Agent and Environment

Agent:
- Perceives environment.
- Makes decisions.
- Aims to maximize reward.

Environment:
- 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 \( \langle S, A, P, R, \gamma \rangle \):
  - \( S \): Set of states representing the environment.
  - \( A \): Set of actions the agent can take.
  - \( P \): Transition probability function, \( P(s'|s,a) \).
  - \( R \): Reward function, \( R(s,a) \).
  - \( \gamma \): Discount factor, \( \gamma \in [0,1] \).
Q-Learning Update Rule

The Q-Learning update rule can be expanded as:

\[ Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) \max_{a' \in A} Q^*(s', a') \]

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

\[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[ R(s, a) + \gamma \max_{a'} Q(s', a') \right] \]
Goal of Reinforcement Learning

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

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

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

where:

- $J(\theta)$ is the expected discounted return under the policy $\pi_\theta$.
- $\tau$ 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 $\pi_\theta$.
- $p_\theta(\tau)$ is the probability distribution over trajectories induced by the policy $\pi_\theta$. 
Deep Q-Learning:

$$\pi(s) = \arg\max_a Q(s, a)$$
Deep Q-Learning:

This max function makes continuous and stochastic actions hard.
Deep Q-Learning:

What if we can learn the policy directly?

$\pi(s) = \arg\max_a (Q(s, a))$
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?

\[ \pi(s) \sim P(a|s) \quad \mathcal{N}(\mu, \sigma^2) \]
Review: The Reparameterization Trick

\[ z = \mu + \sigma \odot \epsilon \]

\[ z \sim \mathcal{N}(\mu, \sigma^2 I) \]
Continuous Policy Gradient:

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

$$\mathcal{N}(\mu, \sigma^2)$$
Goal of Reinforcement Learning

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

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

$$\pi^*_\theta = \arg \max_{\theta} J(\theta)$$
What is our loss function?

**Gradient Descent:**

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

**Objective function:**

\[ \theta_{k+1} = \theta_k + \alpha \nabla_{\theta} J(\theta_k) \]
Policy Gradient Objective

Aim to update the policy parameters $\theta$ in the direction of the gradient $\nabla_{\theta} J(\theta)$ to maximize the expected discounted return:

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

where:

- $\theta_k$ represents the policy parameters at iteration $k$.
- $\alpha$ is the learning rate that controls the step size of the parameter update.
- $\nabla_{\theta} J(\theta_k)$ is the gradient of the expected discounted return with respect to the policy parameters at iteration $k$. 
Policy Gradient Objective

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

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

where:

- $J(\theta)$ is the expected discounted return under the policy $\pi_{\theta}$.
- $\tau$ 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 $\pi_{\theta}$.
- $p_{\theta}(\tau)$ is the probability distribution over trajectories induced by the policy $\pi_{\theta}$. 
Discuss

What are some of the challenges with computing this objective:

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

where:

- \( J(\theta) \) is the expected discounted return under the policy \( \pi_\theta \).
- \( \tau \) 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 \( \pi_\theta \).
- \( p_\theta(\tau) \) is the probability distribution over trajectories induced by the policy \( \pi_\theta \).
Policy Gradient Theorem

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

\[
\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right]
\]
Policy-Gradient Theorem (Part 1)

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

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

$$= \mathbb{E}_{s_0 \sim p(s_0)} [V^{\pi_{\theta}}(s_0)]$$

$$= \sum_{s_0} p(s_0) V^{\pi_{\theta}}(s_0)$$

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 $\theta$:

$$
\nabla_\theta J(\theta) = \sum_{s_0} p(s_0)\nabla_\theta V^{\pi_\theta}(s_0)
$$

(Step 1: Differentiate w.r.t. $\theta$)

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

(Step 2: Expand $V^{\pi_\theta}(s_0)$)

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

(Step 3: Interchange $\nabla_\theta$ and $\sum_a$)

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

(Step 4: Multiply and divide by $\pi_\theta(a|s_0)$)

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

(Step 5: Log-trick)
Policy Gradient Theorem (Part 3)

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

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

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

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

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

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

where $p^\pi_\theta(s)$ is the state distribution induced by the policy $\pi_\theta$. 
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 \( \pi_\theta \).
KEY IDEAS:

- **Estimate the policy gradient:**
  \[
  \nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^{\pi_\theta}(s), a \sim \pi_\theta(a|s)} \left[ \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right]
  \]
  using samples from the policy.

- **Use the return**
  \[G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\]
  as an unbiased estimate of the action-value function
  \[Q^{\pi_\theta}(s_t, a_t)\].

- **Update the policy parameters** \(\theta\) in the direction of the estimated gradient.
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
Algorithm 1 REINFORCE Algorithm

1: Initialize policy parameters $\theta$ randomly
2: for each episode do
3:     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 $\pi_\theta$
4:     for each timestep $t$ in the trajectory do
5:         Compute the return $G_t = \sum_{k=0}^{T-t} \gamma^k T_{t+k}$
6:         Estimate the policy gradient: $\hat{g} = \nabla_\theta \log \pi_\theta(a_t | s_t) G_t$
7:     end for
8:     Update the policy parameters: $\theta \leftarrow \theta + \alpha \frac{1}{T} \sum_{t=0}^{T} \hat{g}$, where $\alpha$ is the learning rate
9: end for
REINFORCE: Demo

Episode = 1582
Episode reward = 97.0
Average of last 500 rewards = 72.552
Average of last 100 rewards = 94.08
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.
- The policy used for generating data is the same as the policy being improved.

\[ \pi_\theta(s) = a \]
Reinforcement learning algorithms can be categorized as either on-policy or off-policy based on how they collect and use data for learning.

**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
Issues with Policy Gradient Methods
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?

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
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:

  \[
  \hat{g} = \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))
  \]
Typically train image captioning models with the cross-entropy loss

- Would be nice to directly optimize for downstream metrics of interest
  - Typically n-gram metrics of some form
Self-Critical Sequence Training

- Train image captioning model with the cross-entropy objective
- Use REINFORCE to optimize some non-differentiable metric
  - CIDEr: Average cosine similarity of n-gram feature vectors for the candidate and reference

Self-Critical Sequence Training

\[ h_0, c_0 = LSTM(BOS, h_0, c_0) \]
\[ h_1, c_1 = LSTM(BOS, h_0, c_0) \]
\[ h_2, c_2 = LSTM(w_1^s, h_1, c_1) \]
\[ (w_1^s, \ldots, w_T^s) \]
\[ p_\theta(w|h_1) \]
\[ p_\theta(w|h_2) \]
\[ \text{sample} \]
\[ \text{sample} \]
\[ \text{CIDER} \]
\[ r(w_1^s, \ldots, w_T^s) \]
\[ \{w_1^*, \ldots, w_T^*\} \]
\[ (r(w_1^s, \ldots, w_T^s) - r(\hat{w}_1, \ldots, \hat{w}_T)) \nabla_\theta \log p_\theta(w_1^s, \ldots, w_T^s) \]

Self-Critical Sequence Training

- Can directly optimize non-differentiable downstream metrics with REINFORCE

<table>
<thead>
<tr>
<th>Training Metric</th>
<th>Evaluation Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIDEr</td>
</tr>
<tr>
<td>XE</td>
<td>90.9</td>
</tr>
<tr>
<td>XE (beam)</td>
<td>94.0</td>
</tr>
<tr>
<td>CIDEr</td>
<td>106.3</td>
</tr>
<tr>
<td>BLEU</td>
<td>94.4</td>
</tr>
<tr>
<td>ROUGE4L</td>
<td>97.7</td>
</tr>
<tr>
<td>METEOR</td>
<td>80.5</td>
</tr>
</tbody>
</table>
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!
Algorithm 3 Actor-Critic Algorithm (Q-Function Critic)

1: Initialize actor network $\pi_\theta(a|s)$ with random weights $\theta$
2: Initialize critic network $Q_\phi(s, a)$ with random weights $\phi$
3: for each episode do
4:     Initialize state $s$
5:     for each step of the episode do
6:         Choose action $a \sim \pi_\theta(a|s)$
7:         Take action $a$, observe reward $r$ and next state $s'$
8:         Choose next action $a' \sim \pi_\theta(a'|s')$
9:         Compute TD error: $\delta = r + \gamma Q_\phi(s', a') - Q_\phi(s, a)$
10:        Update critic weights $\phi$ using TD learning:
11:            $\phi \leftarrow \phi + \alpha_\phi \delta \nabla_\phi Q_\phi(s, a)$
12:        Compute policy gradient:
13:            $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)$
14:        Update actor weights $\theta$ using policy gradient ascent:
15:            $\theta \leftarrow \theta + \alpha_\alpha \nabla_\theta J(\theta)$
16:     s $\leftarrow s'$
17:     end for
18: 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!
- Actor-Critic algorithms introduce a learnable critic to estimate the gradient
  - Reduces variance