Instructions for the **midterm presentations** are on piazza resources:

- [https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kg742kik5kv6tg](https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kg742kik5kv6tg)
- Deadline for pre-recorded presentation: Friday, November 13th, 2020 at 8pm ET
- 7 minutes, mostly about error analysis and updated ideas, don't try to present everything...

Instructions for **midterm report** are also online:

- [https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kgraer741fw3n4](https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kgraer741fw3n4)
- Deadline: Sunday, November 15th, 2020
- 8 pages for teams of 3 and 9 pages for the other teams
- Multimodal baselines, error analysis, proposed ideas

Start working on midterm assignments now!
Admin

Reading wildcard
- Each student gets one (1) wild card to be used as a way to extend by up to 24 hours their deadline for the summary deadline (which is usually Fridays at 8pm)
- See details on piazza
Admin

Piazza live Q&A

Please share your questions and comments on Piazza Live Q&A

Live responses by your TAs and follow-up by the instructor after the main lecture
Used Materials

Acknowledgement: Some 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, 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:
$S_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$
We aim to maximize 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
**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 its policy as a result of experience

Special case: deterministic policies
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 $H$

Goal:
$$
\arg \max_{\pi} \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t R_t \mid \pi \right]
$$

Return:
$$
G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
$$
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 optimality equations**
  - \( Q^*(s, a) \) \quad Q-value iteration
  - \( V^*(s) \) \quad Value iteration

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

Fully known MDP states transitions rewards

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

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

- Bellman optimality equations
  - \( Q^*(s, a) \)
  - \( V^*(s) \)
- Bellman expectation equations
  - \( Q^\pi(s, a) \)
  - \( V^\pi(s) \)

Q-value iteration
Value iteration
Q-policy iteration
Policy iteration

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 → Bellman optimality equations → Replace true expectation over transitions with estimates

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

Tabular Q-learning

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

simulation and exploration, epsilon greedy is important!

old estimate \quad target
Recap: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates

Tabular Q-learning

\[ s' \sim P(s'|s, a) \quad \text{simulation and exploration, epsilon greedy is important!} \]

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

old estimate \quad 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 \hspace{10cm} target

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

Q-learning target \hspace{5cm} Q-network

Correlated samples + non-stationary targets
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

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

Q-learning target

Q-network

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: Language and RL
- Applications: RL for language (e.g. text generation)
Value-based and Policy-based RL

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

**State value functions**

\[ V^\pi(s) \]
\[ V^*(s) \]

**Action value functions**

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

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

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

Slides from Fragkiadaki
Value-based and Policy-based RL

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

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

- **Actor-Critic**
  - Learned Value Function
  - Learned Policy

Slides from Fragkiadaki
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

Slides from Fragkiadaki
Pong from pixels
Pong from pixels

e.g.,

height  width

[80 x 80]
array of
Pong from pixels

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)
...

Slides from Karpathy
Pong from pixels

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

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

\[\ldots\]

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

maximize:

Slides from Karpathy
Pong from pixels

Except, we don’t have labels...

Should we go UP or DOWN?

Slides from Karpathy
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.

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

Pretend every action we took here was the wrong label.

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

$\pi(a | s)$
Pong from pixels

\[ \pi(a \mid s) \]

1. Initialize a policy network at random
2. **Repeat Forever:**
3. Collect a bunch of rollouts with the policy **epsilon greedy!**
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

\[ \pi(a | s) \]

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.

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

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

\[ \pi(a \mid s) \]

- 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: \[ (-1) \times \log p(y_i \mid x_i) \]

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

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

\[ \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \]

For each policy, define its value:

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

Writing in terms of trajectories

\[ \tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...) \]

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 \( \tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots) \)

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

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

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

Formally, let’s define a class of parameterized policies

\[ \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \]

For each policy, define its value:

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

We want to find the optimal policy

\[ \theta^* = \arg \max_\theta J(\theta) \]

How can we do this?

Gradient ascent on policy parameters
REINFORCE algorithm

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

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

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

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

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

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:

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

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

Tractable :-(
Can we compute these without knowing the transition probabilities?

We have:

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

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

Therefore when sampling a trajectory, we can estimate gradients:

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)] \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t) \]
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 \( r(\text{trajectory}) \) is high, push up the probabilities of the actions seen
- If \( r(\text{trajectory}) \) 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 \( r(\text{trajectory}) \) is high, push up the probabilities of the actions seen
- If \( r(\text{trajectory}) \) 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.

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

\[ \sum_i A_i \times \log p(y_i | x_i) \]
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 $r(\text{trajectory})$ is high, push up the probabilities of the actions seen
- If $r(\text{trajectory})$ is low, push down the probabilities of the actions seen

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

Input: a differentiable policy parameterization $\pi(a|s, \theta), \forall a \in A, s \in S, \theta \in \mathbb{R}^n$
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 $r(\text{trajectory})$ is high, push up the probabilities of the actions seen
- If $r(\text{trajectory})$ 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 \( r(\text{trajectory}) \) is high, push up the probabilities of the actions seen
- If \( r(\text{trajectory}) \) 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 this estimator?
Variance reduction with a baseline

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

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

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

**What is important then?** Whether a reward is higher or lower than what you expect to get.

**Idea:** Introduce a baseline function dependent on the state, which gives us an estimator:

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

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

A better baseline: want to push 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.

Recall: $Q$ and $V$ - action value and state value functions!
Actor-critic methods

A better baseline: want to push 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

Recall: Q and V - action value and state value functions!

We are happy with an action $a$ in a state $s$ if $Q(s,a) - V(s)$ is large. Otherwise 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)

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

- **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

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

Actor: decides what actions to take

$$\pi_{\theta}(a \mid s)$$

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

Critic: evaluates how good the action is

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

Q-learning target

Q-network

Actor: decides what actions to take

$$
\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( Q^\pi(\mathbf{s}_t, a_t) - V^\pi(\mathbf{s}_t) \right) \nabla_\theta \log \pi_\theta(a_t | \mathbf{s}_t)
$$

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).

**Critic:** evaluates how good the action is

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

**Actor:** decides what actions to take

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

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

- **Value Based**
  - Value iteration
  - Policy iteration
  - (Deep) Q-learning
    - Learned Value Function
    - Implicit policy (e.g. $\epsilon$-greedy)

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

- **Actor-Critic**
  - Actor (policy)
  - Critic (Q-values)
    - Learned Value Function
    - Learned Policy
Applications: Stochastic optimization
Stochastic Optimization

$$\max_{\phi} E_{q_{\phi}(z)}[f(z)]$$
Stochastic Optimization

\[ \max_{\theta, \phi} \mathcal{L}(x; \theta, \phi) \qquad \text{Evidence lower bound} \]

\[ \max_{\theta, \phi} E_{q_{\phi}(z|x)}[\log p(x|z; \theta)] - D_{KL}(q_{\phi}(z|x)||p(z)) \]

\[ \max_{\theta, \phi} E_{q_{\phi}(z|x)}[\log p(x|z; \theta)] \]

**VAEs**

Figure courtesy: Kingma & Welling, 2014
Stochastic Optimization

$$\max_{\phi} E_{q_\phi(z)}[f(z)]$$

**Evidence lower bound**

$$\max_{\theta, \phi} \mathcal{L}(x; \theta, \phi)$$

$$\max_{\theta, \phi} E_{q_\phi(z|x)}[\log p(x|z; \theta)] - D_{KL}(q_\phi(z|x)||p(z))$$

$$\max_{\theta, \phi} E_{q_\phi(z|x)}[\log p(x|z; \theta)]$$

*Solve by reparameterization!*

**VAEs**

Figure courtesy: Kingma & Welling, 2014
Stochastic Optimization

\[ \max_{\theta, \phi} \mathcal{L}(x; \theta, \phi) \quad \text{Evidence lower bound} \]

\[ \max_{\theta, \phi} E_{q_\phi(z|x)}[\log p(x|z; \theta)] \]

Solve by reparameterization!

We require that:
- \( z \) is continuous
- \( q(z) \) is reparameterizable
- \( f(z) \) is differentiable wrt \( \phi \)

- Sample \( z \sim q_\phi(z) \)
- Sample \( \epsilon \sim \mathcal{N}(0, 1) \), \( z = \mu + \sigma \epsilon \)
Stochastic Optimization

**VAEs**

\[
\max_{\theta, \phi} \mathcal{L}(x; \theta, \phi) \quad \text{Evidence lower bound}
\]

\[
\max_{\theta, \phi} E_{q_\phi(z|x)}[\log p(x|z; \theta)]
\]

Solve by reparameterization!

- \( z \) is continuous
- \( q(z) \) is reparameterizable
- \( f(z) \) is differentiable wrt \( \phi \)

- Sample \( z \sim q_\phi(z) \)
- Sample \( \epsilon \sim \mathcal{N}(0, 1) \), \( z = \mu + \sigma \epsilon \)

**RL**

\[
\max_\phi E_{q_\phi(z)}[f(z)]
\]

\[
\max_\phi J(\phi) \quad \text{Reward}
\]

\[
\max_\phi E_{\tau \sim p(\tau; \phi)}[r(\tau)]
\]
Stochastic Optimization

**VAEs**

\[
\begin{align*}
\max_{\theta, \phi} & \mathcal{L}(x; \theta, \phi) \quad \text{Evidence lower bound} \\
\max_{\theta, \phi} & E_{q_\phi(z|x)}[\log p(x|z; \theta)]
\end{align*}
\]

Solve by reparameterization!

- \( z \) is continuous
- \( q(z) \) is reparameterizable
- \( f(z) \) is differentiable wrt \( \phi \)

**RL**

\[
\begin{align*}
\max_{\phi} & E_{q_\phi(z)}[f(z)] \\
\max_{\phi} & J(\phi) \quad \text{Reward} \\
\max_{\phi} & E_{\tau \sim p(\tau; \phi)}[r(\tau)]
\end{align*}
\]

Reparameterization???

In RL (at least for discrete actions):
- \( T \) is a sequence of discrete actions
- \( p(T; \phi) \) is not reparameterizable
- \( r(T) \) is a black box function
  i.e. the environment

\[\begin{align*}
p(x|z; \theta) & \\
z & \\
q_\phi(z|x) & \quad \text{Latent distribution} \\
x & \\
\text{Sample } z & \sim q_\phi(z) \\
\text{Sample } \epsilon & \sim \mathcal{N}(0, I), \ z = \mu + \sigma \epsilon
\end{align*}\]
**Stochastic Optimization**

**VAEs**

\[
\max_{\theta, \phi} \mathcal{L}(x; \theta, \phi) \quad \text{Evidence lower bound}
\]

\[
\max_{\theta, \phi} E_{q_\phi(z|x)}[\log p(x|z; \theta)]
\]

Solve by reparameterization!

- \(q(z)\) is reparameterizable
- \(f(z)\) is differentiable wrt \(\phi\)

**RL**

\[
\max_{\phi} E_{q_\phi(z)}[f(z)]
\]

\[
\max_{\phi} J(\phi) \quad \text{Reward}
\]

Reparameterization???

- \(z\) is continuous
- \(\pi_\phi(a|s)\)
- Sample \(z \sim q_\phi(z)\)
- Sample \(\epsilon \sim \mathcal{N}(0, I)\), \(z = \mu + \sigma \epsilon\)

In RL (at least for discrete actions):
- \(T\) is a sequence of discrete actions
- \(p(T; \phi)\) is not reparameterizable
- \(r(T)\) is a black box function
  - i.e. the environment

**REINFORCE is a general-purpose solution!**
Revisiting REINFORCE

$$\max_{\phi} E_{q_{\phi}(z)}[f(z)]$$

We want to take gradients wrt $\phi$ of the term:

$$\nabla_{\phi} E_{q_{\phi}(z)}[f(z)] = E_{q_{\phi}(z)}[f(z)\nabla_{\phi} \log q_{\phi}(z)]$$
Revisiting REINFORCE

\[ \max_{\phi} E_{q_{\phi}(z)}[f(z)] \]

We want to take gradients wrt \( \phi \) of the term:

\[ \nabla_{\phi} E_{q_{\phi}(z)}[f(z)] = E_{q_{\phi}(z)}[f(z)\nabla_{\phi} \log q_{\phi}(z)] \]

We can now compute a Monte Carlo estimate:

Sample \( z^1, \cdots, z^K \) from \( q_{\phi}(z) \) and estimate

\[ \nabla_{\phi} E_{q_{\phi}(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k) \nabla_{\phi} \log q_{\phi}(z^k) \]
Revisiting REINFORCE

$$\max_{\phi} E_{q_\phi(z)}[f(z)]$$

We want to take gradients wrt $\phi$ of the term:

$$\nabla_\phi E_{q_\phi(z)}[f(z)] = E_{q_\phi(z)}[f(z)\nabla_\phi \log q_\phi(z)]$$

We can now compute a Monte Carlo estimate:

Sample $z^1, \cdots, z^K$ from $q_\phi(z)$ and estimate

$$\nabla_\phi E_{q_\phi(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k)\nabla_\phi \log q_\phi(z^k)$$

What we derived. Sample trajectories and compute:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$
We can now compute a Monte Carlo estimate:

\[
\max_\phi E_{q_\phi(z)}[f(z)]
\]

We just need the distribution \(q()\) to allow for easy sampling.

- \(z\) can be discrete or continuous!
- \(q(z)\) can be a discrete and continuous distribution! (but must be differentiable wrt \(\phi\))
- \(f(z)\) can be a black box!
Applications: Text generation

GANs for text generation
1. Text data is discrete
   ○ Discriminator gradient does not exist for samples from categorical distribution
   ○ Gradient sparse due to large dictionary size
Applications: Text generation

GANs for text generation
1. Text data is discrete
   ○ Discriminator gradient does not exist for samples from categorical distribution
   ○ Gradient sparse due to large dictionary size

More efficient search strategy for most likely sentence

[Yu et. al., AAAI 2017]
Applications: Text generation

GANs for text generation

1. Text data is discrete
   - Discriminator gradient does not exist for samples from categorical distribution
   - Gradient sparse due to large dictionary size

More efficient search strategy for most likely sentence

Training generator:
After sampling all words using Monte Carlo search, compute reward for generator based on discriminator feedback
- if similar to real text, high reward
- if different from real text, low reward

\[
\nabla_\phi E_{q_\phi(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k)\nabla_\phi \log q_\phi(z^k)
\]

disc reward

[Yu et. al., AAAI 2017]
Applications: Text generation

GANs for text generation

1. Text data is discrete
   - Discriminator gradient does not exist for samples from categorical distribution
   - Gradient sparse due to large dictionary size

More efficient search strategy for most likely sentence

Training generator:
After sampling all words using Monte Carlo search, compute reward for generator based on discriminator feedback
- if similar to real text, high reward
- if different from real text, low reward

Training discriminator is easy - all params are differentiable

[Yu et. al., AAAI 2017]
Applications: Text generation

GANs for text generation

1. Text is sensitive to noise (small disturbances easily alters the meaning of text)
2. Sparse discriminator feedback (feedback only makes sense on full sentences)

+ Instead of discriminator reward after whole sentence, use a recurrent discriminator which compares generated vs real prefixes to give dense rewards at all time steps
+ large variance from REINFORCE: use large batch sizes and subtract baseline (moving average of rewards)
+ other tricks, see paper

[d’Autume et. al., NeurIPS 2019]
Applications: Text generation

GANs for text generation

1. Text is sensitive to noise (small disturbances easily alters the meaning of text)
2. Sparse discriminator feedback (feedback only makes sense on full sentences)

Other approaches as well without using policy gradients
- discriminator directly comparing in logit space
- use Gumbel softmax (Jang et al. 2016)
- see more in
  https://www.cl.uni-heidelberg.de/statnlpgroup/blog/rl4nmt/
  and
  https://deepgenerativemodels.github.io/assets/slides/cs236_lecture15.pdf

[d’Autume et. al., NeurIPS 2019]
Applications: Dialog generation

GANs for dialog generation and evaluation

Model Breakdown

Generative Model (G)

Encoding

I'm

fine

Decoding

EOS

Discriminative Model (D)

P = 90% human generated

Sample:
Input message
Response 1
Response 2
...
Response K

Define rewards:
1. Ease of answering
2. Information flow
3. Meaningfulness
4. Discriminator wrt human dialog

Sample $z^1, \cdots, z^K$ from $q_{\phi}(z)$ and estimate

$$\nabla_{\phi} E_{q_{\phi}(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k) \nabla_{\phi} \log q_{\phi}(z^k)$$
Applications: Optimizing general rewards

Instead of optimizing for cross-entropy (not final evaluation metric), optimize directly for the evaluation metric e.g. BLEU score

- BLEU score only defined on raw text after sampling from softmax
- Not differentiable through standard gradient methods.

Sample $z^1, \ldots, z^K$ from $q_\phi(z)$ and estimate

$$\nabla_\phi E_{q_\phi(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k) \nabla_\phi \log q_\phi(z^k)$$

[Ranzato et. al., ICLR 2016]
Slides from Wang, ACL 2018 tutorial
Applications: Hard attention

Hard attention ‘gates’ (0/1) rather than soft attention (softmax between 0-1)
- Can be seen as discrete layers in between differentiable neural net layers

[Xu et al., ICML 2015]
[Chen et al., ICMI 2017]
Applications: Hard attention

Hard attention ‘gates’ (0/1) rather than soft attention (softmax between 0-1)
- Can be seen as discrete layers in between differentiable neural net layers

Sentiment analysis, emotion recognition

Image captioning

Figure 3. Visualization of the attention for each generated word. The rough visualizations obtained by upsampling the attention weights and smoothing. (top)“soft” and (bottom) “hard” attention (note that both models generated the same captions in this example).

[Xu et. al., ICML 2015]
[Chen et al., ICMI 2017]
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**

\[ \text{V}_{\text{key}} \ \text{V}_{\text{skull}} \ \text{V}_{\text{ladder}} \ \text{V}_{\text{rope}} \]

**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 S and A
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.

from Amazon Mturk:-( asked annotators to play the game and describe entities

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

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
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

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

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?
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 **settlers** to **start improving** the **terrain**.
- When the **settlers** become **active**, chose **build road**.
- Use **settlers** 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><strong>53.7</strong></td>
<td><strong>5.9</strong></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

Das et. al., CoRL 2018
Summary of RL methods

- **Value Based**
  - Value iteration
  - Policy iteration
  - (Deep) Q-learning
  - Learned Value Function
  - Implicit policy (e.g. \( \varepsilon \)-greedy)

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

- **Actor-Critic**
  - Actor (policy)
  - Critic (Q-values)
  - Learned Value Function
  - Learned Policy
Summary of applications

Stochastic optimization

Sample $z^1, \ldots, z^K$ from $q_\phi(z)$ and estimate

$$\nabla_\phi E_{q_\phi(z)}[f(z)] \approx \frac{1}{K} \sum_k f(z^k) \nabla_\phi \log q_\phi(z^k)$$

disc reward

Text generation

Discrete layers

General reward functions

Reject  Pass  Reject
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

Language for rewards

- “Jump over the skull while going to the left”

Language as domain knowledge

- is an enemy who chases you
- is a stationary collectible
- is a randomly moving enemy
- is a stationary immovable wall

Language to structure policies

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

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