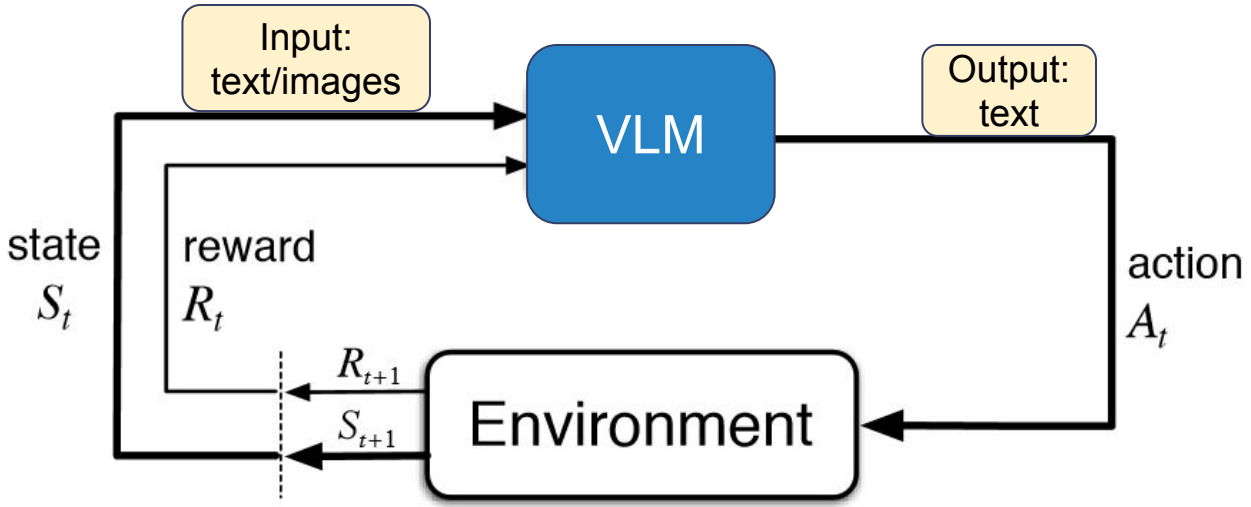
- $\pi(o_t, \text{history}(o_{0:t-1}, a_{0:t-1}), i)$
  - VLM policy takes the current observation o_t (which doesn't necessarily contain all relevant state info)
  - Some (usually condensed) history of past observations and actions the agent took
  - Possibly other information (may depend on your problem)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



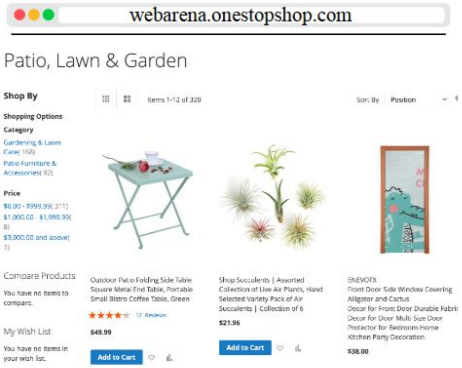
Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

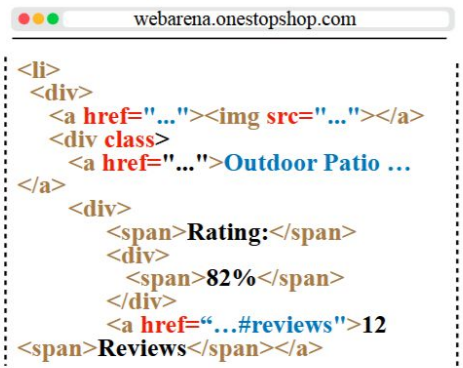
What this means for VLM Agents

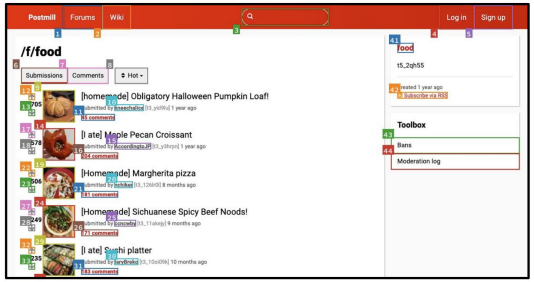
- Basics of environment, maximizing reward, policies, all apply
  - State/observation given by text image tokens
  - Output is text (then interpreted as environment actions)
  - Want to maximize some reward (answer a query, fill out a web form, do some action in the environment)



Recall: Obs/Action Space of CU Agent







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 a	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

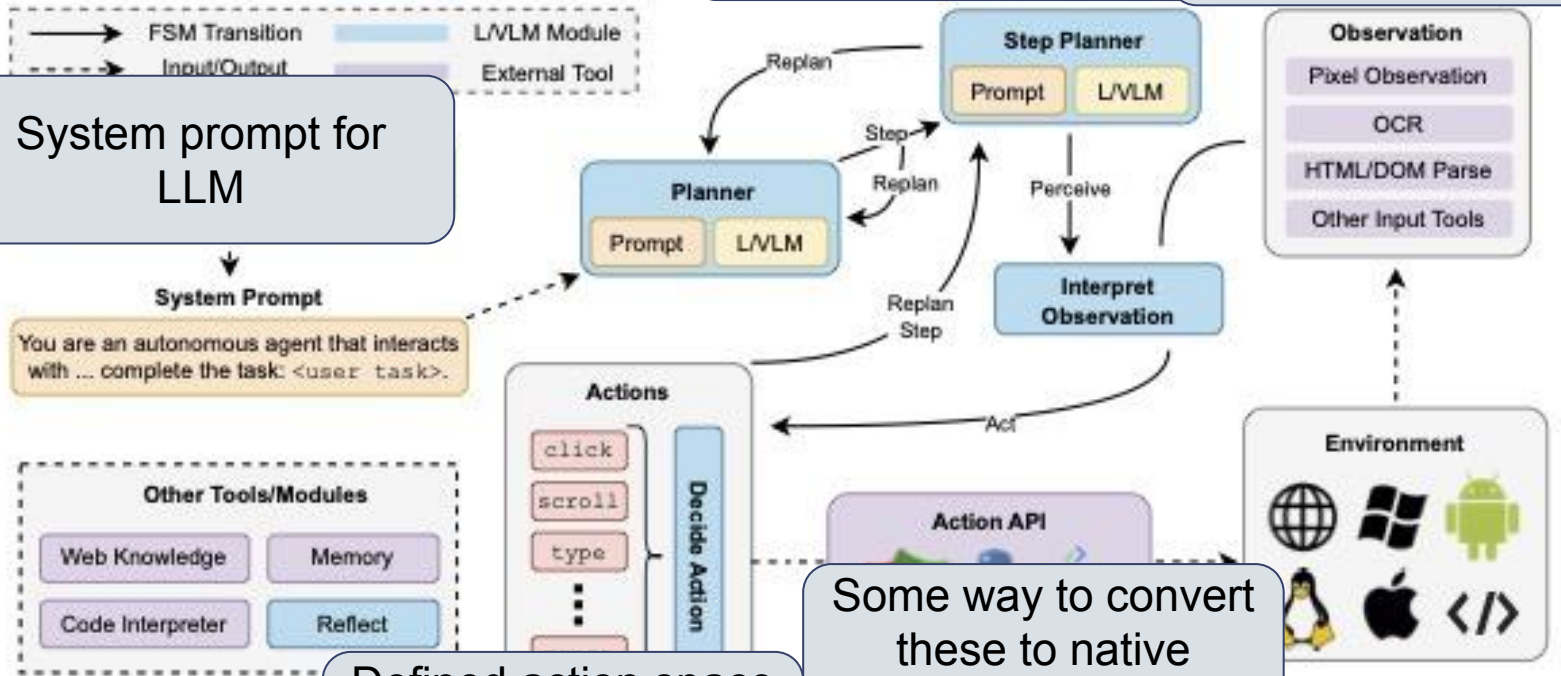
What this means for VLM Agents

- 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

Some way to convert these to native actions in env

Tools VLM can i that do different t

Defined action space that can be expressed in text

Image credit: Farhan Ishmam

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