Carnegie Mellon University
Language Technologies Institute

Intro to Reinforcement Learning Part II

11-777 Multimodal Machine Learning Fall 2020

Paul Liang

pliang@cs.cmu.edu

@pliang279

Admin

Start working on midterm assignments now!

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

- 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...

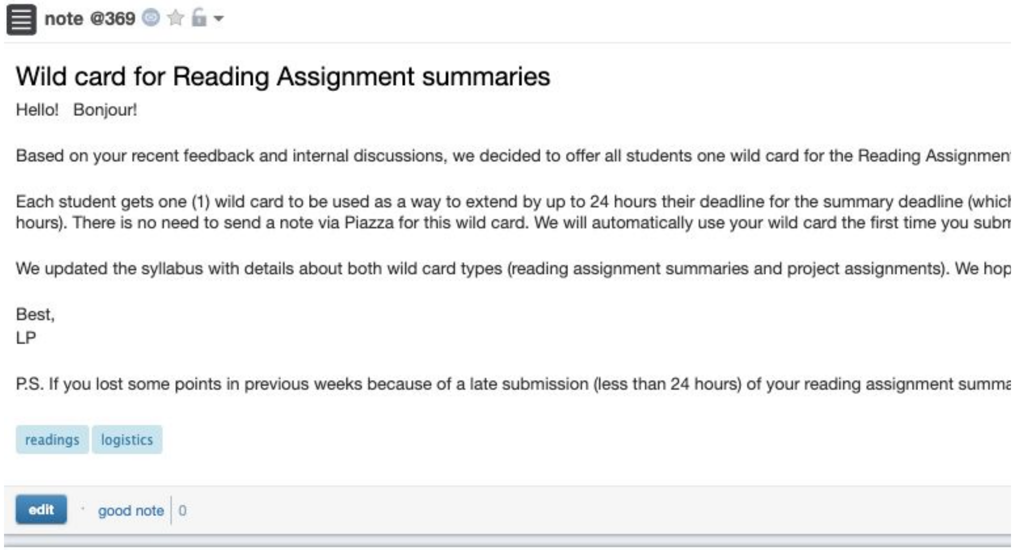
Instructions for **midterm report** are also online:

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

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



note @369

Wild card for Reading Assignment summaries

Hello! Bonjour!

Based on your recent feedback and internal discussions, we decided to offer all students one wild card for the Reading Assignmen

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 (whicl
hours). There is no need to send a note via Piazza for this wild card. We will automatically use your wild card the first time you subn

We updated the syllabus with details about both wild card types (reading assignment summaries and project assignments). We hop

Best,
LP

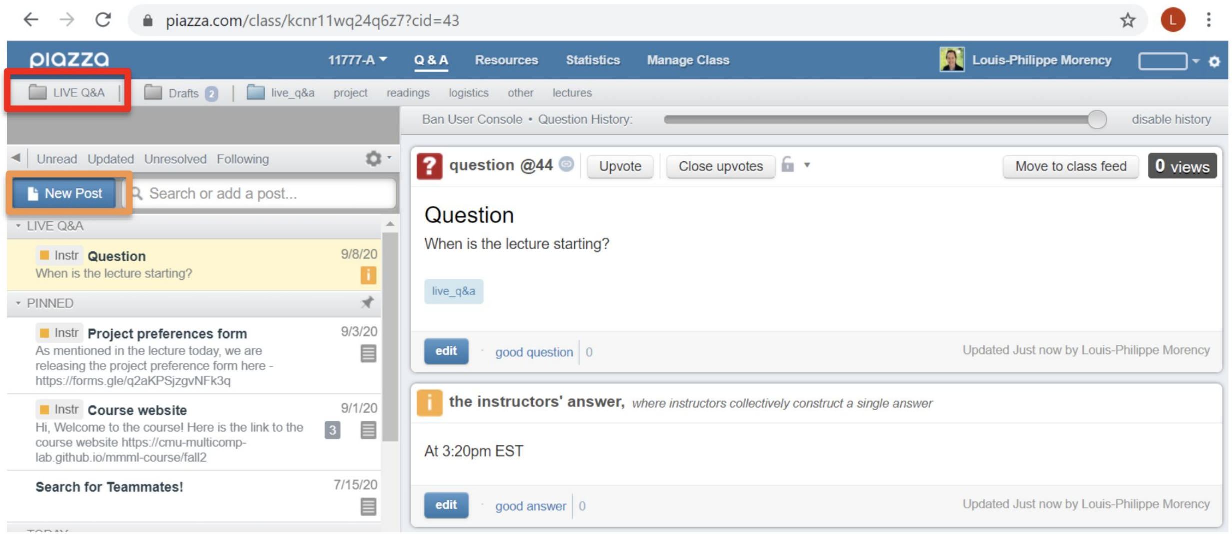
P.S. If you lost some points in previous weeks because of a late submission (less than 24 hours) of your reading assignment summa

readings logistics

edit good note 0

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

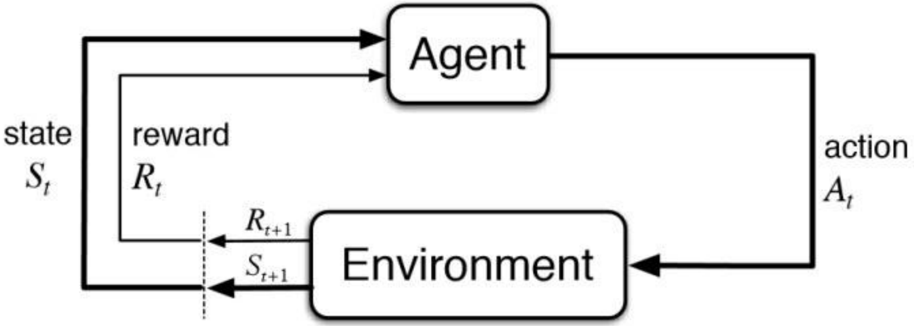
Used Materials

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 $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Trajectory

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

Slides from Fragkiadaki

Return

We aim to maximize *total discounted reward*:

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

Discount factor

γ close to 0 leads to "myopic" evaluation

γ close to 1 leads to "far-sighted" evaluation

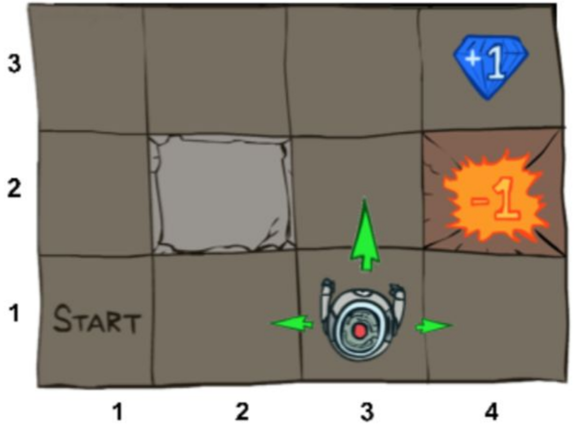
Policy

Definition: A policy is a distribution over actions given states

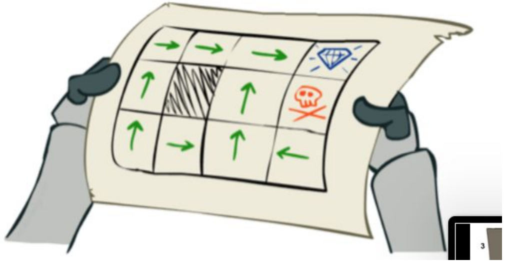
$$\pi(a \mid s) = \mathbf{Pr}(A_t = a \mid S_t = s), \forall t$$

- 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



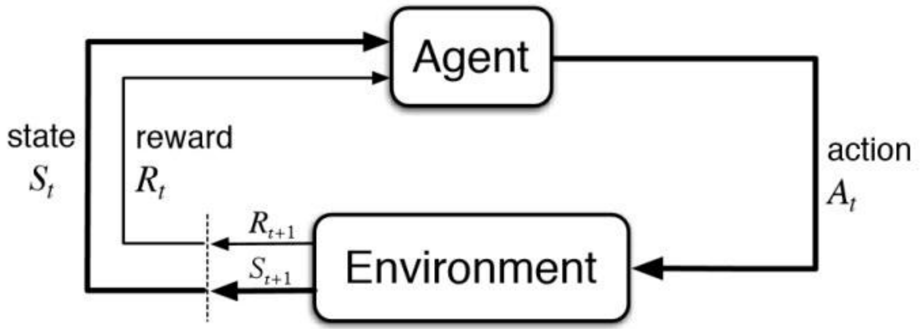
π:



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 γ
- Horizon H

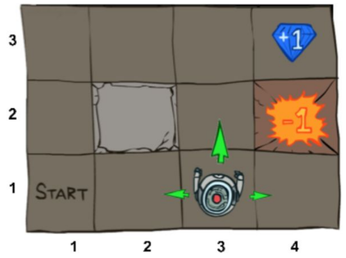


Return:

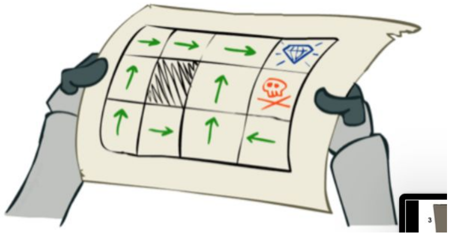
$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Goal:

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



π:



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

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

Fully known MDP
states
transitions
rewards

Bellman optimality equations

$Q^*(s,a)$	**Q-value iteration**
$V^*(s)$	**Value iteration**

Bellman expectation equations

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

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

Recap: Exact methods

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

Fully known MDP
states
transitions
rewards

Bellman optimality equations

$Q^*(s,a)$ — **Q-value iteration**

$V^*(s)$ — **Value iteration**

Bellman expectation equations

$Q^\pi(s,a)$ — **Q-policy iteration**

$V^\pi(s)$ — **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

Tabular Q-learning

$s' \sim P(s'|s, a)$ simulation and exploration, epsilon greedy is important!

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

Recap: Tabular Q-learning

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

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

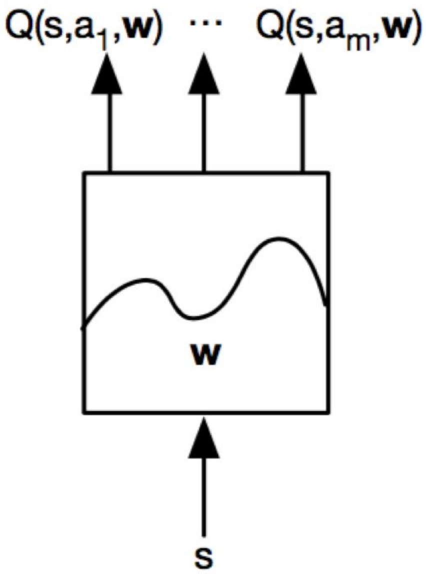
simulation and exploration, epsilon greedy is important!

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

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

Tabular: keep a |S| x |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



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

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

Correlated samples + non-stationary targets

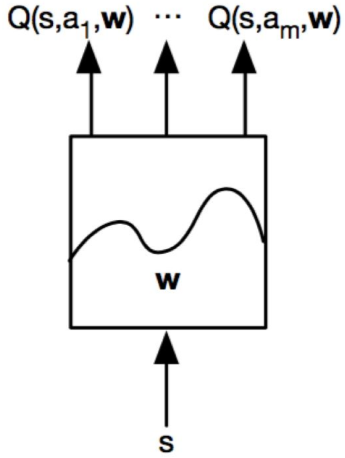
Recap: Deep Q-learning

- 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

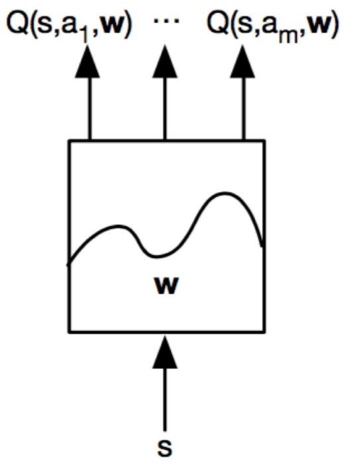
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

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



- Use stochastic gradient descent
- Update **w-** with updated **w** every ~1000 iterations

Recap: Deep Q-learning



$$\underbrace{Q^*(s,a)}_{\text{old estimate}} = \mathbb{E}_{s'}\Big[\underbrace{r(s,a,s') + \gamma \max_{a'} Q^*(s',a')}_{\text{target}}\Big]$$

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\Bigg[\Bigg(\underbrace{r + \gamma \max_{a'} Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\Bigg)^2\Bigg]$$

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)

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

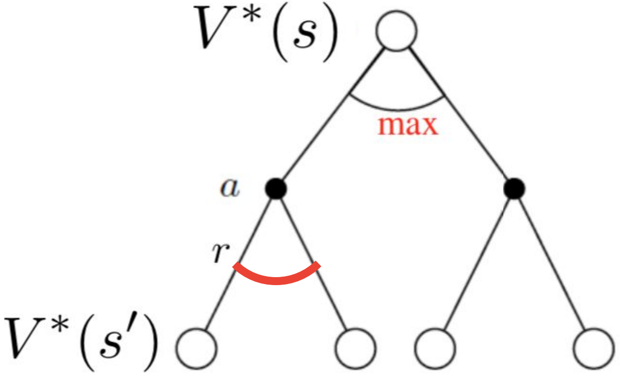
Recap: Obtaining the optimal policy

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. ε-greedy)

State value functions

$$V^{\pi}(s)$$

$$V^{*}(s)$$

Action value functions

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

$$Q^{*}(s,a)$$

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

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

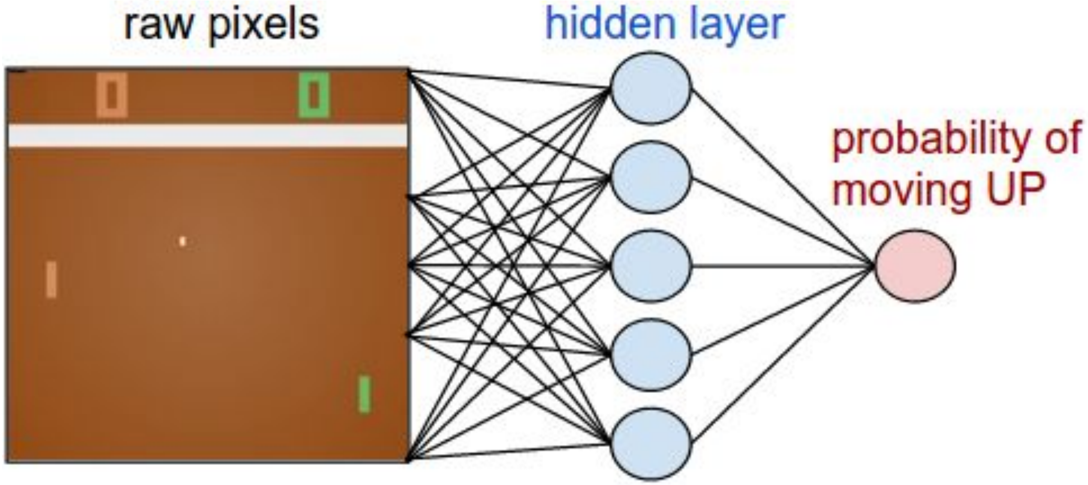
Slides from Fragkiadaki

Value-based and Policy-based RL

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

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

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



Slides from Fragkiadaki

Directly learning the policy

- Often π 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 π 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'

- 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*

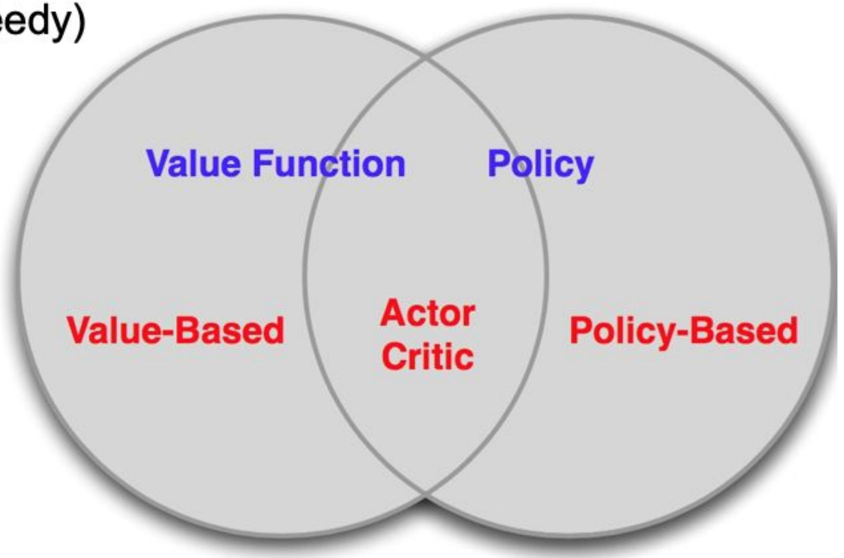
$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\ \epsilon, & \text{else} \end{cases}$$

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

Slides from Fragkiadaki

Value-based and Policy-based RL

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)
- Policy Based
  - No Value Function
  - Learned Policy
- Actor-Critic
  - Learned Value Function
  - Learned Policy



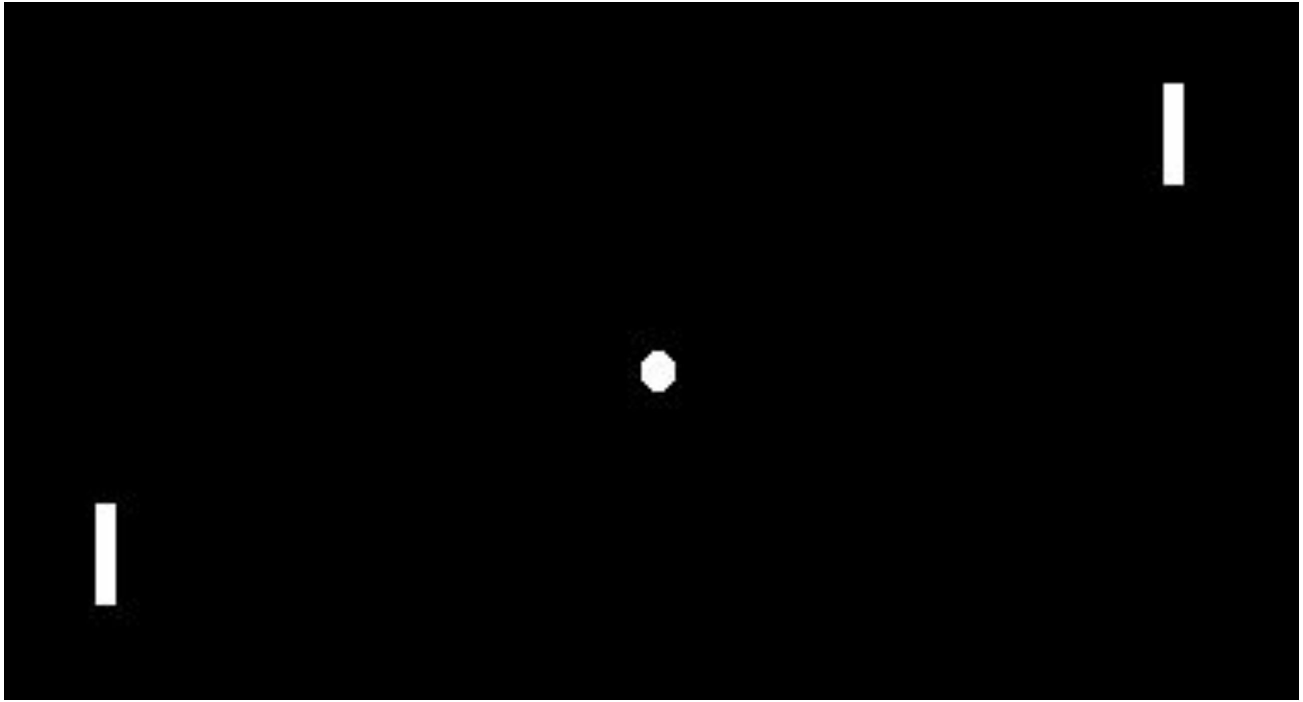
Slides from Fragkiadaki

Value-based and Policy-based RL

	Policy-based	Value-based
■ Conceptually:	Optimize what you care about	Indirect, exploit the problem structure, self-consistency
■ Empirically:	More compatible with rich architectures (including recurrence) More versatile More compatible with auxiliary objectives	More compatible with exploration and off-policy learning More sample-efficient when they work

Slides from Fragkiadaki

Pong from pixels



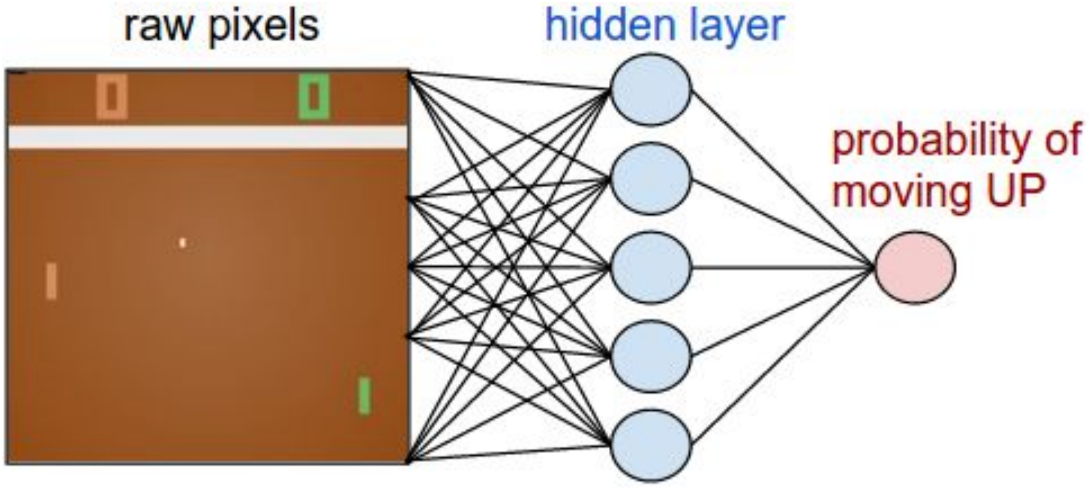
Slides from Karpathy

Pong from pixels

e.g.,

height width

[80 x 80]
array of

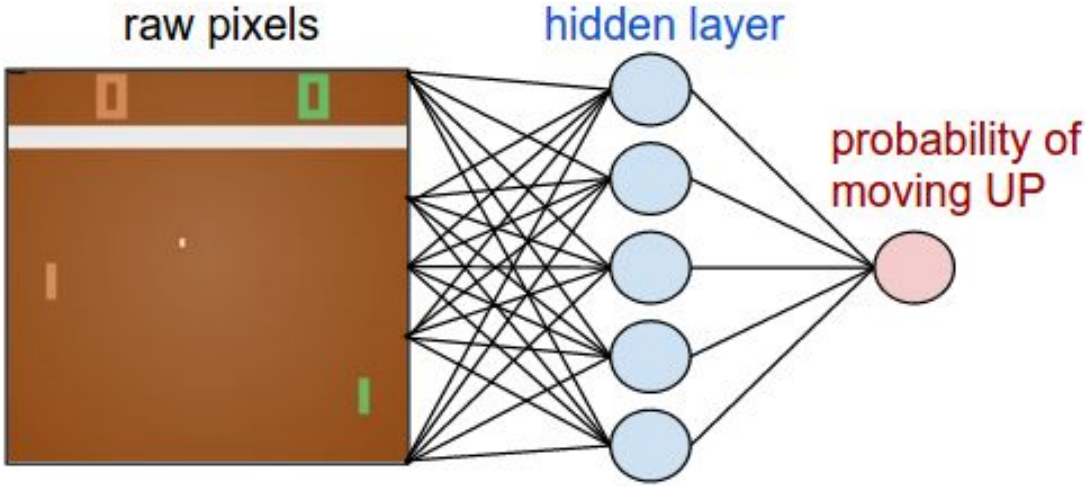


Pong from pixels

e.g.,

height width

[80 x 80]
array of



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

Slides from Karpathy

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

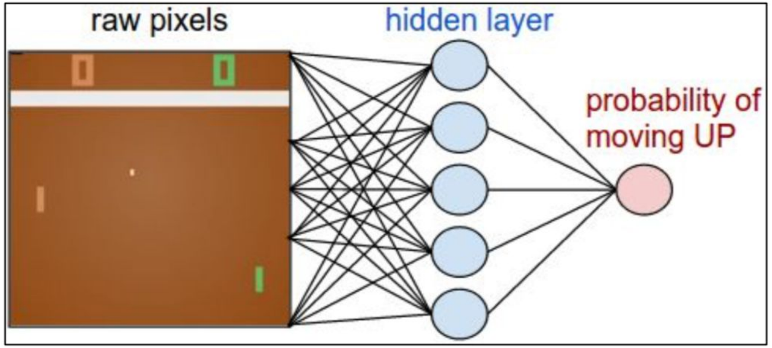
Pong from pixels

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

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

maximize:

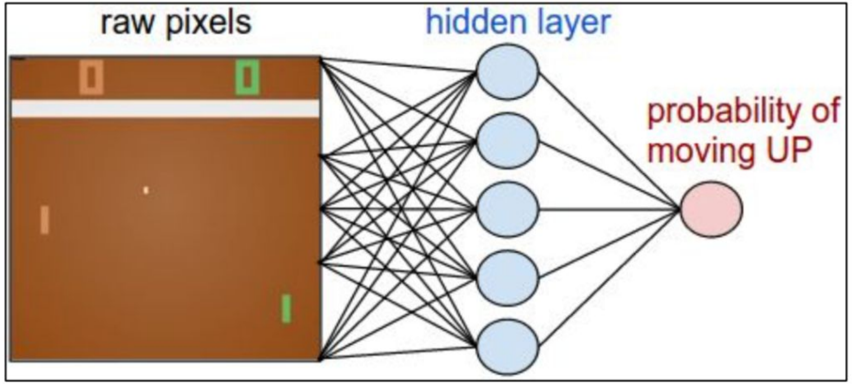
$$\sum_i \log p(y_i | x_i)$$



Slides from Karpathy

Pong from pixels

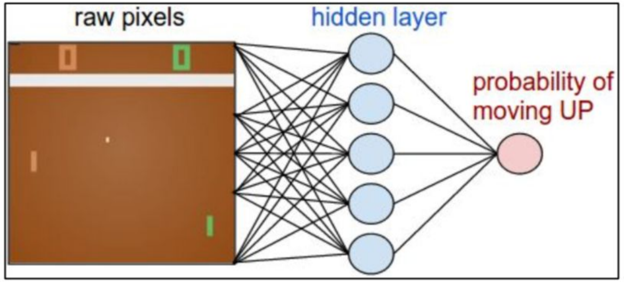
Except, we don't have labels...



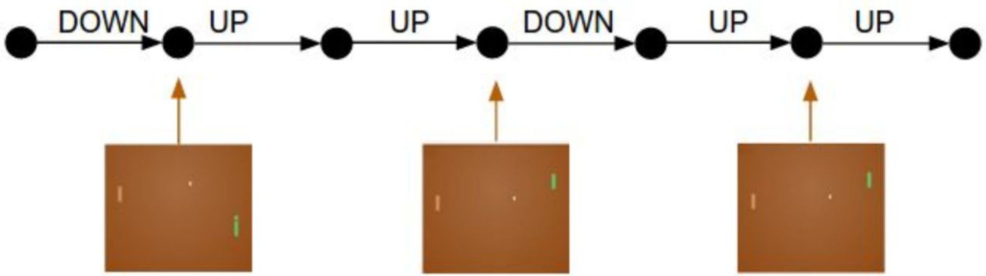
Should we go UP or DOWN?

Pong from pixels

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



Rollout the policy and collect an episode



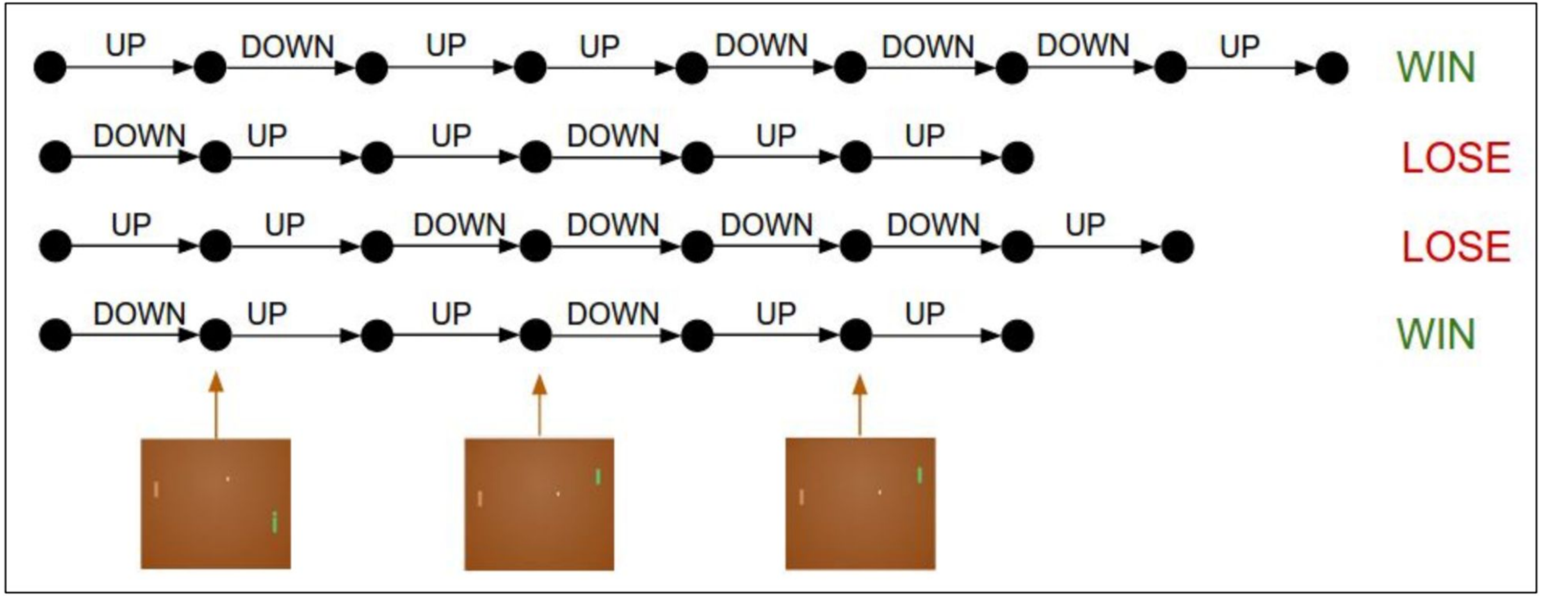
WIN

Slides from Karpathy

Pong from pixels

Collect many rollouts...

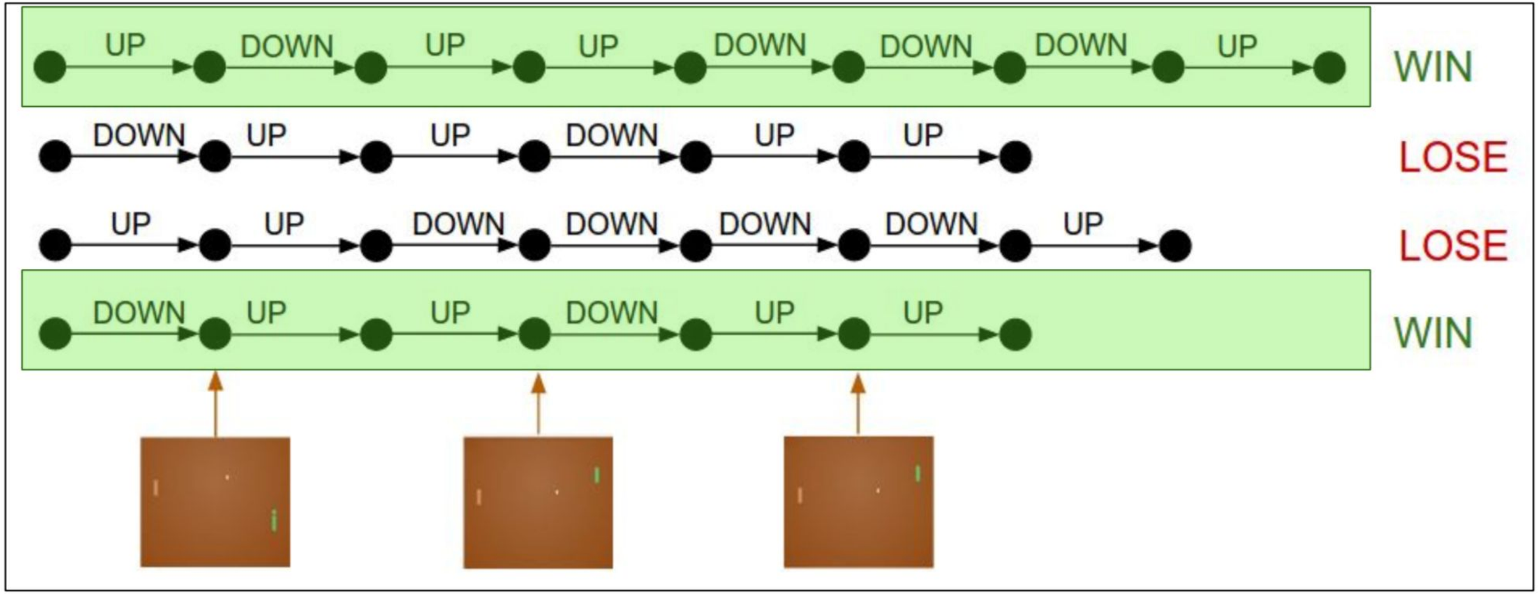
4 rollouts:



Slides from Karpathy

Pong from pixels

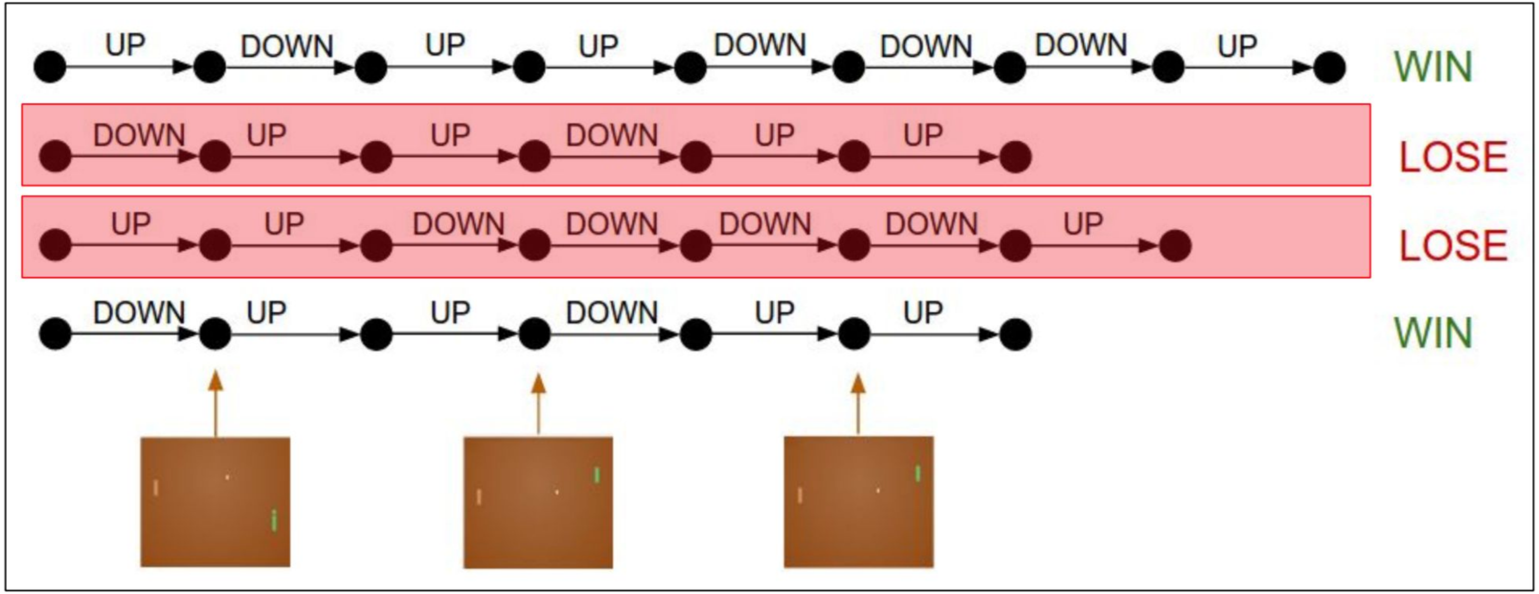
Not sure whatever we did here, but apparently it was good.



Slides from Karpathy

Pong from pixels

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



Slides from Karpathy

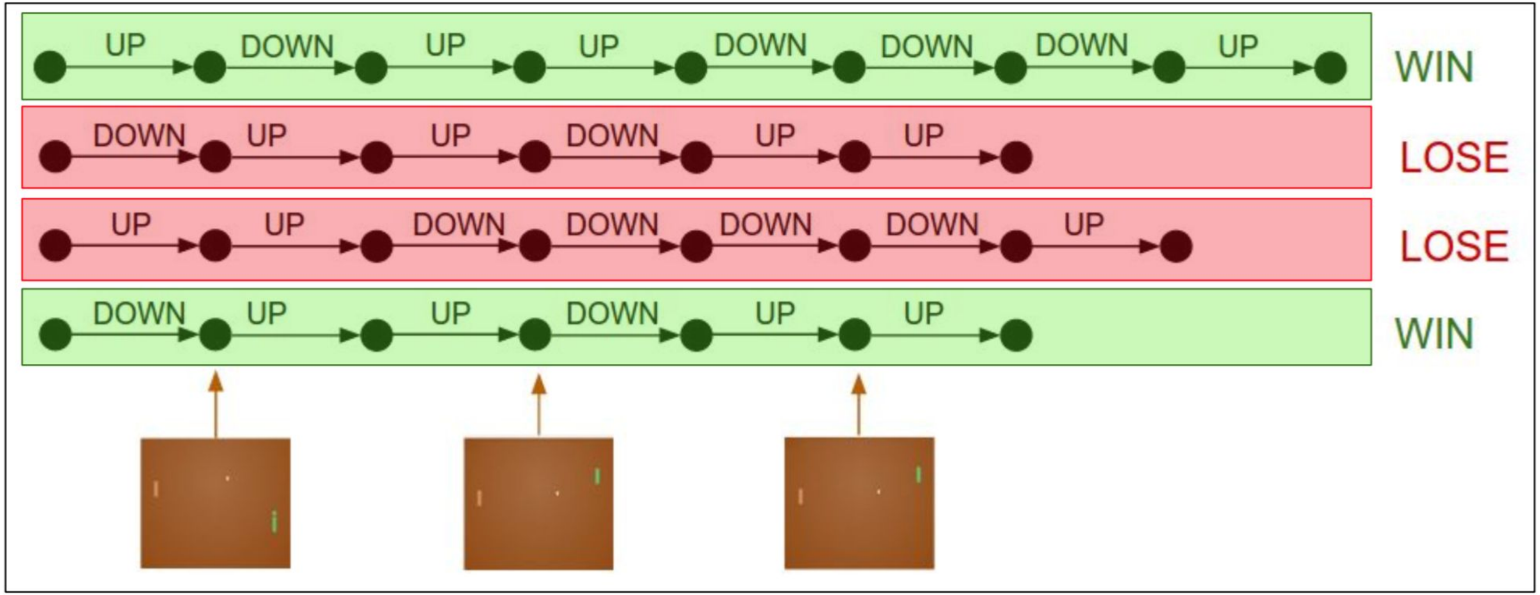
Pong from pixels

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: $(-1) * \log p(y_i \mid x_i)$



Slides from Karpathy

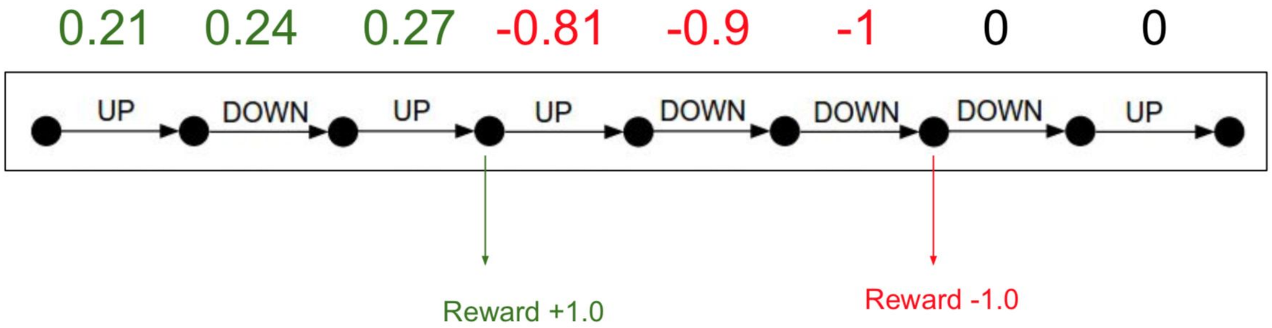
Pong from pixels

Discounting

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

$$\boxed{\sum_i \boxed{A_i} * \log p(y_i | x_i)}$$

Discounted rewards



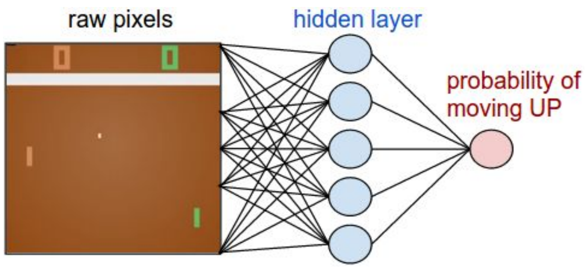
\gamma = 0.9

Slides from Karpathy

Pong from pixels

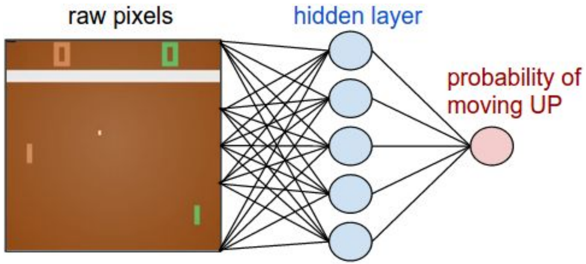
$$\pi(a \mid s)$$

1. `Initialize a policy network at random`



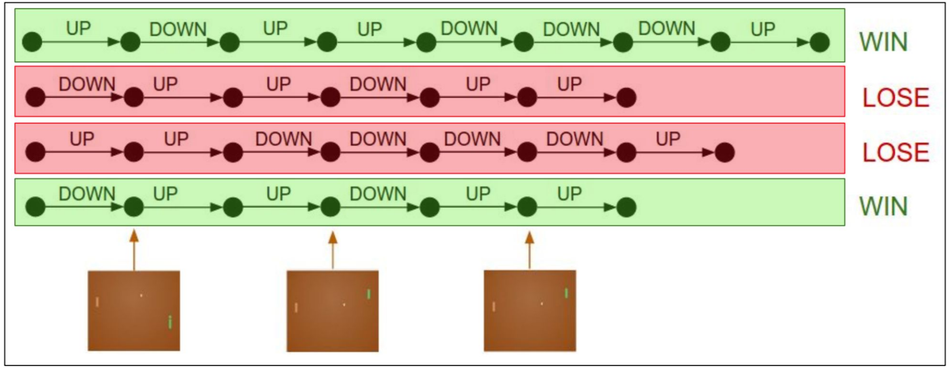
Pong from pixels

$\pi(a\,|\,s)$




```
1.   Initialize a policy network at random
2.   Repeat Forever:
3.       Collect a bunch of rollouts with the policy
```

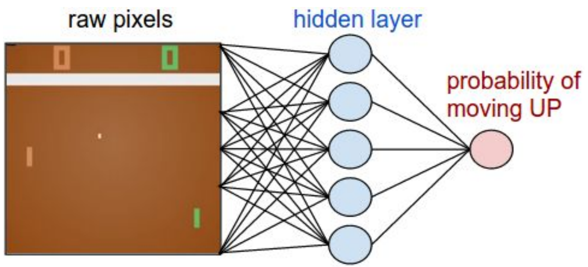
epsilon greedy!



Slides from Karpathy

Pong from pixels

$\pi(a|s)$



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

epsilon greedy!

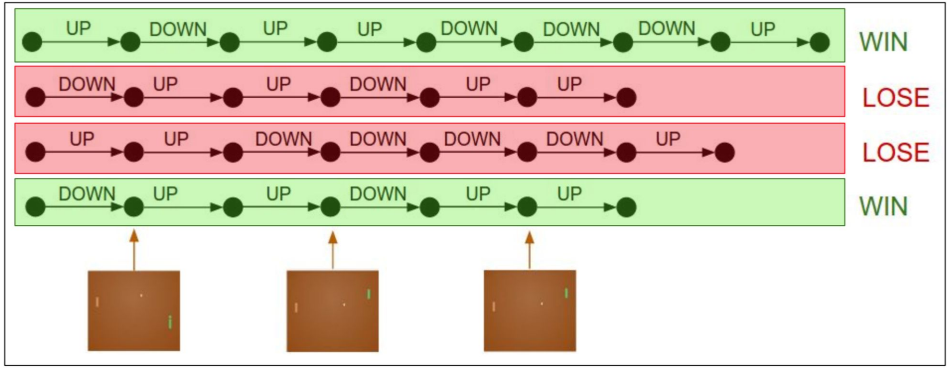
Pretend every action we took here was the correct label.

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

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$

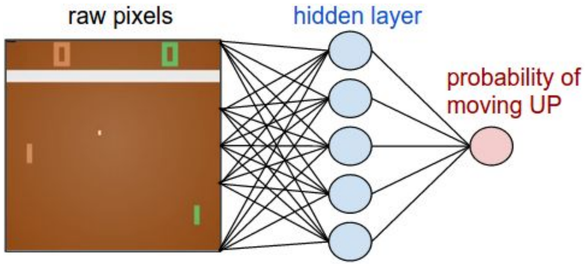
$$\boxed{\sum_i A_i * \log p(y_i|x_i)}$$



Slides from Karpathy

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
4.      Increase the probability of actions that worked well
```

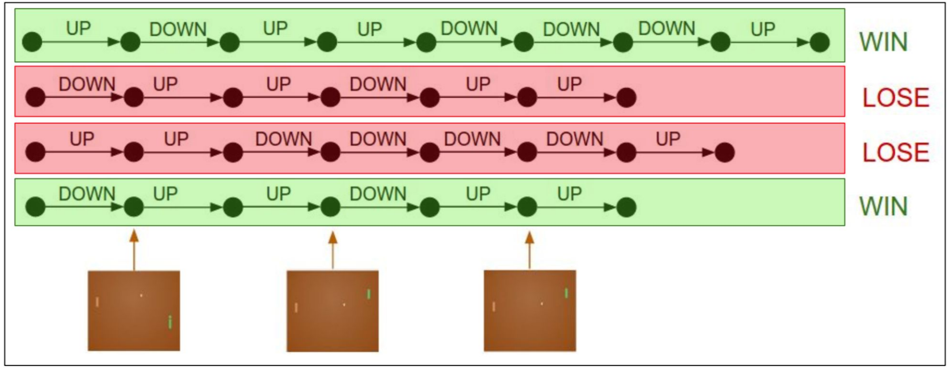
epsilon greedy!

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: $(-1) * \log p(y_i \mid x_i)$



$$\boxed{\sum_i A_i * \log p(y_i \mid x_i)}$$

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

Pong from pixels

Slides from Karpathy

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
```

$$\boxed{\sum_i A_i * \log p(y_i | x_i)}$$

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:

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

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

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

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

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

Reward of a trajectory

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

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

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

REINFORCE algorithm

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

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

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

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 θ

REINFORCE algorithm

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

$$= \int_\tau r(\tau) p(\tau;\theta) \, d\tau$$

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

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 θ

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

$$= \int_{\tau} r(\tau) p(\tau;\theta)\ d\tau$$

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

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 θ

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\dfrac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_{\tau} \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\ d\tau$$

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

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

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

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, we can estimate gradients:

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

Intuition

Gradient estimator:

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

Interpretation:

- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(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:

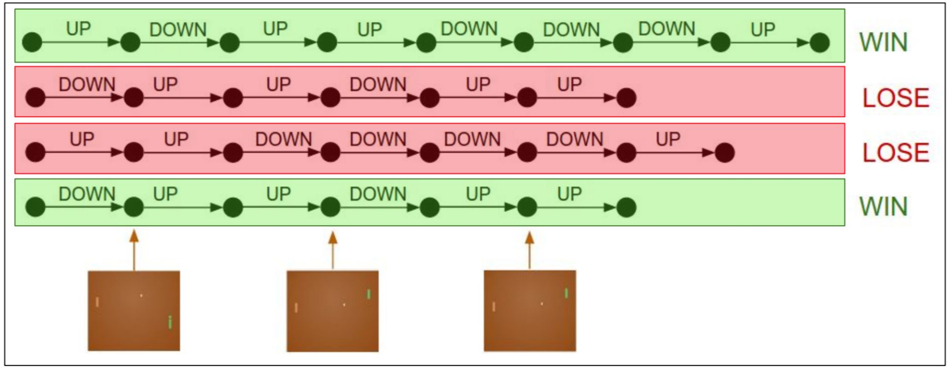
- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(trajectory)** is low, push down the probabilities of the actions seen

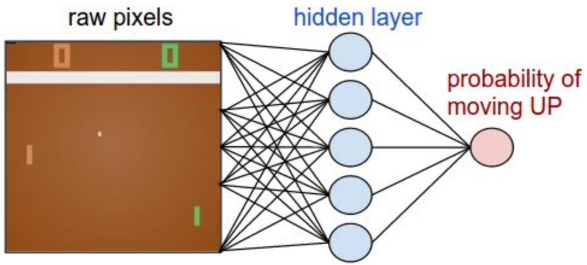
Pretend every action we took here was the correct label.

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

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$





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

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

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

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

Intuition

Gradient estimator:

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

Interpretation:

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

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

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

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(trajectory)** is high, push up the probabilities of the actions seen
- If **r(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:

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

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:

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

Actor-critic methods

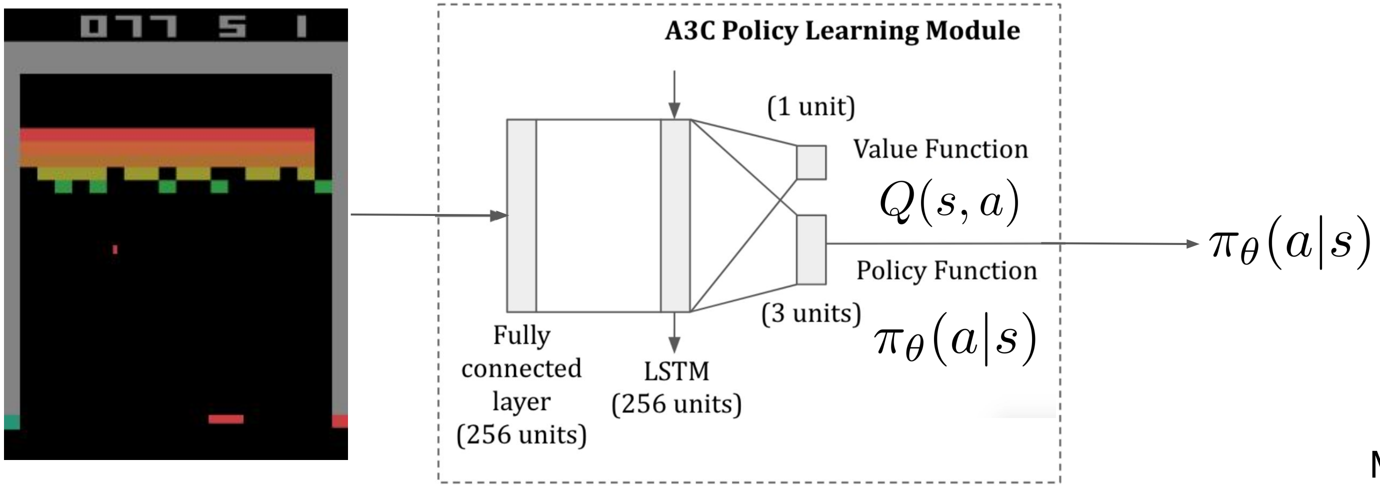
Problem: we don't know Q and V - can we learn them?

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)



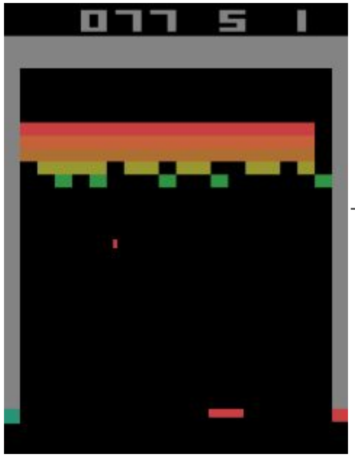
Minh et. al., ICML 2016

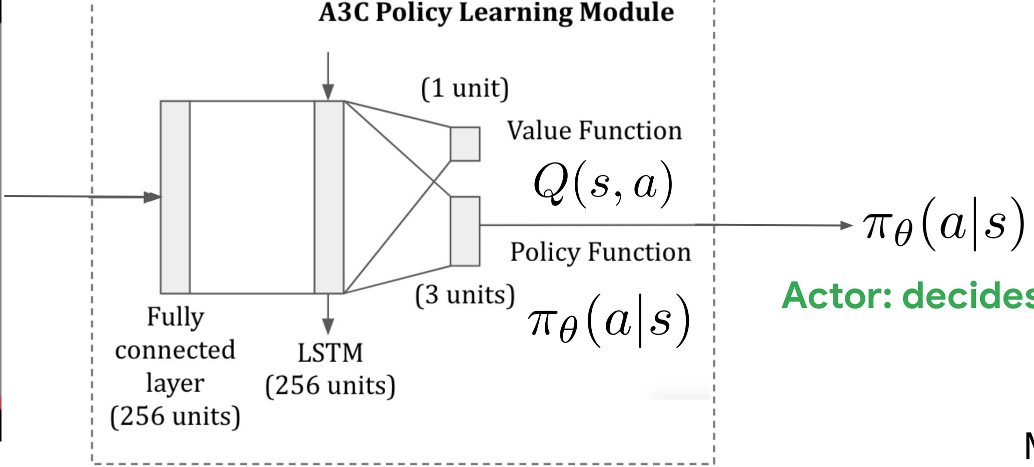
Actor-critic methods

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)

Critic: evaluates how good the action is





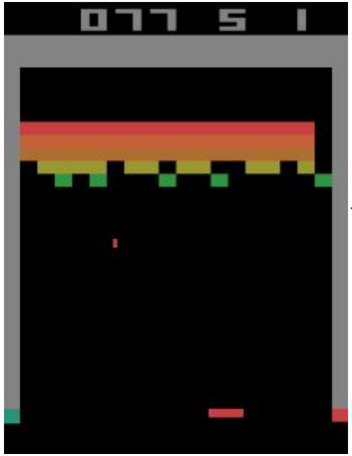
Actor: decides what actions to take

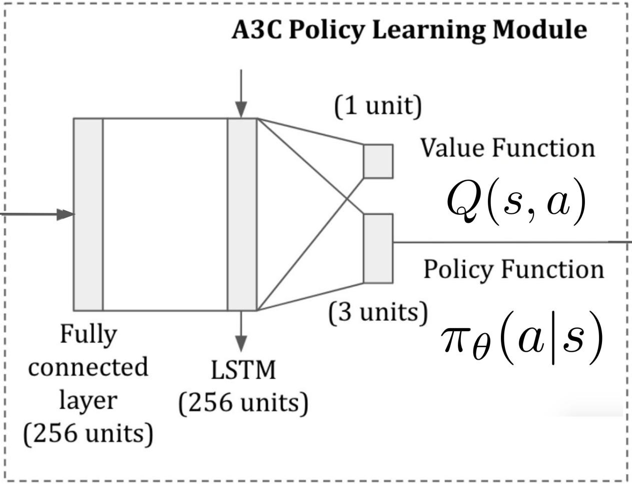
Minh et. al., ICML 2016

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

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

$\rightarrow \pi_\theta(a|s)$

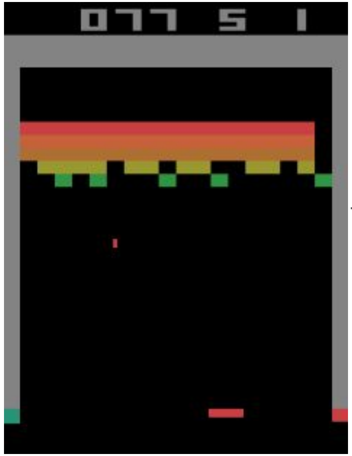
Actor: decides what actions to take

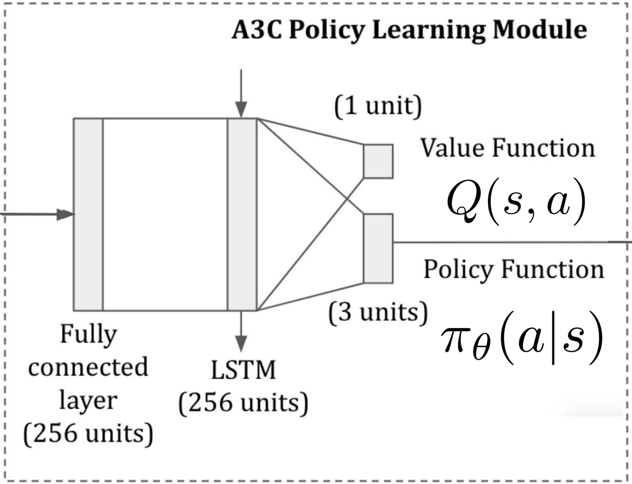
Minh et. al., ICML 2016

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

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

$$\pi_\theta(a|s)$$

Actor: decides what actions to take

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

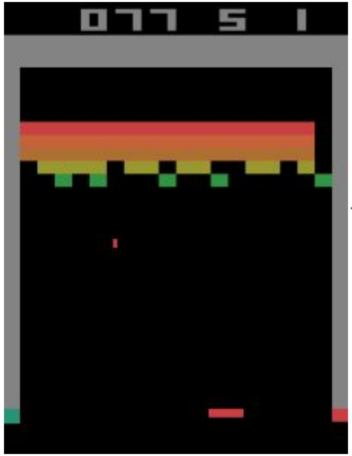
Minh et. al., ICML 2016

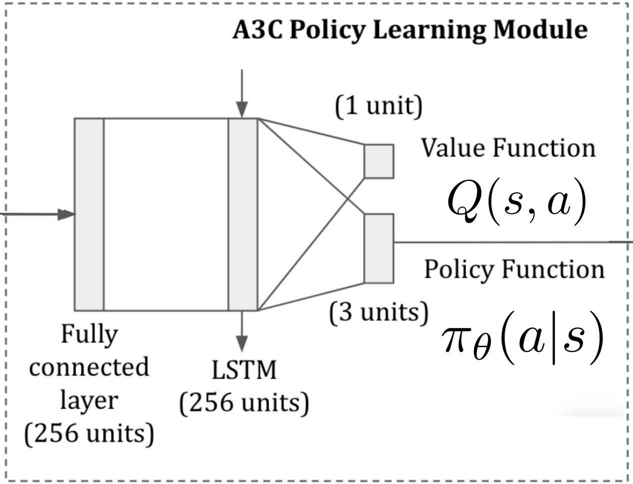
Actor-critic methods

Exploration + experience replay
Decorrelate samples
Fixed targets

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)





$$\pi_\theta(a|s)$$

Critic: evaluates how good the action is

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

Actor: decides what actions to take

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

Minh et. al., ICML 2016

Summary of RL methods

Value iteration
Policy iteration
(Deep) Q-learning

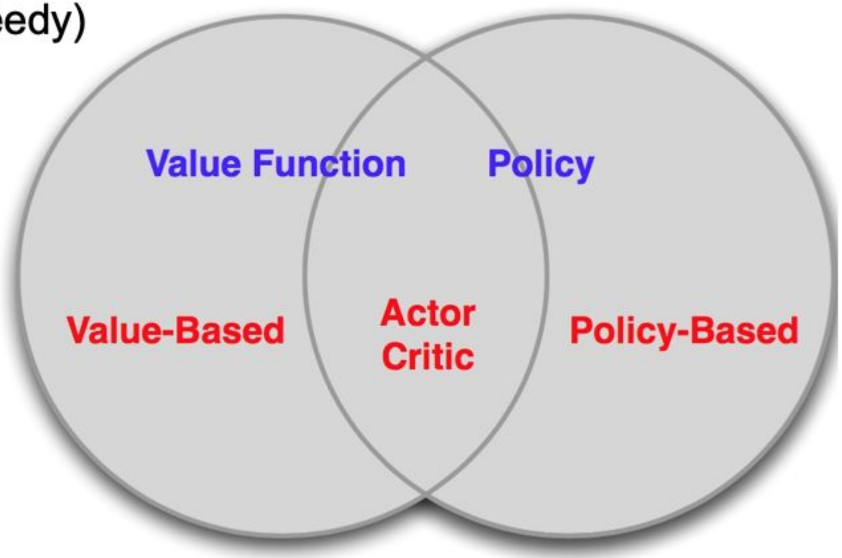
- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

Policy gradients

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

Actor (policy)
Critic (Q-values)

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



Slides from Fragkiadaki

Applications: Stochastic optimization

Stochastic Optimization

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

Stochastic Optimization

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

VAEs

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

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

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

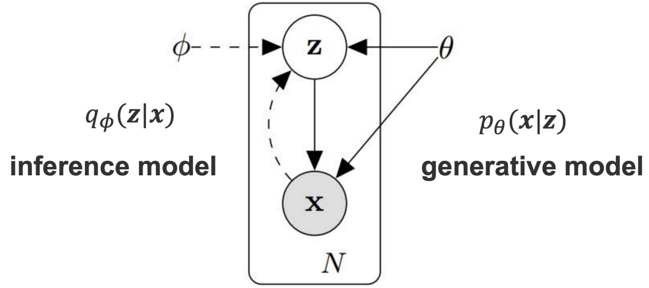


Figure courtesy: Kingma & Welling, 2014

Stochastic Optimization

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

VAEs

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

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

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

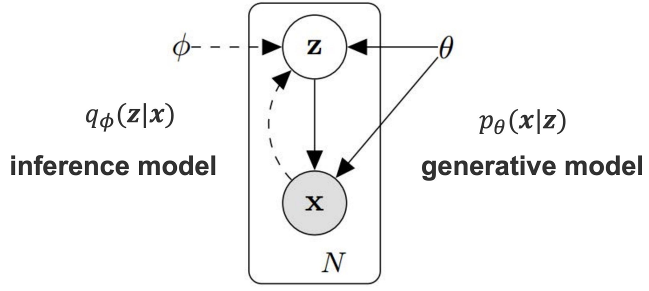
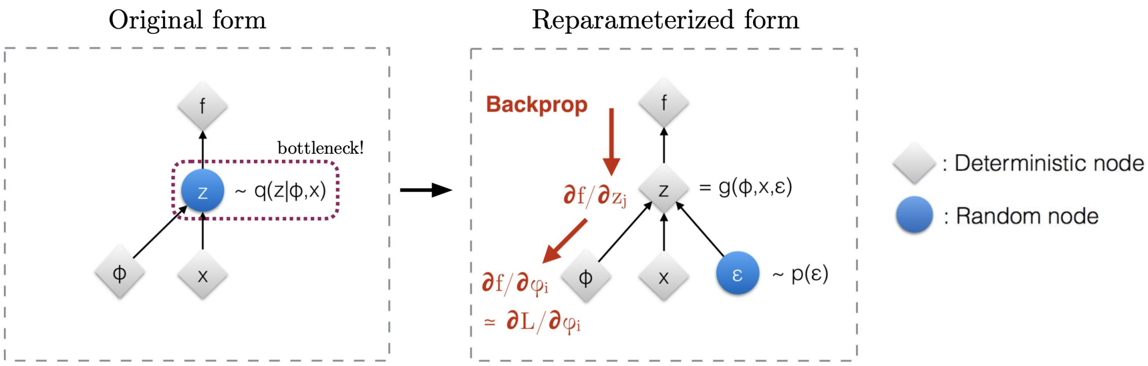


Figure courtesy: Kingma & Welling, 2014



Stochastic Optimization

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

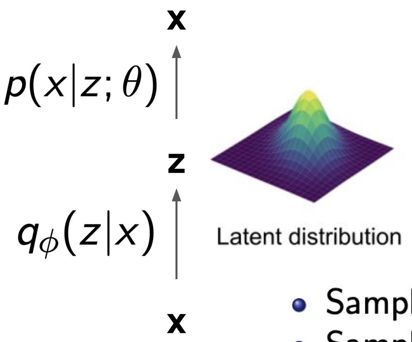
VAEs

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



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

Stochastic Optimization

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

VAEs

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

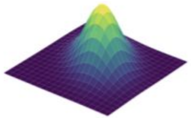
$$\max_{\theta,\phi} E_{q_\phi(z|x)}[\log p(x|z;\theta)]$$

Solve by reparameterization!

$\mathbf{x}$

$p(x|z;\theta) \uparrow$

$\mathbf{z}$



$q_\phi(z|x) \uparrow$

Latent distribution

$\mathbf{x}$

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

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

RL

$$\max_{\phi} J(\phi)$$ Reward

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

Stochastic Optimization

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

VAEs

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

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

Solve by reparameterization!

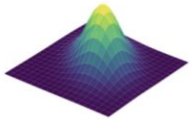
x

$p(x|z;\theta)$

z

$q_\phi(z|x)$

x



Latent distribution

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

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

RL

$$\max_{\phi} J(\phi)$$ Reward

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

Reparameterization???

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

r

???

a

$\pi_\phi(a|s)$

s

Stochastic Optimization

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

VAEs

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

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

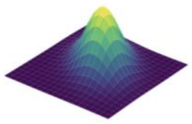
Solve by reparameterization!

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

x

$p(x|z;\theta)$ ↑

z



$q_\phi(z|x)$ ↑

Latent distribution

x

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

RL

$$\max_{\phi} J(\phi)$$ Reward

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

Reparameterization???

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

r

??? ↑

a

$\pi_\phi(a|s)$ ↑

s

REINFORCE is a general-purpose solution!

Revisiting REINFORCE

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

We want to take gradients wrt ϕ of the term:

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

Revisiting REINFORCE

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

We want to take gradients wrt ϕ of the term:

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

We can now compute a Monte Carlo estimate:

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

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

Revisiting REINFORCE

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

We want to take gradients wrt ϕ of the term:

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

We can now compute a Monte Carlo estimate:

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

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

What we derived: sample trajectories and compute:

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

Revisiting REINFORCE

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

We can now compute a Monte Carlo estimate:

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

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

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

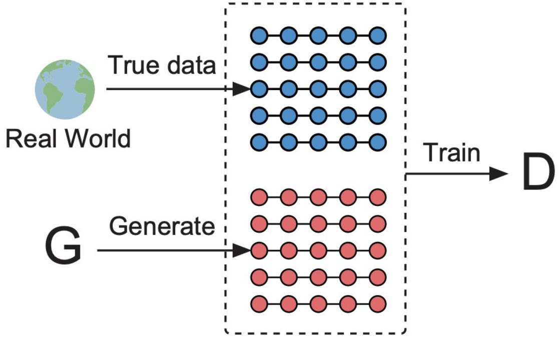
We just need the distribution q() to allow for easy sampling

- z can be discrete or continuous!
- q(z) can be a discrete and continuous distribution! (but must be differentiable wrt ϕ)
- 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



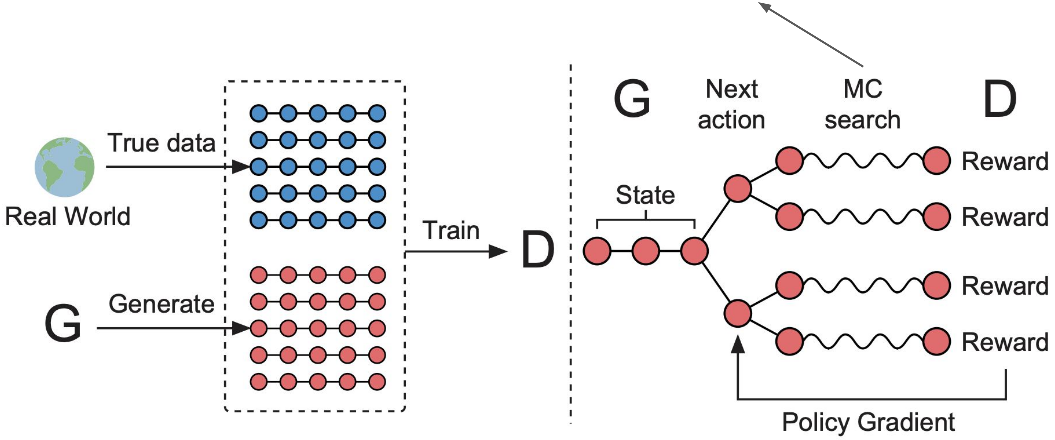
[Yu et. al., AAAI 2017]

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



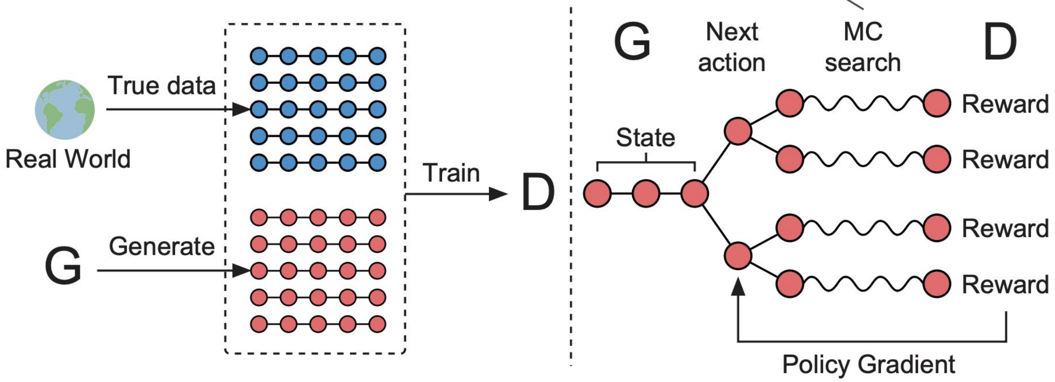
[Yu et. al., AAAI 2017]

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

complete sentences

LM

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

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

disc reward

[Yu et. al., AAAI 2017]

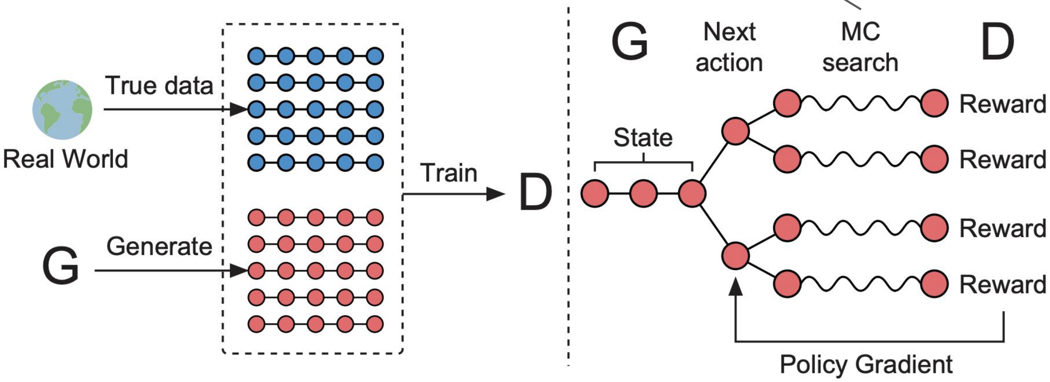
Applications: Text generation

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

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

More efficient search strategy for most likely sentence



$$\text{Sample } \overset{\text{complete sentences}}{\boxed{\mathbf{z}^1, \cdots, \mathbf{z}^K}} \text{ from } \overset{\text{LM}}{\boxed{q_\phi(\mathbf{z})}} \text{ and estimate}$$

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[f(\mathbf{z})] \approx \frac{1}{K} \sum_k \underbrace{\boxed{f(\mathbf{z}^k)}}_{\text{disc reward}} \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

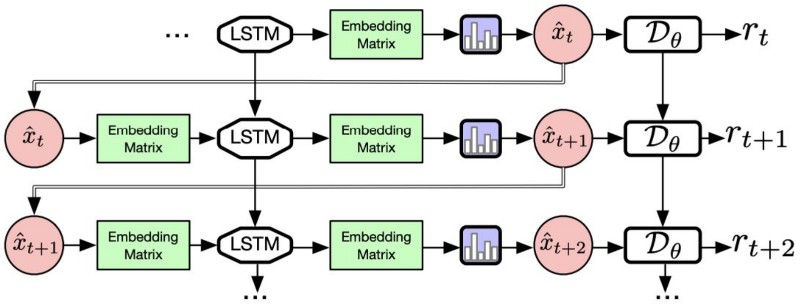
[Yu et. al., AAAI 2017]

Training discriminator is easy - all params are differentiable

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

Sample $\overbrace{\boxed{\mathbf{z}^1, \cdots, \mathbf{z}^K}}^{\textbf{prefixes}}$ from $\overbrace{\boxed{q_\phi(\mathbf{z})}}^{\text{LM}}$ and estimate

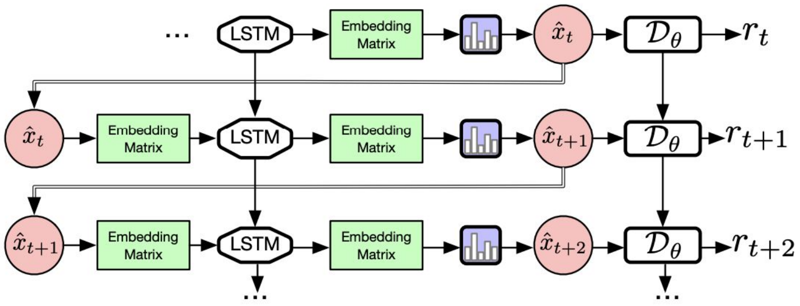
$$\nabla_\phi E_{q_\phi(\mathbf{z})}[f(\mathbf{z})] \approx \frac{1}{K} \sum_k \underbrace{\boxed{f(\mathbf{z}^k)}}_{\text{disc reward}} \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

[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)



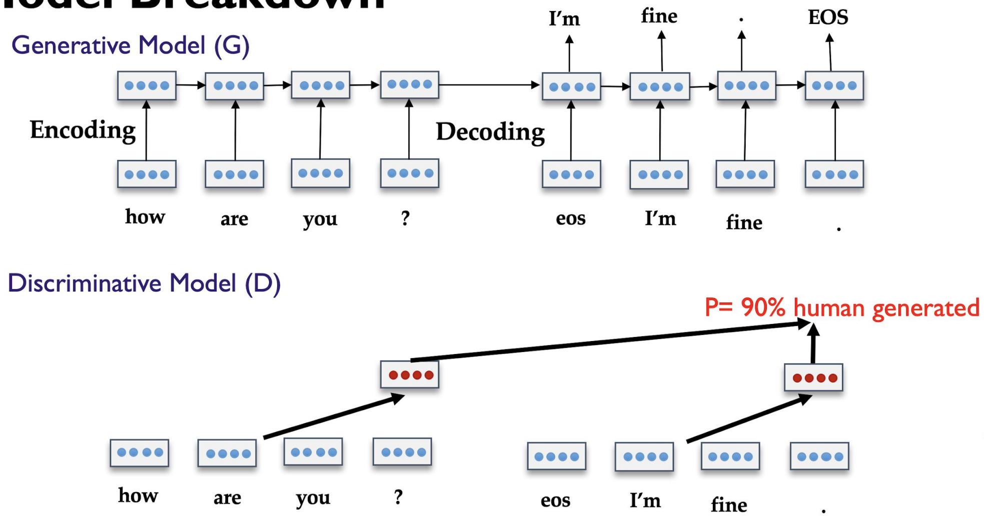
Other approaches as well without using policy gradients
- discriminator directly comparing in logit space
- use Gumbel softmax (Jang et al. 2016)
- see more in https://www.cl.uni-heidelberg.de/statnlpgroup/blog/rl4nmt/ and https://deepgenerativemodels.github.io/assets/slides/cs236_lecture15.pdf

[d'Autume et. al., NeurIPS 2019]

Applications: Dialog generation

GANs for dialog generation and evaluation

Model Breakdown



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

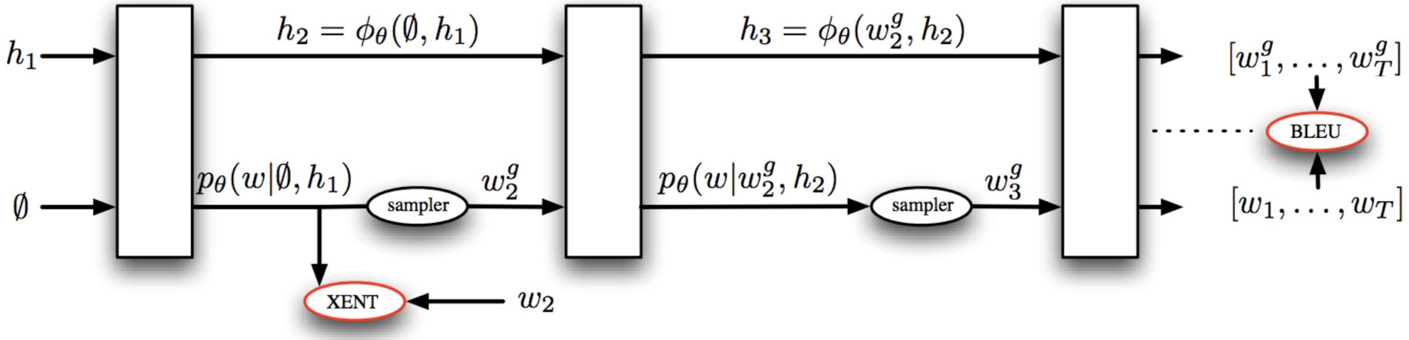
$$\nabla_\phi E_{q_\phi(\mathbf{z})}[f(\mathbf{z})] \approx \frac{1}{K} \sum_k \boxed{f(\mathbf{z}^k)} \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

[Li et. al., EMNLP 2017]
Slides from Wang, ACL 2018 tutorial

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.



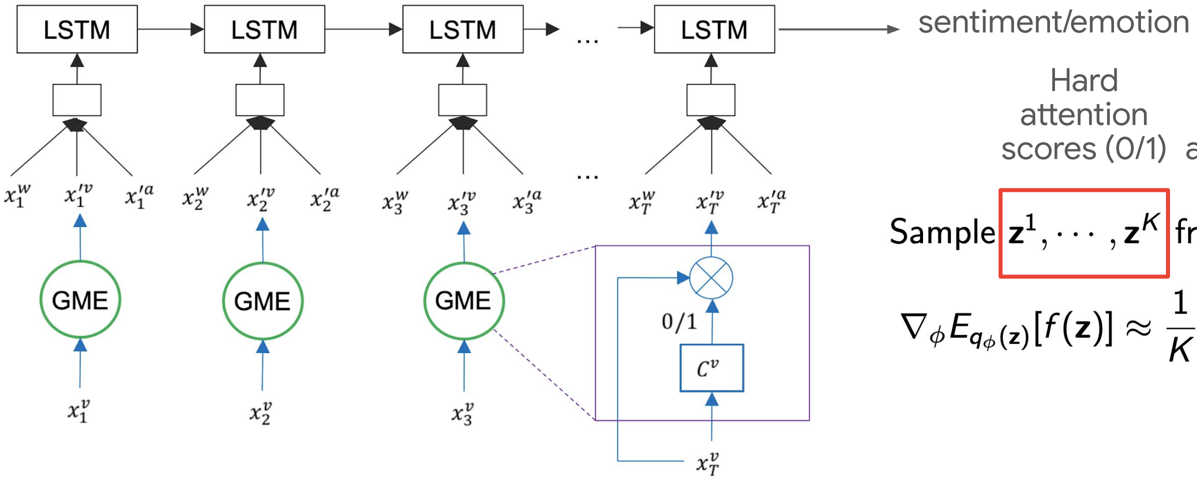
Sample $\mathbf{z}^1, \cdots, \mathbf{z}^K$ from $q_\phi(\mathbf{z})$ and estimate

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[f(\mathbf{z})] \approx \frac{1}{K}\sum_k \underbrace{\boxed{f(\mathbf{z}^k)}}_{\text{BLEU score}} \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

[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



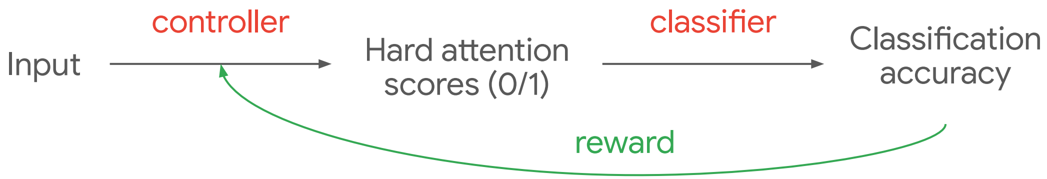
Hard attention scores (0/1)

Controller: input to hard attention scores

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

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

classification accuracy

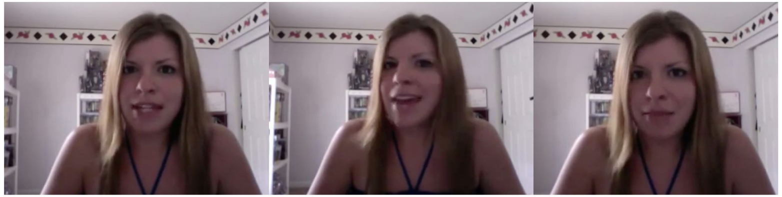


[Xu et. al., ICML 2015]
[Chen et al., ICMI 2017]

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

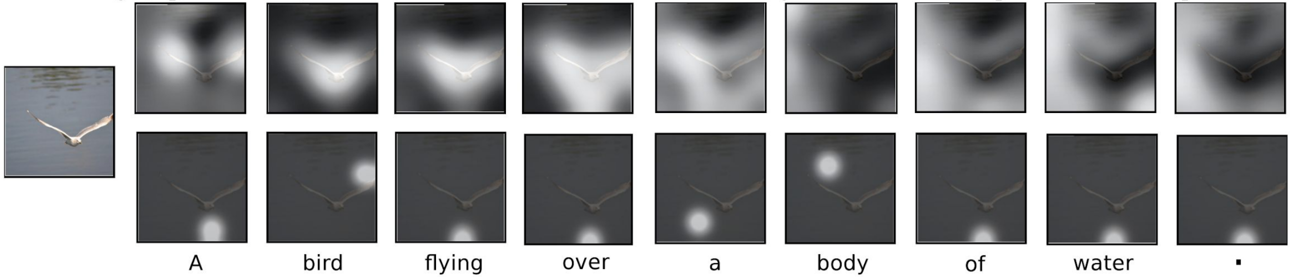
Sentiment analysis, emotion recognition



Reject Pass Reject

Image captioning

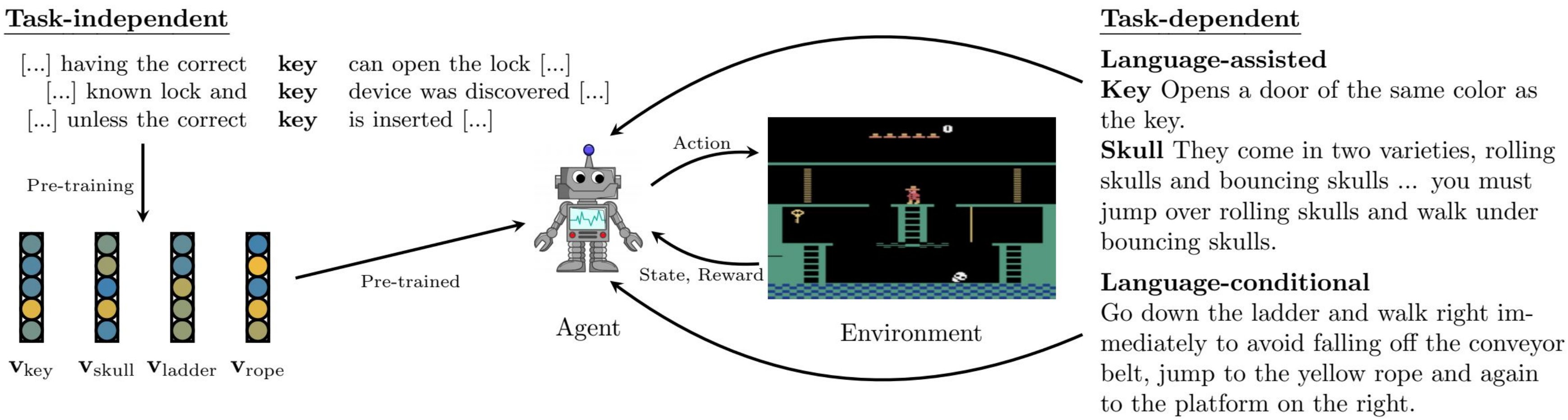
Figure 3. Visualization of the attention for each generated word. The rough visualizations obtained by upsampling the attention weights and smoothing. (top)“soft” and (bottom) “hard” attention (note that both models generated the same captions in this example).



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

Applications: RL and Language

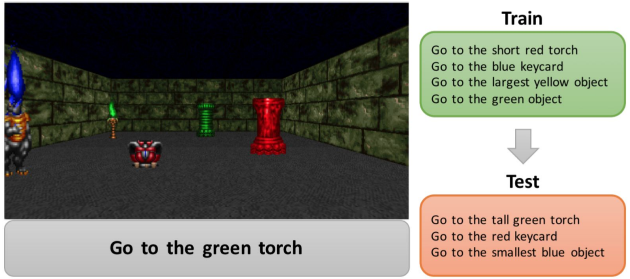
RL and Language



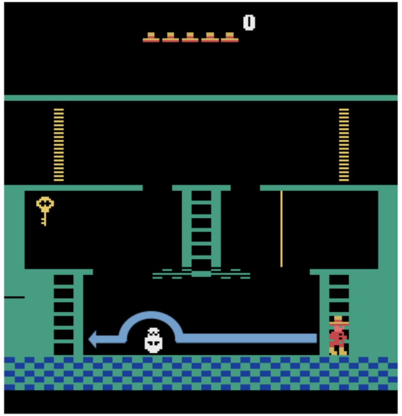
Luketina et. al., IJCAI 2019

Language-conditional RL

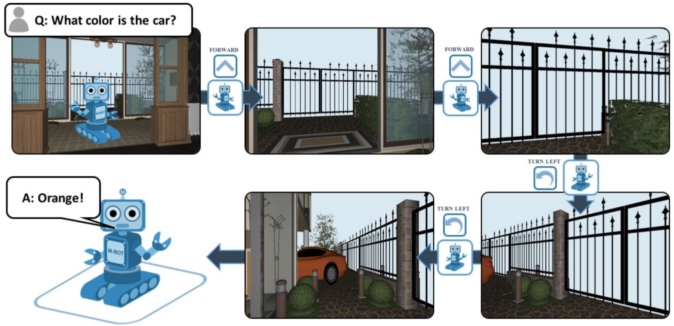
Instruction following



Rewards from instructions



Language in S and A

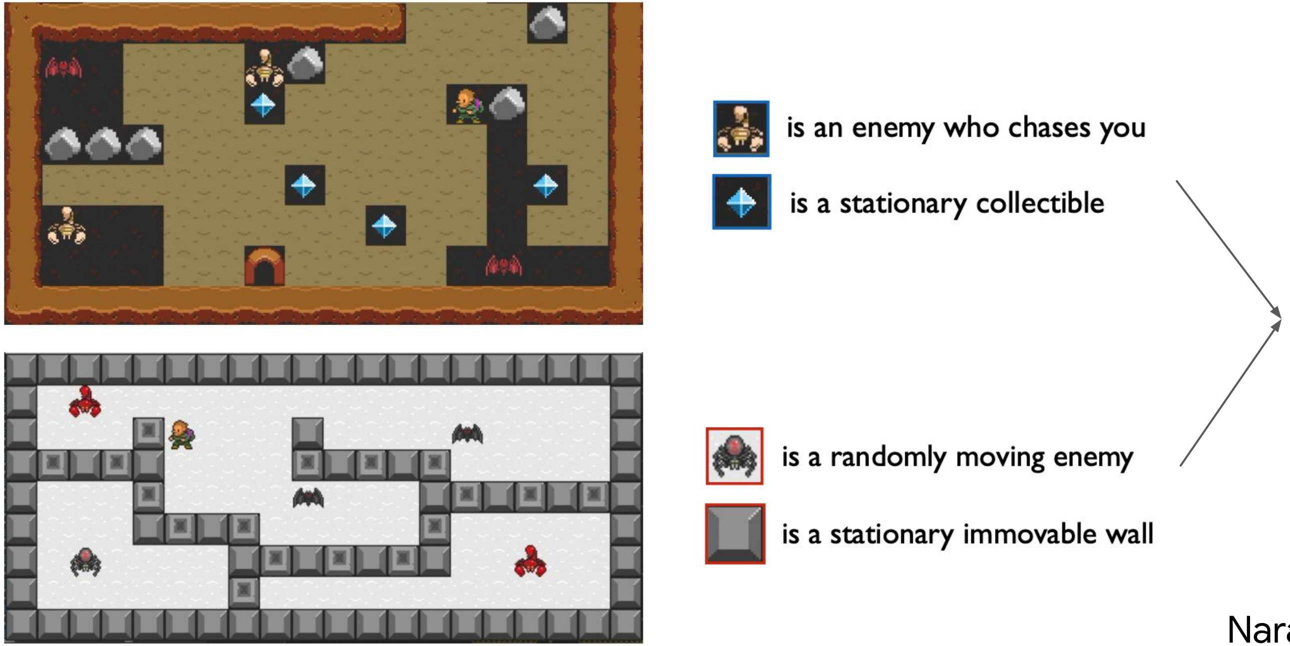


Language-assisted RL

- Language for communicating domain knowledge
- Language for structuring policies

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

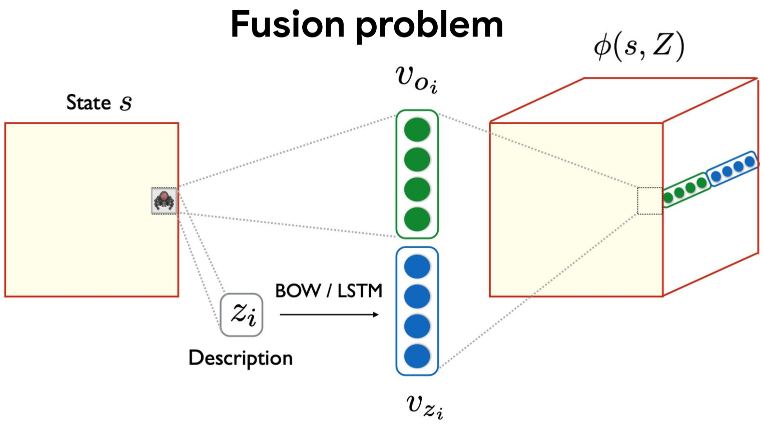


from Amazon Mturk :-(asked annotators to play the game and describe entities

Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

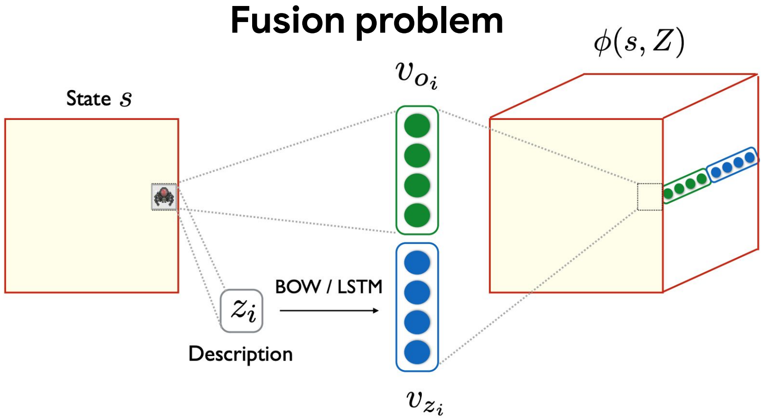
- Properties of entities in the environment are annotated by language

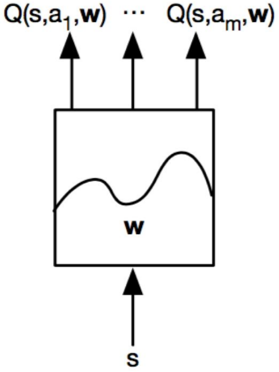


Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

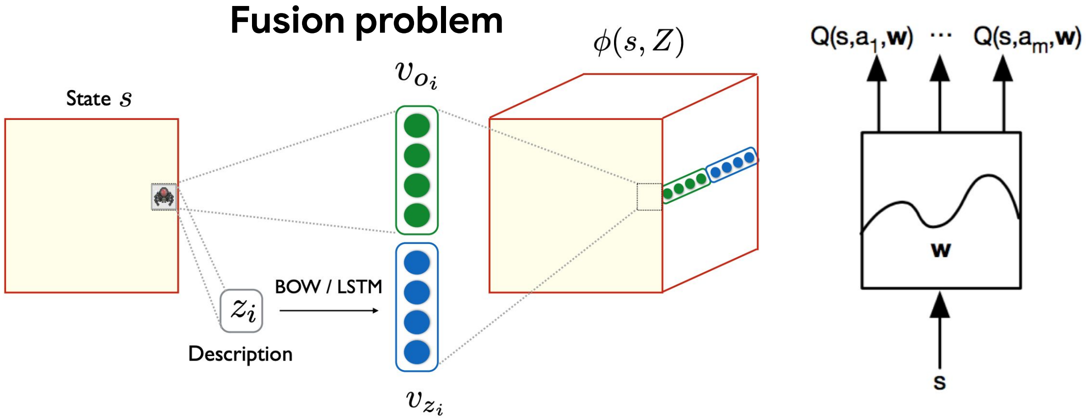




Narasimhan et. al., JAIR 2018

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**

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.*

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 ?

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.

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 ?

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

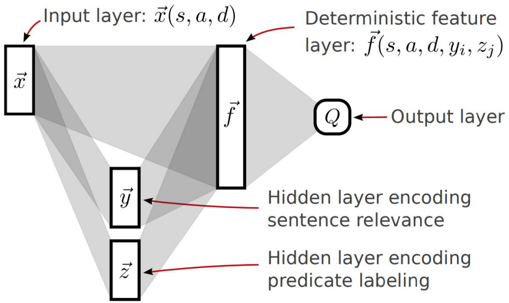
The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

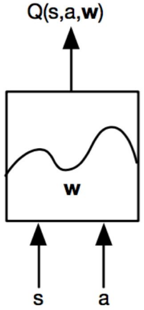
Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

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

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

Relevant sentences

- 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

A: action-description
S: state-description

- After the road is built, use the settlers to start improving the terrain.
  S S S A A A A A
- When the settlers becomes active, chose build road.
  S S S A A A
- Use settlers or engineers to improve a terrain square within the city radius
  A S ✘ A A S A ✘ S S S S

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

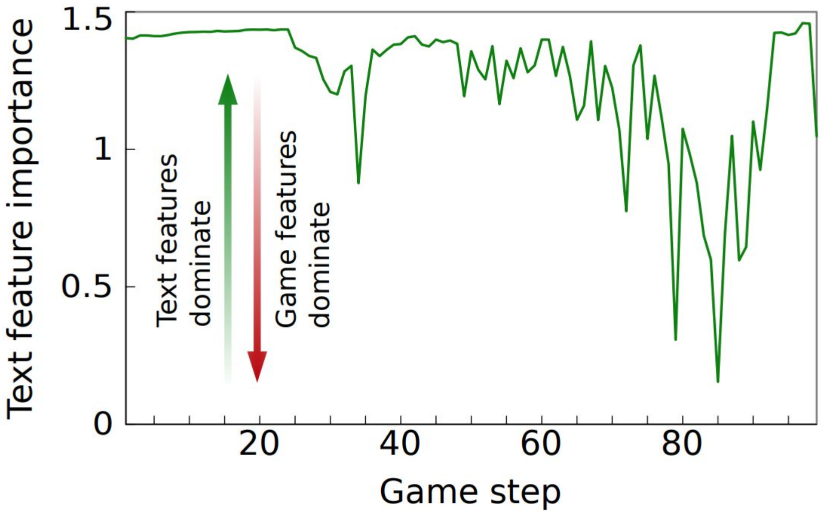
Method	% Win	% Loss	Std. Err.
Random	0	100	—
Built-in AI	0	0	—
Game only	17.3	5.3	$\pm$ 2.7
Sentence relevance	46.7	2.8	$\pm$ 3.5
Full model	**53.7**	5.9	$\pm$ 3.5
Random text	40.3	4.3	$\pm$ 3.4
Latent variable	26.1	3.7	$\pm$ 3.1

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

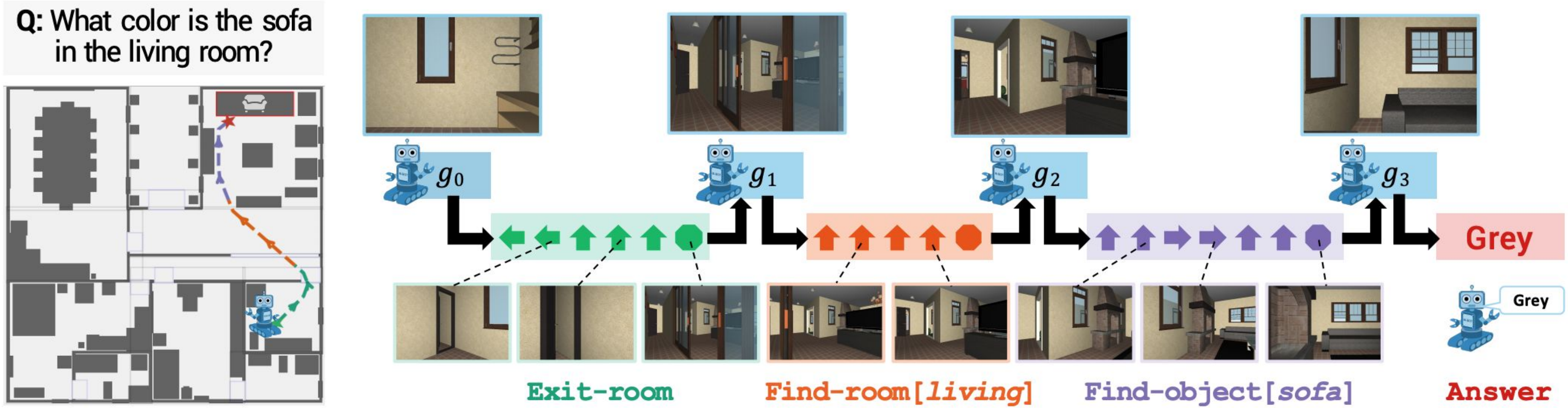


Language is most important at the start when you don't have a good policy
Afterwards, the model relies on game features

Branavan et. al., JAIR 2012

Language for structuring policies

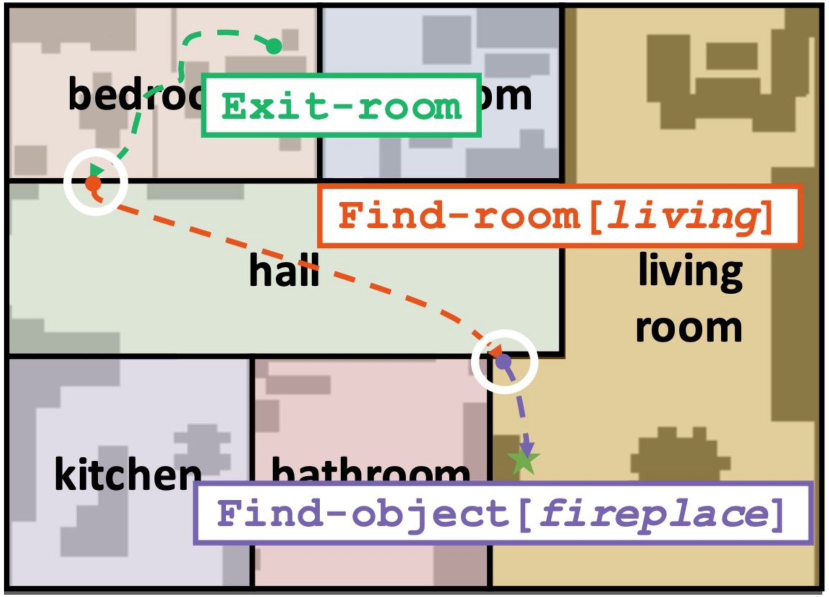
- Composing modules for Embodied QA

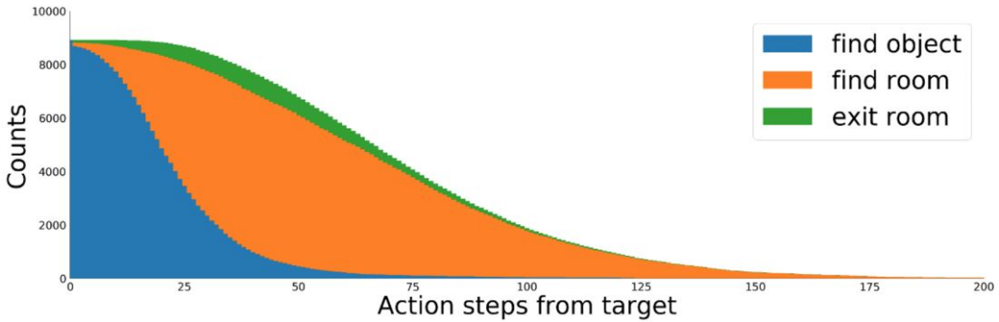


Das et. al., CoRL 2018

Language for structuring policies

- Composing modules for Embodied QA





Das et. al., CoRL 2018

Summary of RL methods

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

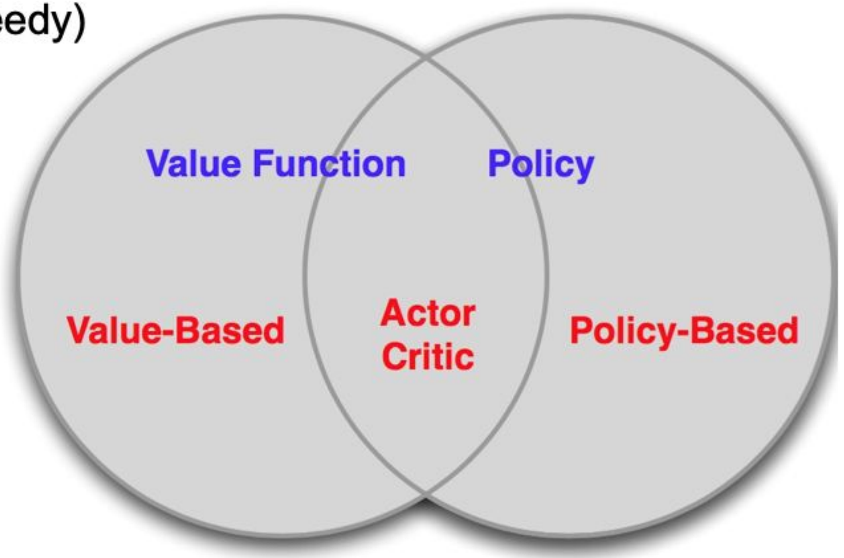
Value iteration
Policy iteration
(Deep) Q-learning

- Policy Based
  - No Value Function
  - Learned Policy

Policy gradients

- Actor-Critic
  - Learned Value Function
  - Learned Policy

Actor (policy)
Critic (Q-values)



Summary of applications

Stochastic optimization

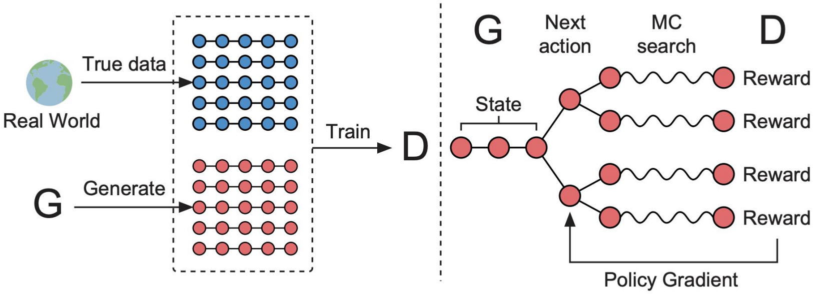
sentences LM

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

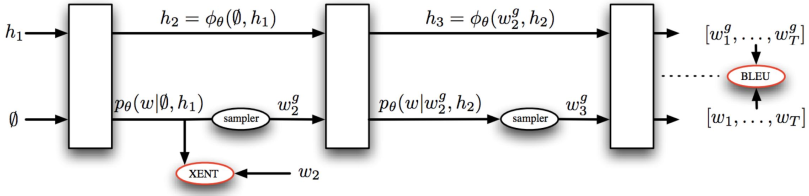
$$\nabla_\phi E_{q_\phi(\mathbf{z})}[f(\mathbf{z})] \approx \frac{1}{K} \sum_k \boxed{f(\mathbf{z}^k)} \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

disc reward

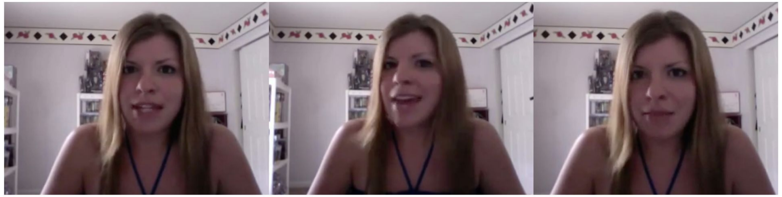
Text generation



General reward functions



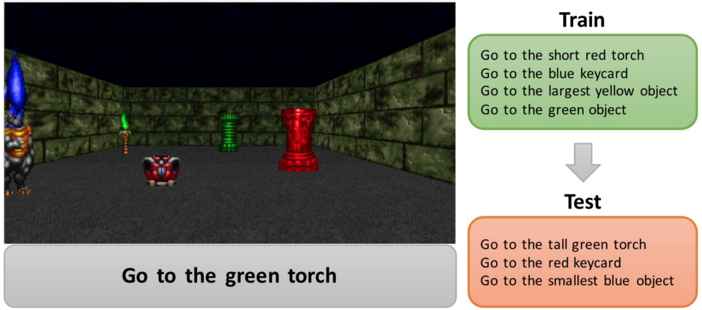
Discrete layers



Reject Pass Reject

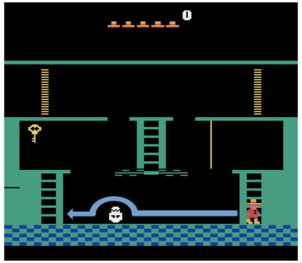
Summary of applications

Instruction following



Language as domain knowledge

Language for rewards



"Jump over the skull while going to the left"

Language to structure policies