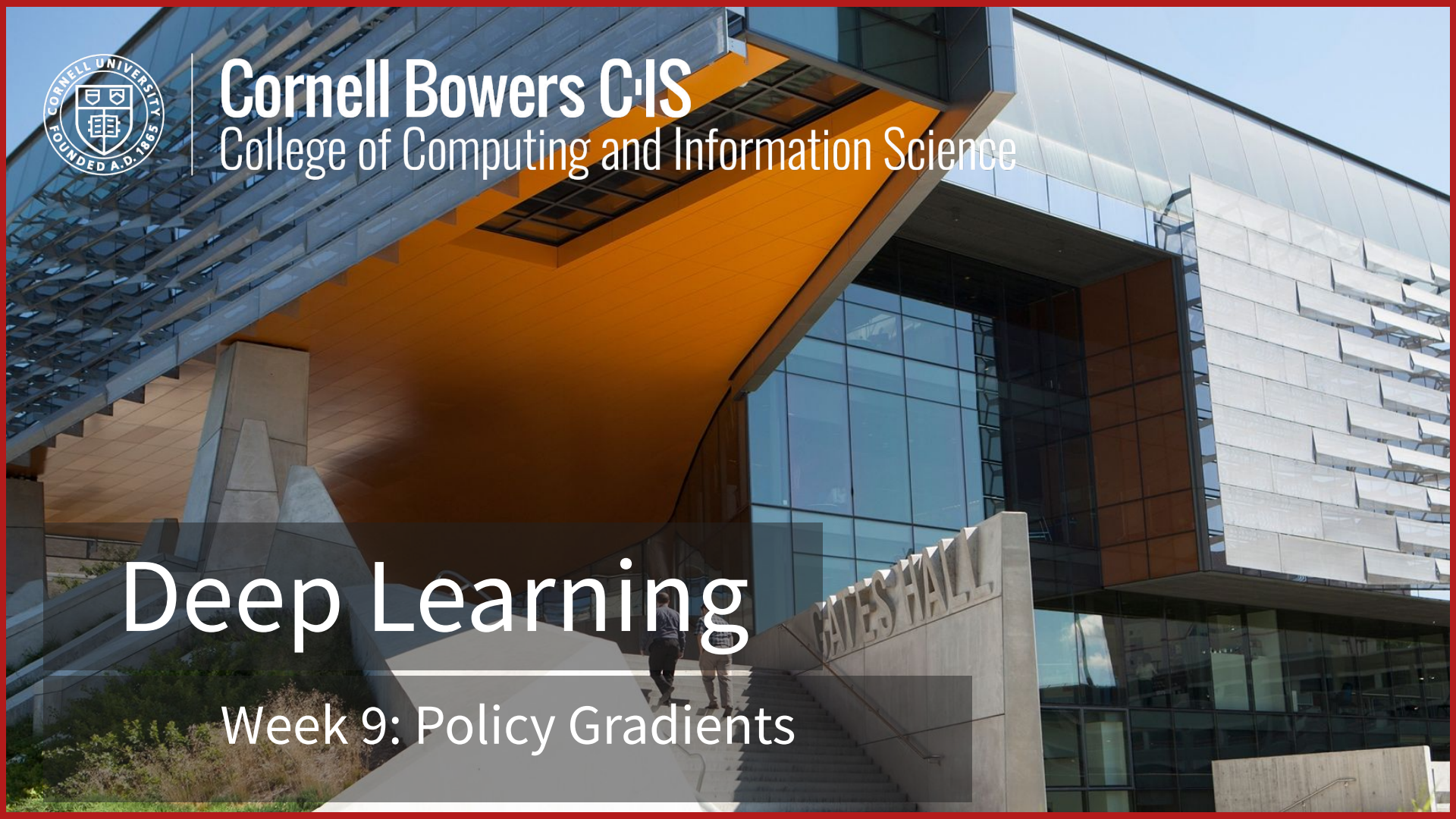


Deep Learning

Week 9: Policy Gradients

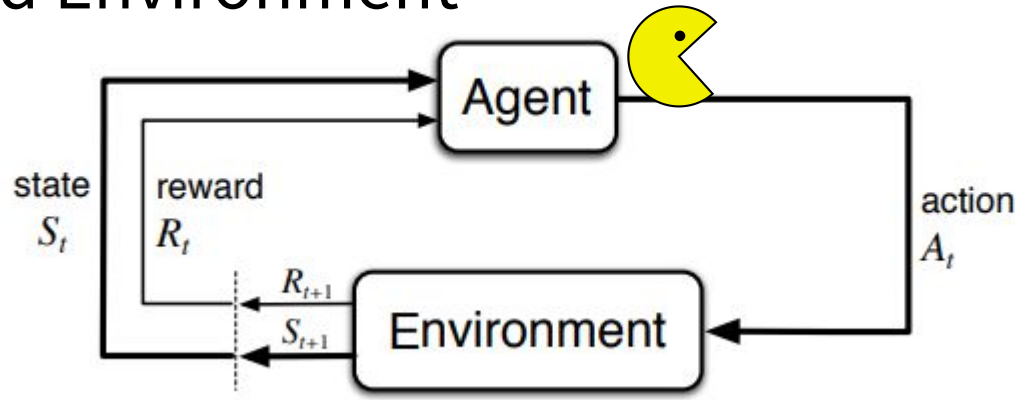
Thanks to

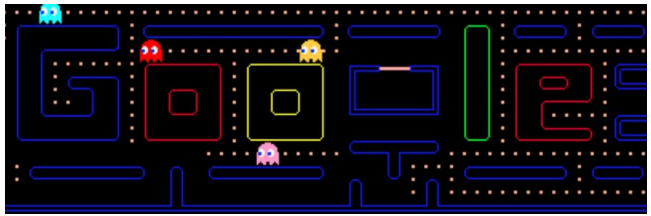
Varsha Kishore
Justin Lovelace
Luke Kulm
Jinzhou Li

Logistics

- Midterm Scores will be out soon … (hopefully tomorrow Apr 16th!)
  - If you have regrade requests, submit them before April 29th
- **Quiz 5** - Paper will be released this week
- Notes sign-up - has everyone who wanted to been able to sign-up?

Agent and Environment





Agent:	Environment:
- Perceives environment. - Makes decisions. - Aims to maximize reward.	- The external context in which an agent operates and interacts with - Provides feedback to agent

Markov Decision Process (MDP)

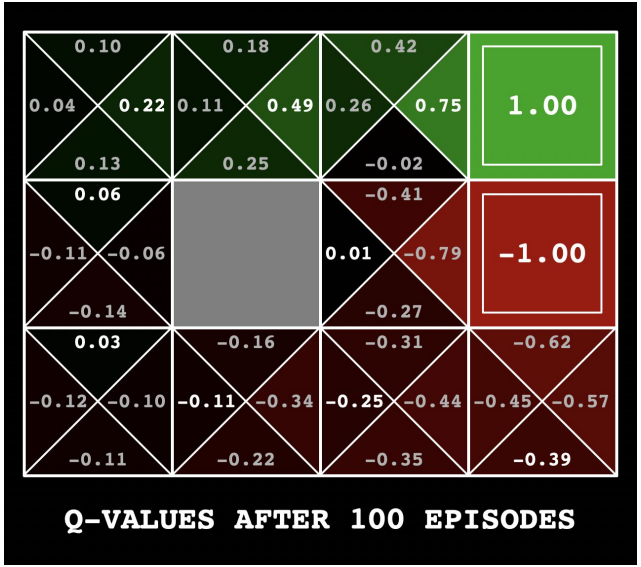
- MDPs provide a framework for modeling sequential decision-making problems.
- An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:
  - $\mathcal{S}$: Set of states representing the environment.
  - $\mathcal{A}$: Set of actions the agent can take.
  - $\mathcal{P}$: Transition probability function, $\mathcal{P}(s'|s, a)$.
  - $\mathcal{R}$: Reward function, $\mathcal{R}(s_t, a_t, s'_t)$
  - γ: Discount factor, $\gamma \in [0, 1]$.

Q-Learning Update Rule

- The Q-Learning update rule can be expanded as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\mathcal{R}(s_t, a_t, s'_t) + \gamma \max_{a'} Q(s', a') - Q(s,a)\right]$$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[\mathcal{R}(s_t, a_t, s'_t) + \gamma \max_{a'} Q(s', a')\right]$$



- The update rule adjusts the current Q-value estimate $Q(s,a)$ in the direction of the target Q-value based on the Bellman error.
  - Q-values are gradually improved and converge towards the optimal Q-function.

$$Q^*(s,a) = \mathcal{R}(s_t, a_t, s'_t) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s', a')$$

Goal of Reinforcement Learning

Find the optimal policy π_θ^* that maximizes the expected discounted return $J(\theta)$:

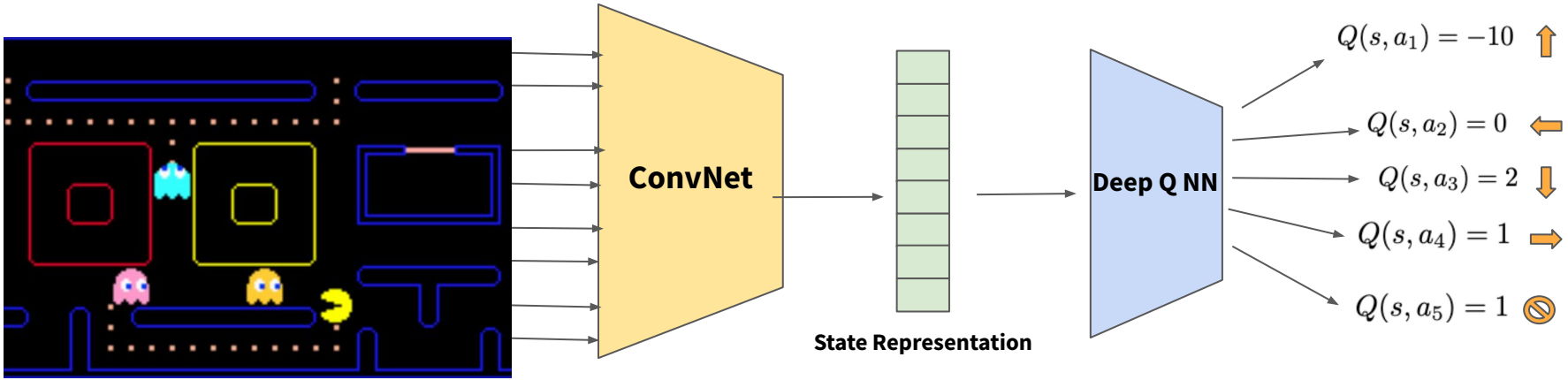
$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \, \mathcal{R}(s_t, a_t, s_{t+1}) \right]$$

$$\pi_\theta^* = \arg\max_\theta J(\theta)$$

where:

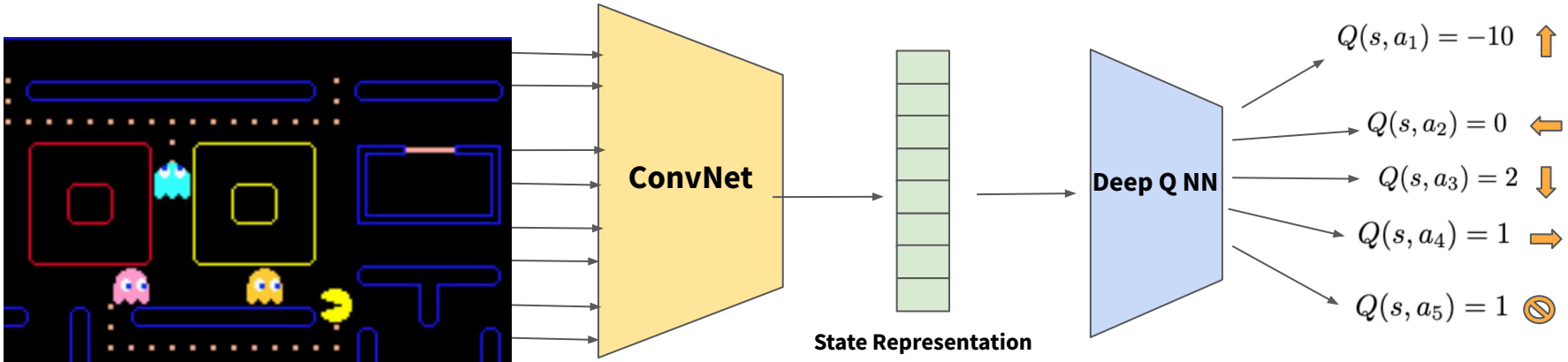
- $J(\theta)$ is the expected discounted return under the policy π_θ.
- τ represents a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ sampled from the policy π_θ.
- $p_\theta(\tau)$ is the probability distribution over trajectories induced by the policy π_θ.

Deep Q-Learning:



$$\pi(s) = \underset{a}{argmax}(Q(s, a))$$

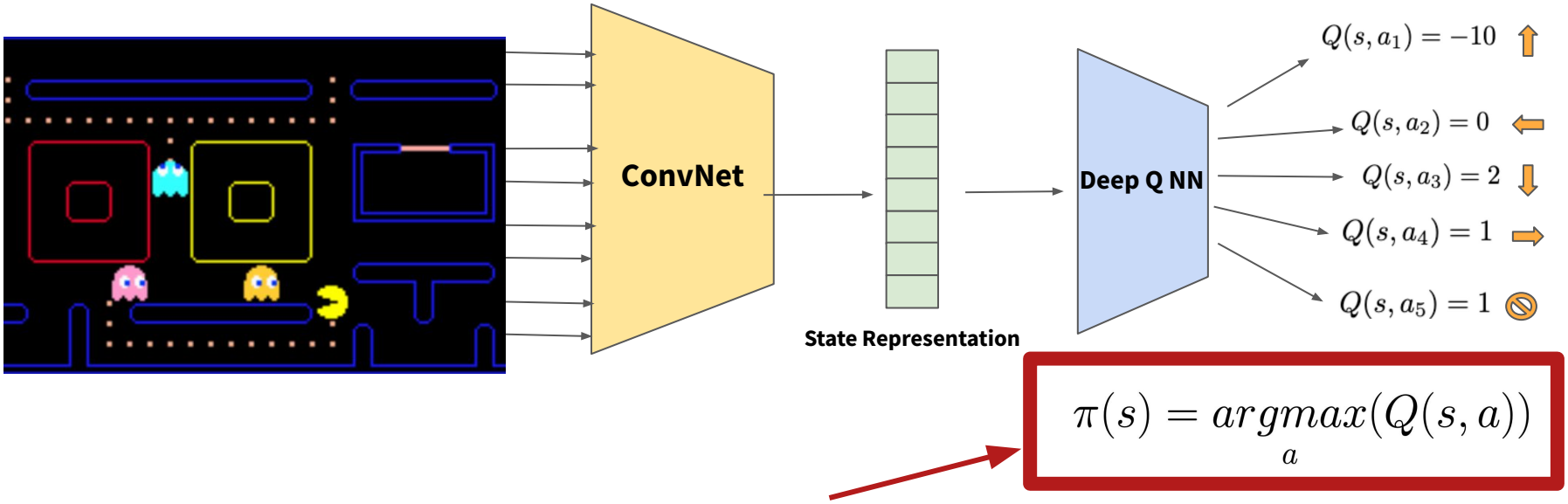
Deep Q-Learning:



$$\pi(s) = \underset{a}{argmax}(Q(s, a))$$

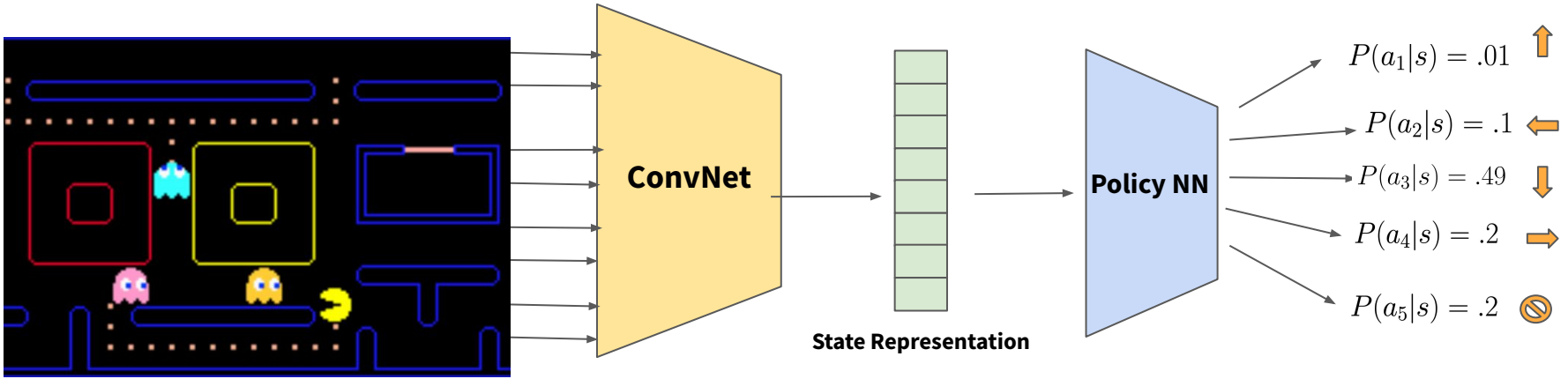
This max function makes continuous and stochastic actions hard

Deep Q-Learning:



What if we can learn the policy directly?

Policy Gradient:

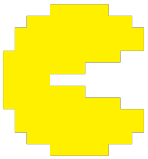


$$\pi(s) \sim P(a|s)$$

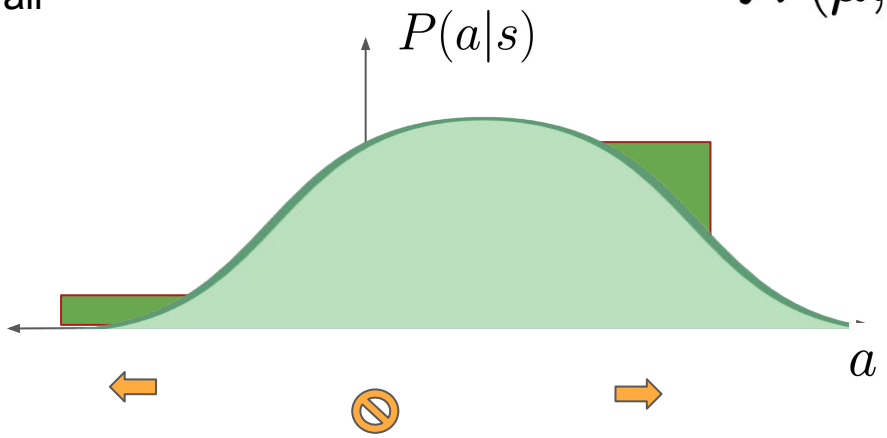
Policy Distribution:

Pacman is trying to decide to go left, right, or not move at all

Pacman is now trying to decide how fast to move?

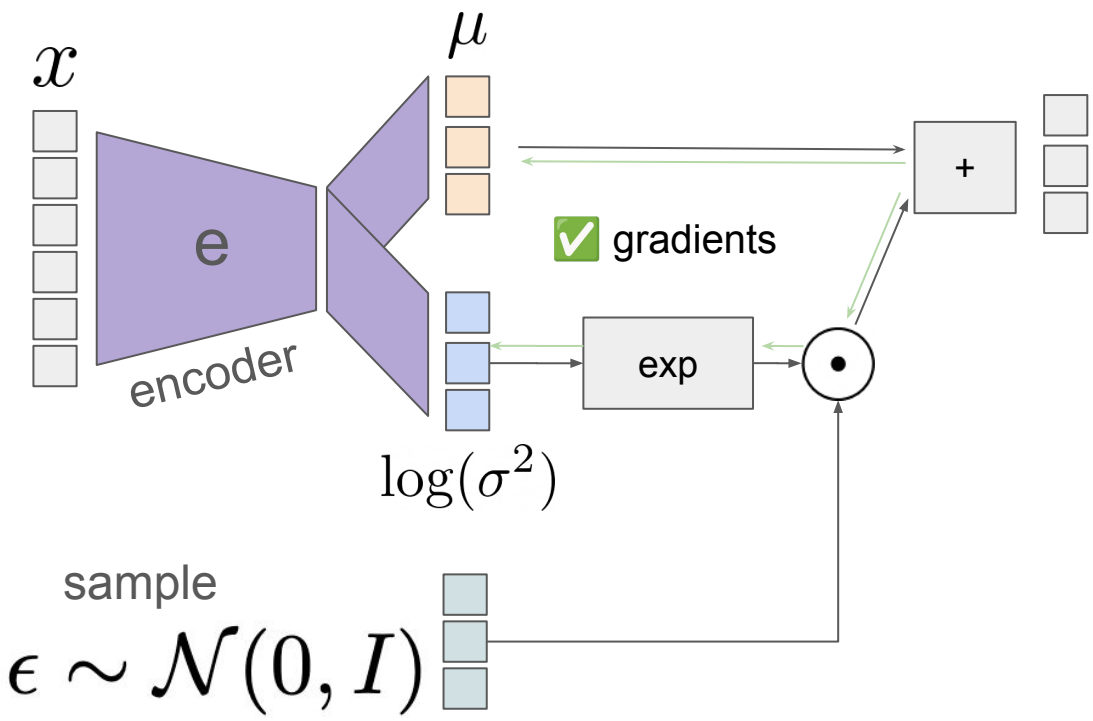


$$\mathcal{N}(\mu, \sigma^2)$$



$$\pi(s) \sim P(a|s)$$

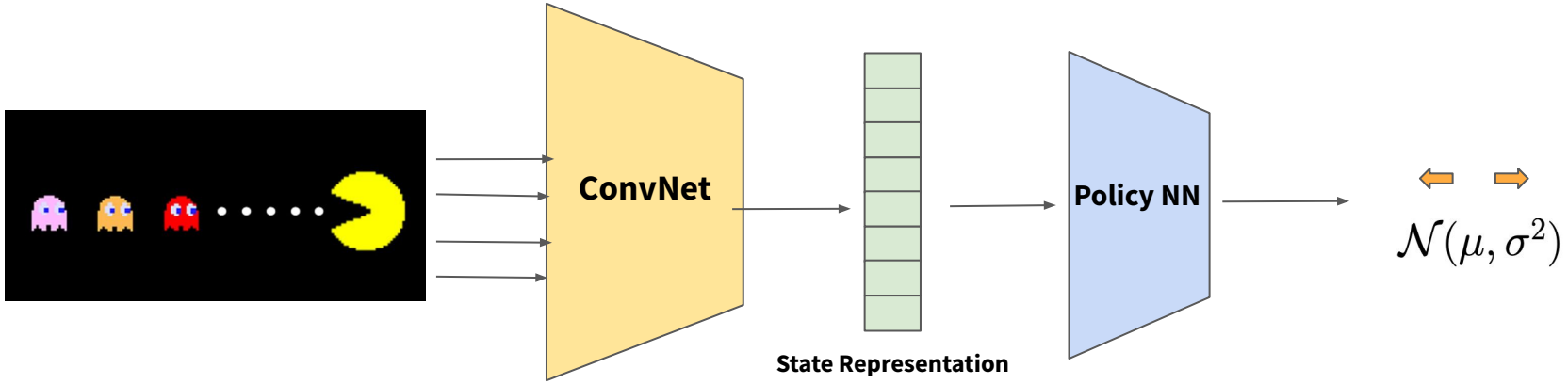
Review: The Reparameterization Trick



$$z = \mu + \sigma \odot \epsilon$$

$$z \sim \mathcal{N}(\mu, \sigma^2 I)$$

Continuous Policy Gradient:



$$\pi(s) \sim \mathcal{N}(\mu, \sigma^2)$$

Goal of Reinforcement Learning

Find the optimal policy π_θ^* that maximizes the expected discounted return $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \, \mathcal{R}(s_t, a_t, s_{t+1}) \right]$$

$$\pi_\theta^* = \arg\max_\theta J(\theta)$$

where:

- $J(\theta)$ is the expected discounted return under the policy π_θ.
- τ represents a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ sampled from the policy π_θ.
- $p_\theta(\tau)$ is the probability distribution over trajectories induced by the policy π_θ.

How do we maximize reward? Gradient **ascent**!!

Gradient Descent:

$$\min_{\theta} \mathcal{L}(\theta)$$

update:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta_k)$$

Policy Iteration:

$$\max_{\theta} J(\theta)$$

update:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

Policy Gradient Objective

To maximize the expected discounted return $J(\theta)$, we need to compute its gradient with respect to the policy parameters θ:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right]$$

where:

- $J(\theta)$ is the expected discounted return under the policy π_θ.
- τ represents a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ sampled from the policy π_θ.
- $p_\theta(\tau)$ is the probability distribution over trajectories induced by the policy π_θ.

Policy Gradient Objective

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right]$$

Computing this gradient directly is challenging because:

- The expectation is taken over all possible trajectories τ, which can be a vast or even infinite set.
- The probability distribution $p_\theta(\tau)$ over trajectories depends on the policy parameters θ in a complex way.
- The trajectory distribution $p_\theta(\tau)$ may be intractable to compute.
- The reward function $\mathcal{R}(s_t, a_t)$ may not be differentiable with respect to θ.

Policy Gradient Theorem

- Will give us a way to estimate the gradient of our objective with respect to our policy network parameters

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \right]$$

Policy-Gradient Theorem (Part 1)

Express the expected discounted return $J(\theta)$ in terms of the state-value function $V^{\pi_\theta}(s)$:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \right] \\ &= \mathbb{E}_{s_0 \sim p(s_0)} \left[V^{\pi_\theta}(s_0) \right] \\ &= \sum_{s_0} p(s_0) V^{\pi_\theta}(s_0) \end{aligned}$$

where:

- $V^{\pi_\theta}(s) = \mathbb{E}_{\tau \sim p_\theta(\tau | s_0 = s)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \right]$ is the state-value function.
- $p(s_0)$ is the initial state distribution.

Recall: Action-Value (Q) Function

- Relationship between value function and Q-function:

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

- Can recover the value function from the Q-function by marginalizing over all possible actions.

Policy-Gradient Theorem (Part 2)

Let's differentiate $J(\theta) = \sum_{s_0} p(s_0) V^{\pi_\theta}(s_0)$ with respect to θ:

$$\nabla_\theta J(\theta) = \sum_{s_0} p(s_0) \nabla_\theta V^{\pi_\theta}(s_0) \qquad \text{(Step 1: Differentiate w.r.t. } \theta\text{)}$$

$$= \sum_{s_0} p(s_0) \nabla_\theta \sum_a \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 2: Expand } V^{\pi_\theta}(s_0)\text{)}$$

$$= \sum_{s_0} p(s_0) \sum_a \nabla_\theta \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 3: Interchange } \nabla_\theta \text{ and } \sum_a\text{)}$$

$$= \sum_{s_0} p(s_0) \sum_a \pi_\theta(a|s_0) \frac{\nabla_\theta \pi_\theta(a|s_0)}{\pi_\theta(a|s_0)} Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 4: Multiply and divide by } \pi_\theta(a|s_0)\text{)}$$

$$= \sum_{s_0} p(s_0) \sum_a \pi_\theta(a|s_0) \nabla_\theta \log \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 5: Log-trick)}$$

Policy Gradient Theorem (Part 3)

Express the gradient $\nabla_\theta J(\theta)$ as an expectation over states and actions:

$$\nabla_\theta J(\theta) = \sum_{s_0} p(s_0) \sum_a \pi_\theta(a|s_0) \nabla_\theta \log \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a)$$

$$= \sum_{s_0} \sum_a p(s_0) \pi_\theta(a|s_0) \nabla_\theta \log \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 1: Rearrange terms)}$$

$$= \sum_{s_0} \sum_a p(s_0, a) \nabla_\theta \log \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \qquad \text{(Step 2: Define joint distribution } p(s_0, a)\text{)}$$

$$= \mathbb{E}_{s_0 \sim p(s_0), a \sim \pi\theta(a|s_0)} \left[\nabla_\theta \log \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \right] \qquad \text{(Step 3: Express as an expectation)}$$

$$= \mathbb{E}_{s \sim p^{\pi_\theta}(s), a \sim \pi_\theta(a|s)} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right] \qquad \text{(Step 4: Generalize to state distribution)}$$

where $p^{\pi_\theta}(s)$ is the state distribution induced by the policy π_θ.

Policy Gradient Theorem

Policy gradient theorem expresses the gradient of the expected discounted return as an expectation over states and actions, weighted by the gradient of the log policy and the action-value function:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^{\pi_\theta}(s), a \sim \pi_\theta(a|s)}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)]$$

where $p^{\pi_\theta}(s)$ is the state distribution induced by the policy π_θ.

REINFORCE Algorithm

Key ideas:

- Estimate the policy gradient:

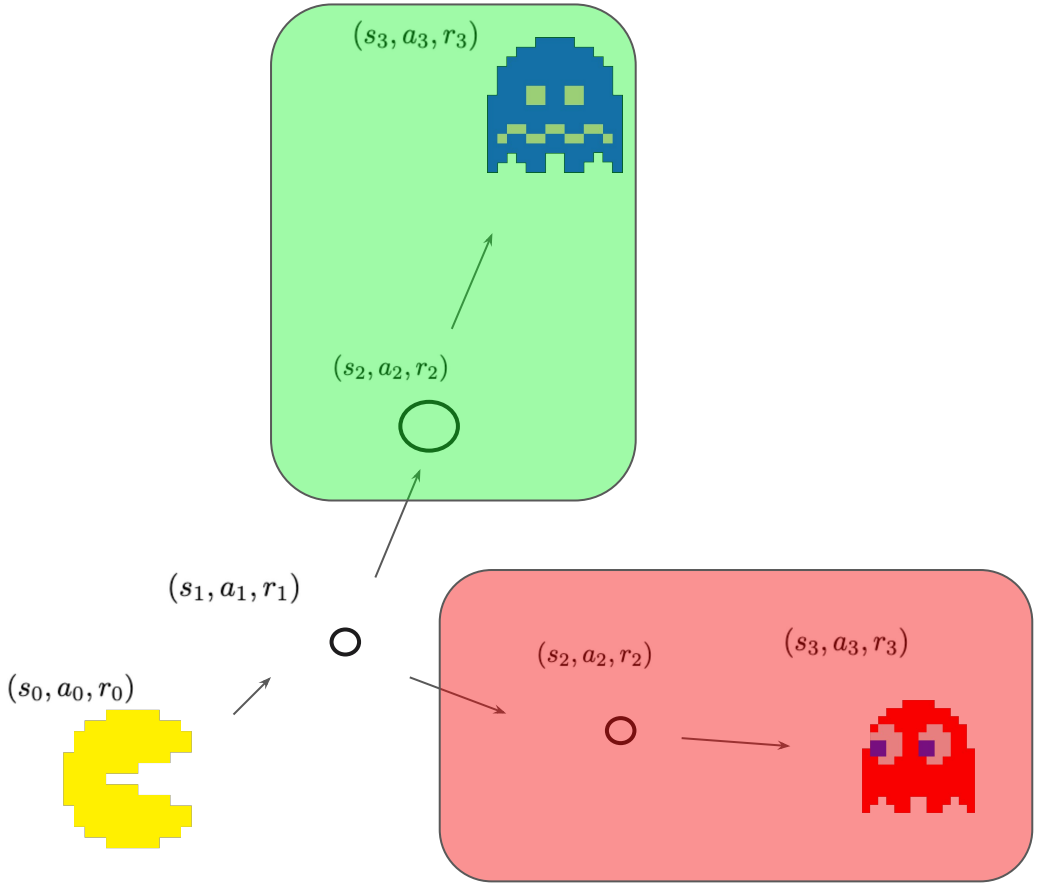
$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^{\pi_\theta}(s), a\sim\pi_\theta(a|s)}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)]$$

  using samples from the policy.
- Use the return $G_t = \sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1}$ as an unbiased estimate of the action-value function $Q^{\pi_\theta}(s_t, a_t)$.
- Update the policy parameters θ in the direction of the estimated gradient.

$$G_t \approx Q^{\pi_\theta}(s_t, a_t) = \mathbb{E}_{\tau\sim p_\theta(\tau|s_t,a_t)} \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k})\right]$$

REINFORCE Algorithm:

1. Run a policy over the environment
2. Record actions, states, and rewards
3. Increase probability of good actions and decrease probability of bad actions



REINFORCE Algorithm

Algorithm 1 REINFORCE Algorithm

Initialize policy parameters θ randomly
for each episode **do**
    Sample a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ following the policy π_θ
    for each timestep t in the trajectory **do**
        Compute the return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$
        Estimate the policy gradient: $\hat{g} = \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$
    end for
    Update the policy parameters: $\theta \leftarrow \theta + \alpha \frac{1}{T} \sum_{t=0}^{T} \hat{g}$, where α is the learning rate
end for

REINFORCE: Demo

REINFORCE Algorithm: Advantages and Disadvantages

Advantages:

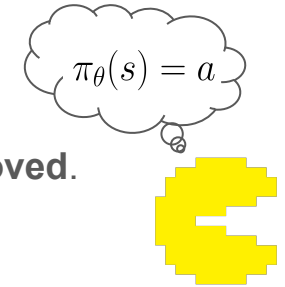
- **Unbiased estimate of the policy gradient.**
- **Can be applied to both continuous and discrete action spaces.**

Limitations:

- **Return is a noisy estimate of the action-value function.**
  - **High variance in the gradient estimates.**
- **High variance leads to potential instabilities and slow convergence.**

On-policy vs. Off-policy

Reinforcement learning algorithms can be categorized as either on-policy or off-policy based on how they collect and use data for learning.

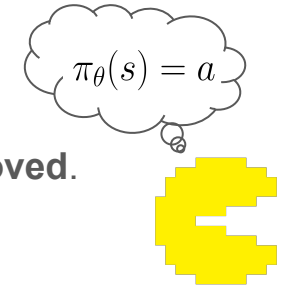


On-Policy Algorithms:

- Learn from experiences generated by the **current policy** being **followed** and **improved**.
- The **policy** used for **generating** data is the **same** as the policy being **improved**.

On-policy vs. Off-policy

Reinforcement learning algorithms can be categorized as either on-policy or off-policy based on how they collect and use data for learning.

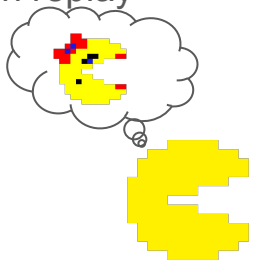


On-Policy Algorithms:

- Learn from experiences generated by the **current policy** being **followed** and **improved**.
- The **policy** used for **generating** data is the **same** as the policy being **improved**.

Off-Policy Algorithms:

- Learn from experiences generated by a **different** policy than the one being **improved**.
- The behavior policy (used for **generating** data) is **different** from the target policy (being **learned**).
- Can learn from data generated by **any policy** and enables efficient use of data through replay buffers.



Discuss: Which of the following are on- vs. off-policy algorithms?

Q-Learning, Deep Q-Learning, REINFORCE

On-Policy Learning

Advantages:

- Simpler to implement and understand
- More stable learning process with fewer hyperparameters to tune
- Better performance in certain tasks requiring specialized behaviors
- Convergence properties are often better understood theoretically
- Can be more suitable for problems where exploration must be carefully controlled

Disadvantages:

- Less sample efficient as data can only be used once
- Requires new data collection for every policy update
- Exploration is more challenging since it must be built into the policy being learned
- Cannot learn from demonstrations or historical data
- Usually requires more environment interactions to achieve good performance

Off-Policy Learning

Advantages:

- More sample efficient through experience replay
- Can learn from demonstrations, historical data, or other agents
- Better exploration-exploitation balance through separate behavior policies
- Can reuse past experiences multiple times for learning
- Enables learning optimal policies while following exploratory behavior

Disadvantages:

- Often more complex to implement correctly
- Can suffer from instability issues during learning
- May require importance sampling corrections for policy differences
- Typically needs more hyperparameter tuning
- Can be prone to overestimation bias in value functions

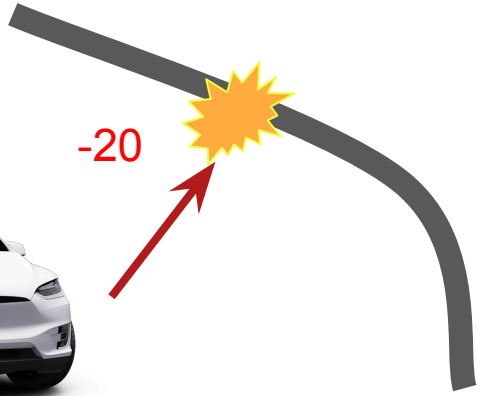
Issues with Policy Gradient Methods

- Exploration
- High variance
- Reward hacking

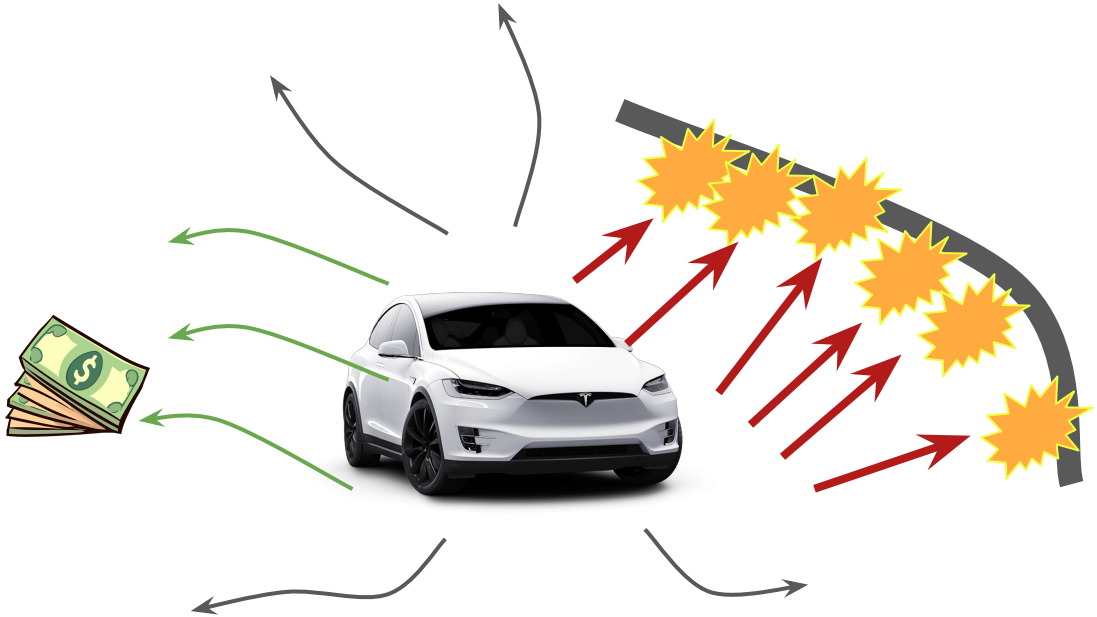
Exploration

1. Might get stuck in Local Optima
2. Want to discover better policies
3. Adapt to changing environments
4. Generalize and handle uncertainty

Exploration



Exploration



Exploration

Solutions to exploration?

+1000

1. Stochastic policy

2. Change our reward to emphasize exploration
   a. Entropy bonus

3. Run algo from different start states

Solutions to exploration?

1. Stochastic policy

2. Change our reward to emphasize exploration
   a. Entropy bonus

3. Run algo from different start states

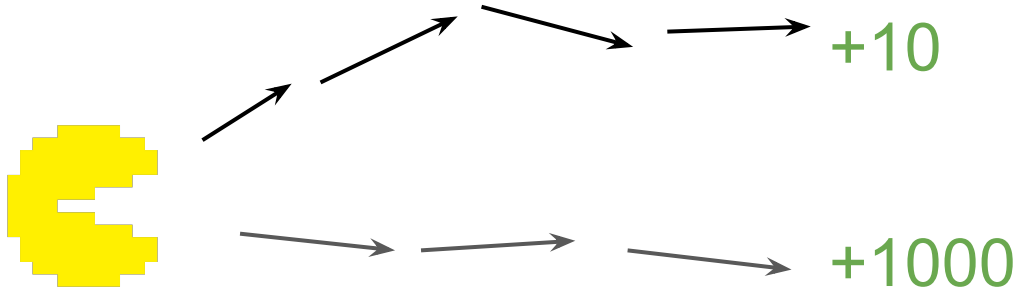
+1000

Variance

- Your policy is dependent on randomness
- Noise = high variance in how you evaluate your policy
- Need relative information about how good a policy is

Is this a good sequence of actions?

$$\hat{g} = \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

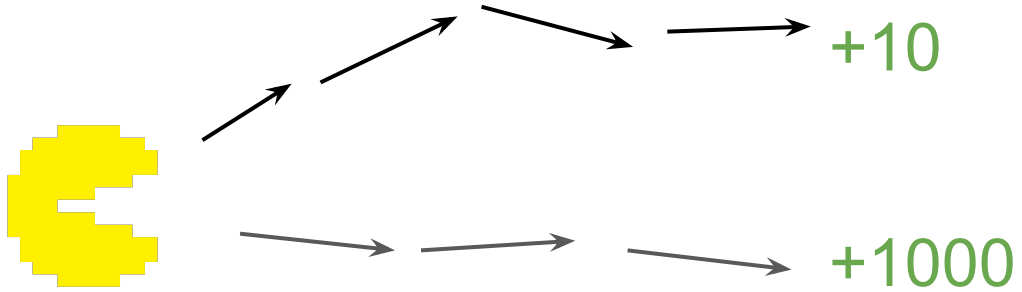


Variance Reduction with a Baseline

How good was that sequence of actions compared to other sequences?

$$\hat{g} = \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))$$

Subtract a Baseline

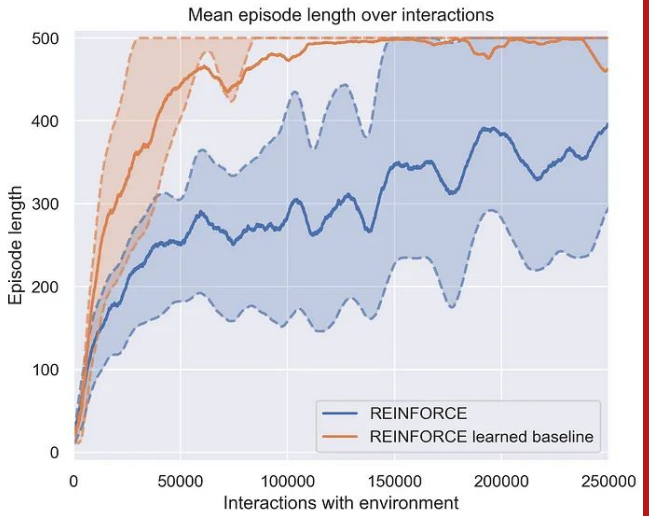


https://medium.com/@fork.tree.ai/understanding-baseline-techniques-for-reinforce-53a1e2279b57

Variance Reduction with a Baseline

- Subtracting a baseline is a simple technique to reduce variance in the REINFORCE algorithm.
  - Improves learning stability and convergence speed.
- A baseline, $b(s_t)$, estimates the expected return from state s_t.
  - Can be an estimated state-value function $V^{\pi_\theta}(s_t)$ or a moving average of returns.
- The policy gradient estimate becomes:

$$\hat{g} = \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))$$



https://medium.com/@fork.tree.ai/understanding-baseline-techniques-for-reinforce-53a1e2279b57

Actor-Critic Algorithms

Combine the benefits of both policy-based and value-based approaches.

Key Components:

- **Actor**: The actor is a policy network $\pi_\theta(a|s)$ that maps states to probability distributions over actions. It selects actions based on the current policy.
- **Critic**: The critic is a value network $V_\phi(s)$ or $Q_\phi(s, a)$ that estimates the expected return or action-value function. It evaluates the quality of the actor's decisions.

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)$$

Actor-Critic Algorithms

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) Q_\phi(s,a)$$

Moving right sounds like a good choice

Don't do that. Bad Idea!

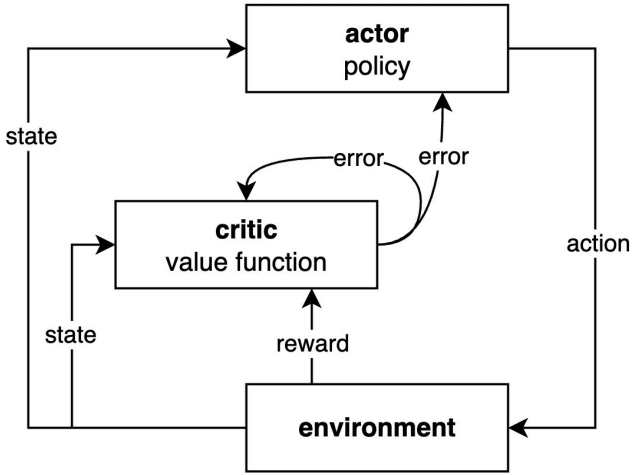
Actor-Critic Algorithm

Algorithm 3 Actor-Critic Algorithm (Q-Function Critic)

Initialize actor network $\pi_\theta(a|s)$ with random weights θ
Initialize critic network $Q_\phi(s, a)$ with random weights ϕ
for each episode **do**
    Initialize state s
    for each step of the episode **do**
        Choose action $a \sim \pi_\theta(a|s)$
        Take action a, observe reward r and next state s'
        Choose next action $a' \sim \pi_\theta(a|s')$
        Compute TD error: $\delta = r + \gamma Q_\phi(s', a') - Q_\phi(s, a)$
        Update critic weights ϕ using TD learning:
            $\phi \leftarrow \phi + \alpha_c \delta \nabla_\phi Q_\phi(s, a)$
        Compute policy gradient:
            $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)$
        Update actor weights θ using policy gradient ascent:
            $\theta \leftarrow \theta + \alpha_a \nabla_\theta J(\theta)$
        $s \leftarrow s'$
    end for
end for

Actor-Critic: Advantages

- Combine the **benefits** of **policy**-based and **value**-based methods.
- The critic helps in **reducing the variance** of the policy gradient estimates.
- The actor allows for **continuous** and **stochastic** action spaces.
- Can be more **sample-efficient** compared to pure policy-based methods.



Challenges with RL in the Real World

Sample Inefficiency + Danger + Cost = Simulation

Reward hacking

- Learn to maximize the reward in unexpected ways.
- How do we get what we want, NOT what we say we want?
- It is important to have a good reward function

Recap

- Policy gradient methods directly learn the policy function
  - Policy Gradient Theorem lets us estimate the gradient of expected reward with respect to the policy parameters
- REINFORCE learns the policy network parameters with an unbiased estimate of the gradient
  - But can have high variance!
  - Exploration must be integrated somehow
- Off-policy vs. On-policy algorithms both have their place with pros and cons
- Actor-Critic algorithms introduce a learnable critic to estimate the gradient
  - Reduces variance
- It is important to have a good reward function (that cannot easily be hacked)