Topics in AI (CPSC 532L):
Multimodal Learning with Vision, Language and Sound

Lecture 12: Deep Reinforcement Learning
Types of **Learning**

**Supervised** training
- Learning from the teacher
- Training data includes desired output

**Unsupervised** training
- Training data does not include desired output

**Reinforcement** learning
- Learning to act under evaluative feedback (rewards)

* slide from Dhruv Batra
What is Reinforcement Learning

**Agent-oriented learning** — learning by interacting with an environment to achieve a goal
- More realizing and ambitious than other kinds of machine learning

Learning **by trial and error**, with only delayed evaluative feedback (reward)
- The kind go machine learning most like natural learning
- Learning that can tell for itself when it is right or wrong

*slide from David Silver*
Example: Hajime Kimura’s RL Robot
Example: Hajime Kimura’s RL Robot

Before

After

* slide from Rich Sutton
Example: Hajime Kimura’s RL Robot

Before

After

* slide from Rich Sutton
Challenges of RL

– Evaluative feedback (reward)
– Sequentiality, delayed consequences
– Need for trial and error, to explore as well as exploit
– Non-stationarity
– The fleeting nature of time and online data

* slide from Rich Sutton
How does RL work?

* slide from David Silver

- At each step $t$ the agent:
  - Executes action $a_t$
  - Receives observation $o_t$
  - Receives scalar reward $r_t$
- The environment:
  - Receives action $a_t$
  - Emits observation $o_{t+1}$
  - Emits scalar reward $r_{t+1}$
Robot Locomotion

**Objective**: Make the robot move forward

**State**: Angle and position of the joints

**Action**: Torques applied on joints

**Reward**: 1 at each time step upright + forward movement

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Objective: Complete the game with the highest score

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Objective: Win the game!

State: Position of all pieces

Action: Where to put the next piece down

Reward: 1 if win at the end of the game, 0 otherwise
Markov Decision Processes

— Mathematical **formulation** of the RL problem

**Defined by:**

\[ S: \text{set of possible states} \]
\[ A: \text{set of possible actions} \]
\[ R: \text{distribution of reward given (state, action) pair} \]
\[ P: \text{transition probability i.e. distribution over next state given (state, action) pair} \]
\[ \gamma: \text{discount factor} \]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Markov Decision Processes

— Mathematical **formulation** of the RL problem

**Defined** by:

- \( S \): set of possible states
- \( A \): set of possible actions
- \( \mathcal{R} \): distribution of reward given (state, action) pair
- \( \mathbb{P} \): transition probability i.e. distribution over next state given (state, action) pair
- \( \gamma \): discount factor

— Life is **trajectory**: \[ \ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots \]
Markov Decision Processes

— Mathematical **formulation** of the RL problem

**Defined** by:

- $S$: set of possible states
- $A$: set of possible actions
- $R$: distribution of reward given (state, action) pair
- $P$: transition probability i.e. distribution over next state given (state, action) pair
- $\gamma$: discount factor

— Life is **trajectory**: $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$

— **Markov property**: Current state completely characterizes the state of the world

$$p(r, s'|s, a) = \text{Prob}\left[R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a \right]$$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Components of the RL Agent

Policy
  — How does the agent behave?

Value Function
  — How good is each state and/or action pair?

Model
  — Agent’s representation of the environment
Policy

— The policy is how the agent acts
— Formally, map from states to actions:

**Deterministic** policy: \( a = \pi(s) \)

**Stochastic** policy: \( \pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \)

* slide from Dhruv Batra*
Policy

— The policy is how the agent acts
— Formally, map from states to actions:

**Deterministic** policy: \( a = \pi(s) \)

**Stochastic** policy: \( \pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \)

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>

*e.g.*

* slide from Dhruv Batra*
The **Optimal** Policy

What is a good policy?
The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

* slide from Dhruv Batra*
The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

**Discounted future rewards!**
The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

**Discounted future rewards!**

Formally: \[ \pi^* = \arg \max_\pi \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right] \]

with \( s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t) \)

* slide from Dhruv Batra
Components of the RL Agent

✓ Policy
  — How does the agent behave?

Value Function
  — How good is each state and/or action pair?

Model
  — Agent’s representation of the environment

* slide from Dhruv Batra
Value Function

A value function is a prediction of future reward

“State Value Function” or simps “Value Function”
  — How good is a state?
  — Am I screwed? Am I winning this game?

“Action Value Function” or Q-function
  — How good is a state action-pair?
  — Should I do this now?

* slide from Dhruv Batra
Value Function and Q-value Function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

— The **value function** (how good is the state) at state $s$, is the expected cumulative reward from state $s$ (and following the policy thereafter):

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Value Function and Q-value Function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

— The **value function** (how good is the state) at state $s$, is the expected cumulative reward from state $s$ (and following the policy thereafter):

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

— The **Q-value function** (how good is a state-action pair) at state $s$ and action $a$, is the expected cumulative reward from taking action $a$ in state $s$ (and following the policy thereafter):

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Components of the RL Agent

✓ Policy
  - How does the agent behave?

✓ Value Function
  - How good is each state and/or action pair?

Model
  - Agent’s representation of the environment

* slide from Dhruv Batra
Model

Model predicts what the world will do next

* slide from David Silver
Model

Model predicts what the world will do next

* slide from David Silver
Components of the RL Agent

✓ Policy
  - How does the agent behave?

✓ Value Function
  - How good is each state and/or action pair?

✓ Model
  - Agent’s representation of the environment

* slide from Dhruv Batra
Maze Example

**Reward:** -1 per time-step

**Actions:** N, E, S, W

**States:** Agent’s location

* slide from David Silver
Maze Example: Policy

Arrows represent a policy $\pi(s)$ for each state $s$.

* slide from David Silver
Maze Example: Value

Numbers represent value $V_\pi(s)$ of each state $s$
Maze Example: Model

Grid layout represents transition model

Numbers represent the immediate reward for each state (same for all states)

* slide from David Silver
Components of the RL Agent

**Policy**
- How does the agent behave?

**Value Function**
- How good is each state and/or action pair?

**Model**
- Agent’s representation of the environment
## Approaches to RL: Taxonomy

### Model-free RL

**Value-based RL**
- Estimate the optimal action-value function $Q^*(s, a)$
- No policy (implicit)

**Policy-based RL**
- Search directly for the optimal policy $\pi^*$
- No value function

### Model-based RL
- Build a model of the world
- Plan (e.g., by look-ahead) using model

* slide from Dhruv Batra
## Approaches to RL: Taxonomy

<table>
<thead>
<tr>
<th>Model-free RL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong>-based RL</td>
</tr>
<tr>
<td>- Estimate the optimal action-value function $Q^*(s, a)$</td>
</tr>
<tr>
<td>- No policy (implicit)</td>
</tr>
<tr>
<td><strong>Policy</strong>-based RL</td>
</tr>
<tr>
<td>- Search directly for the optima policy $\pi^*$</td>
</tr>
<tr>
<td>- No value function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model-based RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Build a model of the world</td>
</tr>
<tr>
<td>- Plan (e.g., by look-ahead) using model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actor-critic RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Value function</td>
</tr>
<tr>
<td>- Policy function</td>
</tr>
</tbody>
</table>

* slide from Dhruv Batra
Deep RL

**Value-based RL**
- Use neural nets to represent Q function
  \[ Q(s, a; \theta) \]
  \[ Q(s, a; \theta^*) \approx Q^*(s, a) \]

**Policy-based RL**
- Use neural nets to represent the policy
  \[ \pi_\theta \]
  \[ \pi_\theta^* \approx \pi^* \]

**Model-based RL**
- Use neural nets to represent and learn the model

* slide from Dhruv Batra
Approaches to RL

**Value-based RL**
- Estimate the optimal action-value function $Q^*(s, a)$
- No policy (implicit)
Optimal Value Function

Optimal Q-function is the maximum achievable value

\[ Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = Q^{\pi^*}(s, a) \]
Optimal Value Function

Optimal Q-function is the maximum achievable value

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

Once we have it, we can act optimally

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

* slide from David Silver
Optimal Value Function

Optimal Q-function is the maximum achievable value

\[ Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = Q^{\pi^*}(s, a) \]

Once we have it, we can act optimally

\[ \pi^*(s) = \arg\max_a Q^*(s, a) \]

Optimal value maximizes over all future decisions

\[
Q^*(s, a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + ... \\
= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})
\]

* slide from David Silver
**Optimal Value Function**

Optimal Q-function is the maximum achievable value

\[ Q^*(s, a) = \max_\pi Q^\pi(s, a) = Q^\pi^*(s, a) \]

Once we have it, we can act optimally

\[ \pi^*(s) = \arg\max_a Q^*(s, a) \]

Optimal value maximizes over all future decisions

\[
Q^*(s, a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots
\]

\[ = r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \]

Formally, \( Q^* \) satisfied Bellman Equations

\[
Q^*(s, a) = \mathbb{E}_{s'} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]
\]

* slide from David Silver*
Solving for the Optimal Policy

**Value iteration** algorithm: Use Bellman equation as an iterative update

\[ Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right] \]

Q_i will converge to Q* as i -> infinity
Solving for the Optimal Policy

**Value iteration** algorithm: Use Bellman equation as an iterative update

\[ Q_{i+1}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s', a') | s, a] \]

\( Q_i \) will converge to \( Q^* \) as \( i \to \infty \)

**What’s the problem with this?**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Solving for the Optimal Policy

**Value iteration** algorithm: Use Bellman equation as an iterative update

\[ Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right] \]

\( Q_i \) will converge to \( Q^* \) as \( i \to infinity \)

What’s the problem with this?

**Not scalable.** Must compute \( Q(s, a) \) for every state-action pair. If state is e.g. game pixels, computationally infeasible to compute for entire state space!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Solving for the Optimal Policy

**Value iteration** algorithm: Use Bellman equation as an iterative update

\[ Q_{i+1}(s,a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s',a') \mid s,a \right] \]

\( Q_i \) will converge to \( Q^* \) as \( i \to \infty \)

**What's the problem with this?**

**Not scalable.** Must compute \( Q(s,a) \) for every state-action pair. If state is e.g. game pixels, computationally infeasible to compute for entire state space!

**Solution:** use a function approximator to estimate \( Q(s,a) \). E.g. a neural network!

*slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford*
Q-Networks

\[
Q(s, a, w) \approx Q^*(s, a)
\]

* slide from David Silver
Case Study: Playing **Atari** Games

[ Mnih et al., 2013; Nature 2015 ]

**Objective:** Complete the game with the highest score

**State:** Raw pixel inputs of the game state

**Action:** Game controls e.g. Left, Right, Up, Down

**Reward:** Score increase/decrease at each time step

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Q-Network Architecture

\[ Q(s, a; \theta) \]: neural network with weights \( \theta \)

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

[ Mnih et al., 2013; Nature 2015 ]
Q-Network Architecture

\[ Q(s, a; \theta) \]: neural network with weights \( \theta \)

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

[ Mnih et al., 2013; Nature 2015 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Q-Network Architecture

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

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Mnih et al., 2013; Nature 2015 ]
Q-Network Architecture

\[ Q(s, a; \theta) : \text{neural network} \]

with weights \( \theta \)

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

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Mnih et al., 2013; Nature 2015 ]
Q-Network Architecture

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

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

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

Number of actions between 4-18 depending on Atari game

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Mnih et al., 2013; Nature 2015 ]
Q-Network Architecture

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

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

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

A single feedforward pass to compute Q-values for all actions from the current state => efficient!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs213n Stanford

[ Mnih et al., 2013; Nature 2015 ]

Number of actions between 4-18 depending on Atari game
Q-Network Learning

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

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$$
Q-Network Learning

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

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$$

**Forward Pass:**

Loss function: $$L_i(\theta_i) = \mathbb{E} \left[ (y_i - Q(s, a; \theta_i))^2 \right]$$

where $$y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Q-Network Learning

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

\[ Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \]

**Forward Pass:**

Loss function: \[ L_i(\theta_i) = \mathbb{E} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \]

where \[ y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \]

**Backward Pass:**

Gradient update (with respect to Q-function parameters \( \theta \)):

\[ \nabla_{\theta_i} L_i(\theta_i) = \mathbb{E} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Q-Network Learning

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

\[ Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \]

**Forward Pass:**

Loss function:

\[ L_i(\theta_i) = \mathbb{E} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \]

where

\[ y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \]

**Backward Pass:**

Gradient update (with respect to Q-function parameters \( \theta \)):

\[ \nabla_{\theta_i} L_i(\theta_i) = \mathbb{E} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i) \]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Training the Q-Network: **Experience Replay**

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

Address these problems using experience replay
- Continually update a replay memory table of transitions \((s_t, a_t, r_t, s_{t+1})\) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Experience Replay

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Experience Replay

To remove correlations, build data-set from agent's own experience

\[
\begin{array}{c|c|c|c|c}
s_1, a_1, r_2, s_2 \\
| s_2, a_2, r_3, s_3 \\
| s_3, a_3, r_4, s_4 \\
| \cdots \\
| s_t, a_t, r_{t+1}, s_{t+1} \\
\end{array}
\rightarrow
\begin{array}{c}
s, a, r, s' \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
  s_t, a_t, r_{t+1}, s_{t+1} \\
\end{array}
\rightarrow
\begin{array}{c}
  s_t, a_t, r_{t+1}, s_{t+1} \\
\end{array}
\]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Example: Atari Playing

Starting out - 10 minutes of training

The algorithm tries to hit the ball back, but it is yet too clumsy to manage.
Example: Atari Playing

Starting out - 10 minutes of training

The algorithm tries to hit the ball back, but it is yet too clumsy to manage.
Deep RL

**Value-based RL**
- Use neural nets to represent Q function
  \[ Q(s, a; \theta) \]
  \[ Q(s, a; \theta^*) \approx Q^*(s, a) \]

**Policy-based RL**
- Use neural nets to represent the policy
  \[ \pi_\theta \]
  \[ \pi_{\theta^*} \approx \pi^* \]

**Model-based RL**
- Use neural nets to represent and learn the model

* slide from Dhruv Batra
Deep RL

**Value-based RL**
- Use neural nets to represent Q function
  \[ Q(s, a; \theta) \]
  \[ Q(s, a; \theta^*) \approx Q^*(s, a) \]

**Policy-based RL**
- Use neural nets to represent the policy
  \[ \pi_\theta \]
  \[ \pi_\theta^* \approx \pi^* \]

**Model-based RL**
- Use neural nets to represent and learn the model

* slide from Dhruv Batra
**Policy Gradients**

Formally, let’s define a class of parameterized policies:

For each policy, define its value:

\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]
\]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Policy Gradients

Formally, let’s define a class of parameterized policies:

For each policy, define its value:

\[ J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] \]

We want to find the optimal policy \( \theta^* = \arg \max_\theta J(\theta) \)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Formally, let’s define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

We want to find the optimal policy $$\theta^* = \arg \max_{\theta} J(\theta)$$

How can we do this?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Formally, let’s define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

We want to find the optimal policy $$\theta^* = \arg \max_\theta J(\theta)$$

How can we do this?

Gradient ascent on policy parameters!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$$

$$= \int r(\tau) p(\tau; \theta) d\tau$$

Where $r(\tau)$ is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE algorithm

Expected reward:

\[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \]
\[ = \int_{\tau} r(\tau) p(\tau; \theta) d\tau \]

Where \( r(\tau) \) is the reward of a trajectory \( \tau = (s_0, a_0, r_0, s_1, \ldots) \)

Now let’s differentiate this:

\[ \nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau \]

Intractable! Expectation of gradient is problematic when \( p \) depends on \( \theta \)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE algorithm

Expected reward:

\[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] = \int_{\tau} r(\tau)p(\tau; \theta) d\tau \]

Where \( r(\tau) \) is the reward of a trajectory \( \tau = (s_0, a_0, r_0, s_1, \ldots) \)

Now let’s differentiate this:

\[ \nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau; \theta) d\tau \]

However, we can use a nice trick:

\[ \nabla_\theta p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_\theta p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta) \nabla_\theta \log p(\tau; \theta) \]

If we inject this back:

\[ \nabla_\theta J(\theta) = \int_{\tau} (r(\tau) \nabla_\theta \log p(\tau; \theta)) p(\tau; \theta) d\tau = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)] \]

Can estimate with Monte Carlo sampling

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Gradient estimator:

\[ \nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \]

Interpretation:
- If \( r(\tau) \) is high, push up the probabilities of the actions seen
- If \( r(\tau) \) is low, push down the probabilities of the actions seen

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Intuition**

**Gradient estimator:**

\[
\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)
\]

**Interpretation:**

- If \( r(\tau) \) is high, push up the probabilities of the actions seen
- If \( r(\tau) \) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**
Intuition

* slide from Dhruv Batra
Intuition

Gradient estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:
- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE in Action: **Recurrent Attention Model (REM)**

**Objective:** Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class
  - Inspiration from human perception and eye movements
  - Saves computational resources => scalability
  - Able to ignore clutter / irrelevant parts of image

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, [cs231n Stanford](#)

[ Mnih *et al.*, 2014 ]
REINFORCE in Action: **Recurrent Attention Model (REM)**

**Objective:** Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class
- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

**State:** Glimpses seen so far

**Action:** (x,y) coordinates (center of glimpse) of where to look next in image

**Reward:** 1 at the final timestep if image correctly classified, 0 otherwise

[ Mnih et al., 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Objective:** Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class
- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

**State:** Glimpses seen so far
**Action:** (x,y) coordinates (center of glimpse) of where to look next in image
**Reward:** 1 at the final timestep if image correctly classified, 0 otherwise

Glimpsing is a **non-differentiable operation** => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

[ Mnih *et al.*, 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE in Action: **Recurrent Attention Model (REM)**

* * slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE in Action: **Recurrent Attention Model (REM)**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, [cs231n Stanford](https://cs231n.stanford.edu)
REINFORCE in Action: **Recurrent Attention Model (REM)**

![Diagram of Recurrent Attention Model (REM)]

*slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n* Stanford

[Mnih et al., 2014]
REINFORCE in Action: **Recurrent Attention Model (REM)**

\[
(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \quad (x_4, y_4) \quad (x_5, y_5)
\]

\[
\text{Input image} \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{NN} \quad \text{Softmax}
\]

\[
y=2
\]

[ Mnih et al., 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
REINFORCE in Action: **Recurrent Attention Model (REM)**

Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[ Mnih et al., 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
REINFORCE in Action: **Recurrent Attention Model (REM)**

Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[ Mnih et al., 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
REINFORCE in Action: **Recurrent Attention Model (REM)**

Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

*Mnih et al., 2014*
REINFORCE in Action: **Recurrent Attention Model (REM)**

- Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[ Mnih et al., 2014 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Summary

**Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency

**Q-learning**: does not always work but when it works, usually more sample-efficient. **Challenge**: exploration

**Guarantees:**
— Policy Gradients: Converges to a local minima of $J(\theta)$, often good enough!
— Q-learning: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford