Intro to Reinforcement Learning Part II

Paul Liang

pliang@cs.cmu.edu

@pliang279
Acknowledgement: Much of the material and slides for this lecture were borrowed from Pieter Abbeel, Yan Duan, Xi Chen, and Andrej Karpathy’s Deep RL Bootcamp at UC Berkeley, Fei-Fei Li, Justin Johnson, and Serena Yeung’s CS231N course at Stanford, as well as Katerina Fragkiadaki and Ruslan Salakhutdinov’s 10-703 course at CMU, who in turn borrowed much from Rich Sutton’s class and David Silver’s class on Reinforcement Learning.
Recap: Markov Decision Process (MDPs)

An MDP is defined by:
- Set of states $S$
- Set of actions $A$
- Transition function $P(s' | s, a)$
- Reward function $R(s, a, s')$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$

Trajectory

$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$
Recap: Return

In continuing tasks, we often use simple total discounted reward:

\[ G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \]

\( \gamma \) close to 0 leads to "myopic" evaluation
\( \gamma \) close to 1 leads to "far-sighted" evaluation
Recap: Policy

**Definition:** A policy is a distribution over actions given states,

\[ \pi(a \mid s) = \Pr(A_t = a \mid S_t = s), \forall t \]

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

\[ \pi(s) = \text{the action taken with prob} = 1 \text{ when } S_t = s \]
Recap: MDPs, Returns, Policies

An MDP is defined by:
- Set of states $S$
- Set of actions $A$
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$

Return:

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

Goal:

$$\arg \max_{\pi} \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t R_t \mid \pi \right]$$
Reinforcement Learning vs Supervised Learning

**Reinforcement Learning**
- Sequential decision making
- Maximize cumulative reward
- Sparse rewards
- Environment maybe unknown

**Supervised Learning**
- One-step decision making
- Maximize immediate reward
- Dense supervision
- Environment always known
Recap: Exact methods

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

- **Bellman expectation equations**
  - \( Q^*(s, a) \): Q-value iteration
  - \( V^*(s) \): Value iteration
  - \( Q^\pi(s, a) \): Q-policy iteration
  - \( V^\pi(s) \): Policy iteration

Fully known MDP states transitions rewards

Repeat until policy converges. Guaranteed to converge to optimal policy.
Recap: Exact methods

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

- **Q-value iteration**
  - \( Q^*(s, a) \)
- **Value iteration**
  - \( V^*(s) \)
- **Q-policy iteration**
  - \( Q^\pi(s, a) \)
- **Policy iteration**
  - \( V^\pi(s) \)

Repeat until policy converges. Guaranteed to converge to optimal policy.

Iterate over and storage for all states and actions
Requires small, discrete state and action space
Update equations require fully observable MDP and known transitions
Recap: Tabular Q-learning

MDP with unknown transitions $\rightarrow$ Bellman optimality equations $\rightarrow$ Replace true expectation over transitions with estimates $\rightarrow$ Tabular Q-learning

$$s' \sim P(s' | s, a)$$

Simulation and exploration, epsilon greedy is important!

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

Old estimate $\rightarrow$ Target
Recap: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates → Tabular Q-learning

\[ s' \sim P(s'|s, a) \]

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

old estimate \hspace{1cm} target

\[ Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right) \]

Tabular: keep a \(|S| \times |A|\) table of \(Q(s,a)\)
Still requires small and discrete state and action space
How can we generalize to unseen states?
Recap: Deep Q-learning

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

old estimate

\[ \mathcal{L}_i(w_i) = \mathbb{E}_{s, a, r, s' \sim \mathcal{D}_i} \left[ (r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i))^2 \right] \]

Q-learning target

Q-network
Recap: Deep Q-learning

- Sample random mini-batch of transitions \((s,a,r,s')\) from \(D\)
- Compute Q-learning targets w.r.t. old, fixed parameters \(w\)
- Optimize MSE between Q-network and Q-learning targets

\[
\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i) \right)^2 \right]
\]

- Use stochastic gradient descent
- Update \(w\) with updated \(w\) every \(~1000\) iterations
Recap: Deep Q-learning

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

**Old estimate**

**Target**

\[ L_i(w_i) = \mathbb{E}_{s, a, r, s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i) \right)^2 \right] \]

Stochastic gradient descent + Exploration + Experience replay + Fixed Q-targets

Works for high-dimensional state and action spaces

Generalizes to unseen states
Recap: Obtaining the optimal policy

Optimal policy can be found by maximizing over $Q^*(s,a)$

$$\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg\max_a Q^*(s,a) \\
\epsilon, & \text{else}
\end{cases}$$
Recap: Obtaining the optimal policy

Optimal policy can be found by maximizing over $Q^*(s,a)$

$$
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a Q^*(s, a) \\
\epsilon, & \text{else}
\end{cases}
$$

Optimal policy can also be found by maximizing over $V^*(s')$ with one-step look ahead

$$
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma V^*(s') \right] \\
\epsilon, & \text{else}
\end{cases}
$$
Contents

- Policy gradient methods
- Actor-critic
- Applications: RL and language
Value-based and Policy-based RL

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. \(\epsilon\)-greedy)

\[
\begin{align*}
V^\pi(s) & \quad \text{State value functions} \\
V^*(s) & \\
Q^\pi(s, a) & \quad \text{Action value functions} \\
Q^*(s, a) &
\end{align*}
\]

\[
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a \mathbb{E}_{s'} [r(s, a, s') + \gamma V^*(s')] \\
\epsilon, & \text{else}
\end{cases}
\]

\[
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a Q^*(s, a) \\
\epsilon, & \text{else}
\end{cases}
\]
Value-based and Policy-based RL

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. $\varepsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learned Policy

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$
Directly learning the policy

- Often $\pi$ can be simpler than $Q$ or $V$
  - E.g., robotic grasp

$Q(s,a)$ and $V(s)$ very high-dimensional
But policy could be just ‘open/close hand’
Directly learning the policy

- Often \( \pi \) can be simpler than Q or V
  - E.g., robotic grasp

- V: doesn’t prescribe actions
  - Would need dynamics model (+ compute 1 Bellman back-up)

- Q: need to be able to efficiently solve \( \arg \max_u Q_\theta(s, u) \)
  - Challenge for continuous / high-dimensional action spaces

\[
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a \mathbb{E}_{s'} [r(s, a, s') + \gamma V^*(s')] \\
\epsilon, & \text{else}
\end{cases}
\]

\[
\pi^*(a|s) = \begin{cases} 
1 - \epsilon, & \text{if } a = \arg \max_a Q^*(s, a) \\
\epsilon, & \text{else}
\end{cases}
\]
Value-based and Policy-based RL

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learned Policy

- **Actor-Critic**
  - Learned Value Function
  - Learned Policy
<table>
<thead>
<tr>
<th>Conceptually:</th>
<th>Policy-based</th>
<th>Value-based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimize what you care about</td>
<td>Indirect, exploit the problem structure, self-consistency</td>
</tr>
</tbody>
</table>
Value-based and Policy-based RL

**Conceptually:**
- **Policy-based:** Optimize what you care about
- **Value-based:** Indirect, exploit the problem structure, self-consistency

**Empirically:**
- **Policy-based:**
  - More compatible with rich architectures (including recurrence)
  - More versatile
  - More compatible with auxiliary objectives
- **Value-based:**
  - More compatible with exploration and off-policy learning
  - More sample-efficient when they work
Pong from pixels
Pong from pixels

`e.g.,` [80 x 80] array of

height width
Network sees +1 if it scored a point, and -1 if it was scored against. How do we learn these parameters?
Pong from pixels

Suppose we had the training labels…
(we know what to do in any state)

(x1, UP)
(x2, DOWN)
(x3, UP)
...

Pong from pixels

Suppose we had the training labels…
(we know what to do in any state)

\[(x_1, \text{UP})\]
\[(x_2, \text{DOWN})\]
\[(x_3, \text{UP})\]
...

maximize:

\[\sum_i \log p(y_i | x_i)\]
Pong from pixels

Except, we don’t have labels...

Should we go UP or DOWN?
Pong from pixels

Let’s just act according to our current policy...

Rollout the policy and collect an episode.
Pong from pixels

Collect many rollouts...

4 rollouts:
Pong from pixels

Not sure whatever we did here, but apparently it was good.
Pong from pixels

Not sure whatever we did here, but it was bad.
Pong from pixels

Pretend every action we took here was the correct label.

maximize: \( \log p(y_i \mid x_i) \)

Pretend every action we took here was the wrong label.

maximize: \( (-1) \cdot \log p(y_i \mid x_i) \)
Pong from pixels

Discounting

Blame each action assuming that its effects have exponentially decaying impact into the future.

\[ \sum_i A_i \ast \log p(y_i | x_i) \]

Discounted rewards:

0.21  0.24  0.27  -0.81  -0.9  -1  0  0

\( \gamma = 0.9 \)

Reward +1.0

Reward -1.0
Pong from pixels

1. Initialize a policy network at random
Pong from pixels

1. Initialize a policy network at random
2. Repeat Forever:
3. Collect a bunch of rollouts with the policy \( \pi(a|s) \)
Pong from pixels

1. Initialize a policy network at random
2. Repeat Forever:
3. Collect a bunch of rollouts with the policy
4. Increase the probability of actions that worked well

Pretend every action we took here was the correct label.

\[
\text{maximize: } \log p(y_i \mid x_i)
\]

Pretend every action we took here was the wrong label.

\[
\text{maximize: } (-1) \times \log p(y_i \mid x_i)
\]

\[\sum_i A_i \times \log p(y_i \mid x_i)\]
Pong from pixels

1. Initialize a policy network at random
2. Repeat Forever:
3. Collect a bunch of rollouts with the policy
4. Increase the probability of actions that worked well

\[ \pi(a | s) \]

epsilon greedy!

Does not require transition probabilities
Does not estimate Q(), V()
Predicts policy directly

\[ \sum_i A_i \cdot \log p(y_i | x_i) \]

Pretend every action we took here was the correct label.
\[ \text{maximize: } \log p(y_i | x_i) \]

Pretend every action we took here was the wrong label.
\[ \text{maximize: } (-1) \ast \log p(y_i | x_i) \]
Pong from pixels
Policy gradients

Why does this work?

1. Initialize a policy network at random
2. Repeat Forever:
   3. Collect a bunch of rollouts with the policy
   4. Increase the probability of actions that worked well

\[ \sum_i A_i \ast \log p(y_i | x_i) \]
Policy gradients

Formally, let’s define a class of parametrized policies: \( \mathcal{P} = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \)

For each policy, define its value:

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

Writing in terms of trajectories

Probability of a trajectory

\[ p(\tau; \theta) = \pi_\theta(a_0|s_0)p(s_1|s_0, a_0) \times \pi_\theta(a_1|s_1)p(s_2|s_1, a_1) \times \pi_\theta(a_2|s_2)p(s_3|s_2, a_2) \times \ldots = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \]

Reward of a trajectory

\[ r(\tau) = \sum_{t \geq 0} \gamma^t r_t \]
Policy gradients

Writing in terms of trajectories

Probability of a trajectory

\[ p(\tau; \theta) = \pi_\theta(a_0|s_0)p(s_1|s_0,a_0) \]
\[ \times \pi_\theta(a_1|s_1)p(s_2|s_1,a_1) \]
\[ \times \pi_\theta(a_2|s_2)p(s_3|s_2,a_2) \]
\[ \times ... \]
\[ = \prod_{t \geq 0} p(s_{t+1}|s_t,a_t)p_\theta(a_t|s_t) \]

Reward of a trajectory

\[ r(\tau) = \sum_{t \geq 0} \gamma^t r_t \]

\[ J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \]
Policy gradients

Formally, let’s define a class of parametrized policies: \( \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \)

For each policy, define its value:

\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]
\]
Policy gradients

Formally, let's define a class of parametrized policies: \( \Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\} \)

For each policy, define its value:

\[
J(\theta) = \mathbb{E}\left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]
\]

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

How can we do this?
Policy gradients

Formally, let’s define a class of parametrized policies: \( \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \)

For each policy, define its value:

\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]
\]

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

How can we do this?

Gradient ascent on policy parameters
Expected reward: \( J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \)

\[ = \int_{\tau} r(\tau)p(\tau; \theta) \, d\tau \]
REINFORCE algorithm

Expected reward: \[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \]
\[ = \int_{\tau} r(\tau) p(\tau; \theta) \, d\tau \]
\[ p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \]

Now let’s differentiate this: \( \nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau; \theta) \, d\tau \)

Intractable! Gradient of an expectation is problematic when \( p \) depends on \( \theta \)
REINFORCE algorithm

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

\[
p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)
\]

Intractable! Gradient of an expectation is problematic when \( p \) depends on \( \theta \)

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

Expected reward:  

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

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

\[ p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \]

Now let's differentiate this:  

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

Intractable! Gradient of an expectation is problematic when \( p \) depends on \( \theta \)

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:

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

\[ = \mathbb{E}_{\tau \sim p(\tau; \theta)} \left[ r(\tau) \nabla_\theta \log p(\tau; \theta) \right] \]

Tractable :-)}
**REINFORCE algorithm**

Can we compute these without knowing the transition probabilities?

We have: \( p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \)
REINFORCE algorithm

Can we compute these without knowing the transition probabilities?

We have: 
\[ p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t) \]

Thus: 
\[ \log p(\tau; \theta) = \sum_{t \geq 0} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \]
Can we compute these without knowing the transition probabilities?

We have: $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau; \theta) = \sum_{t \geq 0} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$

And when differentiating: $\nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn’t depend on transition probabilities!
REINFORCE algorithm

Can we compute these without knowing the transition probabilities?

We have: \[ p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \]

Thus: \[ \log p(\tau; \theta) = \sum_{t \geq 0} \left( \log p(s_{t+1} | s_t, a_t) + \log \pi_\theta(a_t | s_t) \right) \]

And when differentiating: \[ \nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t | s_t) \]

Doesn’t depend on transition probabilities!

Therefore when sampling a trajectory \( \tau \), we can estimate \( J(\theta) \) with

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} \left[ r(\tau) \nabla_\theta \log p(\tau; \theta) \right] \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t) \]
**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
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.

Pretend every action we took here was the correct label.\[\text{maximize: } \log p(y_i | x_i)\]

Pretend every action we took here was the wrong label.\[\text{maximize: } (-1) \cdot \log p(y_i | x_i)\]

$$\sum_i A_i \cdot \log p(y_i | x_i)$$
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

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization \( \pi(a|s, \theta), \forall a \in \mathcal{A}, s \in \mathcal{S}, \theta \in \mathbb{R}^n \)
Initialize policy weights \( \theta \)
Repeat forever:
  Generate an episode \( S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \), following \( \pi(\cdot|\cdot, \theta) \)
  For each step of the episode \( t = 0, \ldots, T - 1 \):
    \( G_t \leftarrow \text{return from step } t \)
    \( \theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \log \pi(A_t|S_t, \theta) \)
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

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization \( \pi(a|s, \theta), \forall a \in \mathcal{A}, s \in \mathcal{S}, \theta \in \mathbb{R}^n \)
Initialize policy weights \( \theta \)
Repeat forever:
  Generate an episode \( S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \)
  For each step of the episode \( t = 0, \ldots, T - 1 \):
    \( G_t \leftarrow \) return from step \( t \)
    \( \theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \log \pi(A_t|S_t, \theta) \)

following \( \pi(\cdot|\cdot, \theta) \)
epsilon greedy
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?
Variance reduction with a baseline

**Problem:** The raw value of a trajectory isn’t necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**What is important then?** Whether a reward is better or worse than what you expect to get
Variance reduction with a baseline

**Problem:** The raw value of a trajectory isn’t necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**What is important then?** Whether a reward is better or worse than what you expect to get

**Idea:** Introduce a baseline function dependent on the state. Concretely, estimator is now:

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

E.g. exponential moving average of the rewards. Provably reduces variance while remaining unbiased.
Actor-critic methods

A better baseline: Want to push up the probability of an action from a state, if this action was better than the expected value of what we should get from that state.

Q: What does this remind you of?
Actor-critic methods

A better baseline: Want to push up the probability of an action from a state, if this action was better than the expected value of what we should get from that state.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action $a_t$ in a state $s_t$ if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^\pi_\theta(s_t, a_t) - V^\pi_\theta(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$
Actor-critic methods

**Problem:** we don’t know Q and V. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).
Actor-critic methods

**Problem**: we don’t know $Q$ and $V$. Can we learn them?

**Yes**, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

Minh et. al., ICML 2016
Actor-critic methods

**Problem:** we don’t know Q and V. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

---

Minh et al., ICML 2016
Actor-critic methods

**Problem:** we don’t know Q and V. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

Critic: evaluates how good the action is

Actor: decides what actions to take

Minh et. al., ICML 2016
Actor-critic methods

**Problem:** we don’t know $Q$ and $V$. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

Critic: evaluates how good the action is

$$
L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i) \right)^2 \right]
$$

Actor: decides what actions to take

$$
\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_{\theta} \log \pi_\theta(a_t | s_t)
$$

Minh et. al., ICML 2016
Actor-critic methods

**Problem:** we don’t know $Q$ and $V$. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

![Diagram of A3C Policy Learning Module](image)

- Exploration + experience replay
- Decorrelate samples
- Fixed targets

Minh et. al., ICML 2016
Summary of RL methods

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learned Policy

- **Actor-Critic**
  - Learned Value Function
  - Learned Policy
Applications: RL and Language
RL and Language

Task-independent

[... ] having the correct
[... ] known lock and
[... ] unless the correct

key can open the lock [...]
key device was discovered [...]
key is inserted [...]

Pre-training

$V_{key}$ $V_{skull}$ $V_{ladder}$ $V_{rope}$

Pre-trained

Agent

Action

State, Reward

Environment

Luketina et. al., IJCAI 2019
RL and Language

Task-independent

[...] having the correct
[...] known lock and
[...] unless the correct

**key** can open the lock [...]
**key** device was discovered [...]
**key** is inserted [...]

Task-dependent

**Language-assisted**

**Key** Opens a door of the same color as the key.

**Skull** They come in two varieties, rolling skulls and bouncing skulls ... you must jump over rolling skulls and walk under bouncing skulls.

**Language-conditional**

Go down the ladder and walk right immediately to avoid falling off the conveyor belt, jump to the yellow rope and again to the platform on the right.

---

Luketina et. al., IJCAI 2019
Language-conditional RL

- Instruction following
- Rewards from instructions
- Language in the observation and action space
Language-conditional RL: Instruction following

- Navigation via instruction following

Train:
- Go to the short red torch
- Go to the blue keycard
- Go to the largest yellow object
- Go to the green object

Test:
- Go to the tall green torch
- Go to the red keycard
- Go to the smallest blue object

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017
Language-conditional RL: Instruction following

- Navigation via instruction following

**Train**
- Go to the short red torch
- Go to the blue keycard
- Go to the largest yellow object
- Go to the green object

**Test**
- Go to the tall green torch
- Go to the red keycard
- Go to the smallest blue object

**Fusion Alignment**
- Ground language
- Recognize objects
- Navigate to objects
- Generalize to unseen objects

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017
Language-conditional RL: Instruction following

- Interaction with the environment

Chaplot et. al., AAAI 2018
Language-conditional RL: Instruction following

- Gated attention via element-wise product

Fusion
Alignment
Ground language
Recognize objects

Chaplot et al., AAAI 2018
Language-conditional RL: Instruction following

- **Policy learning**
  - Asynchronous Advantage Actor-Critic (A3C) (Mnih et al.)
    - uses a deep neural network to parametrize the policy and value functions and runs multiple parallel threads to update the network parameters.
    - use **entropy regularization** for improved exploration
    - use **Generalized Advantage Estimator** to reduce the variance of the policy gradient updates (Schulman et al.)

Chaplot et al., AAAI 2018
Language-conditional RL: Instruction following

Chaplot et. al., AAAI 2018
Language-conditional RL: Instruction following

Grounding is important for generalization
blue armor, red pillar -> blue pillar

Chaplot et. al., AAAI 2018
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Encourages agent to explore its environment by maximizing **curiosity**. How well can I predict my environment?
1. Less training data
2. Stochastic
3. Unknown dynamics
So I should explore more.

Pathak et. al., ICML 2017
Burda et. al., ICLR 2019
Language-conditional RL: Rewards from instructions

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Natural language for reward shaping

“Jump over the skull while going to the left”
from Amazon Mturk :-( asked annotators to play the game and describe entities

Intermediate rewards to speed up learning
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Natural language for reward shaping

Encourages agent to take actions related to the instructions

Goyal et. al., IJCAI 2019
Language-conditional RL: Rewards from instructions

- Natural language for reward shaping
- Encourages agent to take actions related to the instructions

Montezuma’s revenge

Goyal et al., IJCAI 2019
Language-conditional RL: Language in S and A

- Embodied QA: Navigation + QA

Most methods similar to instruction following

Das et. al., CVPR 2018
Language-assisted RL

- Language for communicating domain knowledge
- Language for structuring policies
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Narasimhan et. al., JAIR 2018

from Amazon Mturk :-( asked annotators to play the game and describe entities
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Narasimhan et al., JAIR 2018
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Narasimhan et. al., JAIR 2018
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Grounded language learning
Helps to ground the meaning of text to the dynamics, transitions, and rewards
Language helps in multi-task learning and transfer learning

Narasimhan et. al., JAIR 2018
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Figure 1: An excerpt from the user manual of the game Civilization II.

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city?

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose relevant sentences
2. Label words into action-description, state-description, or background

Map tile attributes:
- Terrain type (e.g., grassland, mountain, etc)
- Tile resources (e.g., wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city?

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

The natural resources available where a population settles affects its ability to produce food and goods. **Build your city on a plains or grassland square with a river running through it if possible.**

**Map tile attributes:**
- Terrain type (e.g., grassland, mountain, etc)
- Tile resources (e.g., wheat, coal, wildlife, etc)

**City attributes:**
- City population
- Amount of food produced

**Unit attributes:**
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city?

---

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

Phalanxes are twice as effective at defending cities as warriors. ✔
Build the city on plains or grassland with a river running through it. ✔
You can rename the city if you like, but we'll refer to it as washington.
There are many different strategies dictating the order in which advances are researched.

After the **road is built**, use the **settlements** to start **improving** the **terrain**.

When the **settlements** become **active**, choose **build road**.

Use **settlements** or **engineers** to **improve** a **terrain square** within the **city radius**.

Relevant sentences

A: action-description
S: state-description

Branavan et al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

<table>
<thead>
<tr>
<th>Method</th>
<th>% Win</th>
<th>% Loss</th>
<th>Std. Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0</td>
<td>100</td>
<td>—</td>
</tr>
<tr>
<td>Built-in AI</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>Game only</td>
<td>17.3</td>
<td>5.3</td>
<td>± 2.7</td>
</tr>
<tr>
<td>Sentence relevance</td>
<td>46.7</td>
<td>2.8</td>
<td>± 3.5</td>
</tr>
<tr>
<td><strong>Full model</strong></td>
<td>53.7</td>
<td>5.9</td>
<td>± 3.5</td>
</tr>
<tr>
<td>Random text</td>
<td>40.3</td>
<td>4.3</td>
<td>± 3.4</td>
</tr>
<tr>
<td>Latent variable</td>
<td>26.1</td>
<td>3.7</td>
<td>± 3.1</td>
</tr>
</tbody>
</table>

Grounded language learning

Ground the meaning of text to the dynamics, transitions, and rewards

Language helps in learning

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

Language is most important at the start when you don’t have a good policy.
Afterwards, the model relies on game features.

Branavan et. al., JAIR 2012
Language for structuring policies

- Composing modules for Embodied QA

Das et. al., CoRL 2018
Language for structuring policies

- Composing modules for Embodied QA
Summary of RL methods

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- Policy Based
  - No Value Function
  - Learned Policy

- Actor-Critic
  - Learned Value Function
  - Learned Policy
Summary of applications

Instruction following

Train
- Go to the short red torch
- Go to the blue keycard
- Go to the largest yellow object
- Go to the green object

Test
- Go to the tall green torch
- Go to the red keycard
- Go to the smallest blue object

Go to the green torch

Language for rewards

“Jump over the skull while going to the left”
Summary of applications

Instruction following

Go to the green torch

Train
Go to the short red torch
Go to the blue keycard
Go to the largest yellow object
Go to the green object

Test
Go to the tall green torch
Go to the red keycard
Go to the smallest blue object

Language for rewards

“Jump over the skull while going to the left”

Language as domain knowledge

Language to structure policies

Q: What color is the sofa in the living room?

Exit-room  Find-room[living]  Find-object[sofa]  Answer