Intro to Reinforcement Learning Part II

Paul Liang

pliang@cs.cmu.edu

@pliang279

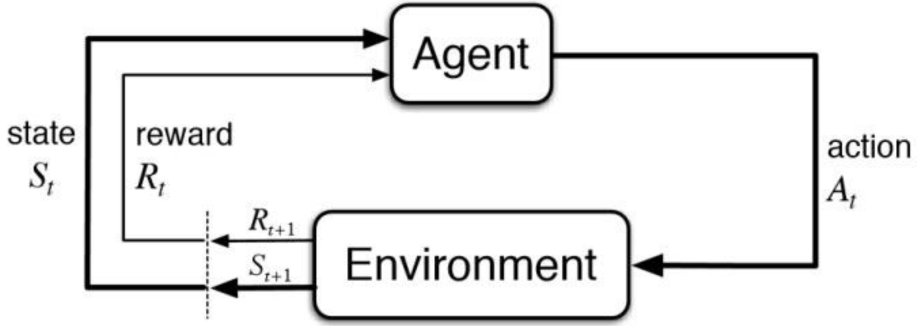
Used Materials

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

Recap: Markov Decision Process (MDPs)

An MDP is defined by:

- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Trajectory

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

Recap: Return

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

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

γ close to 0 leads to "myopic" evaluation

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

Recap: Policy

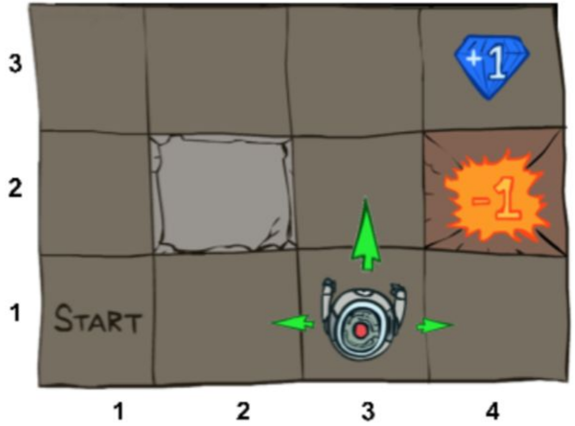
Definition: A policy is a distribution over actions given states,

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

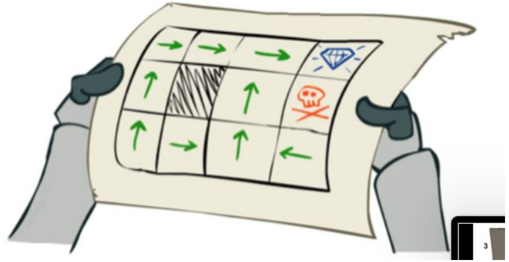
- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

$$\pi(s) = \textit{the action taken with prob} = 1 \ \textit{when}\ S_t = s$$



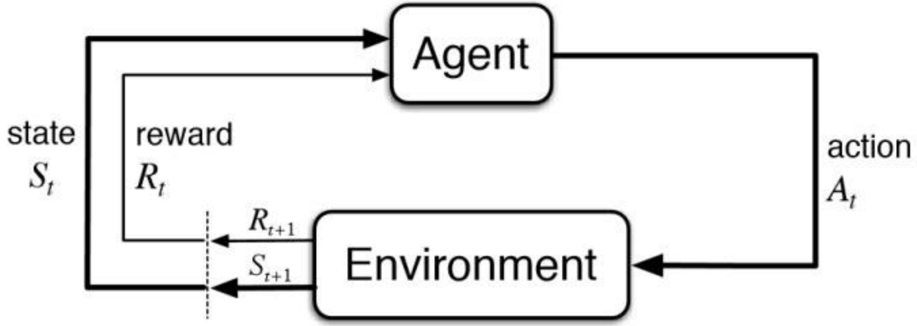
π:



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

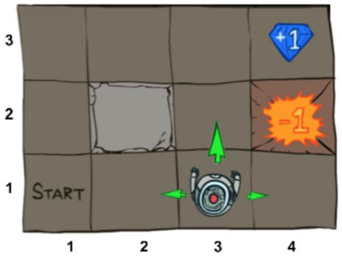


Return:

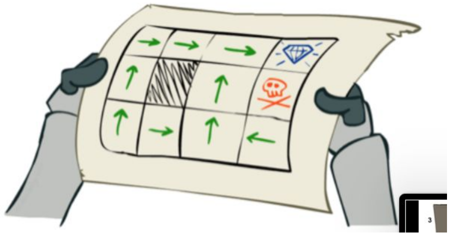
$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Goal:

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



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

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**

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

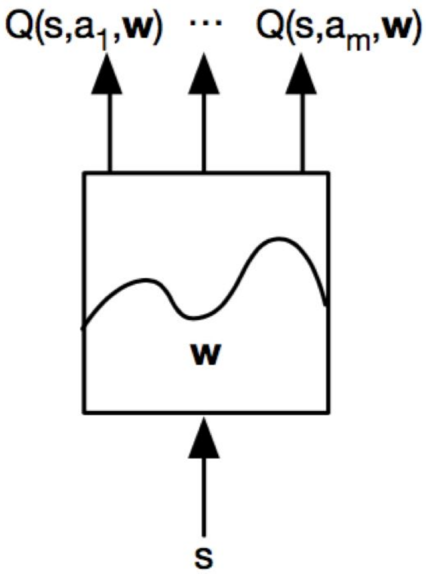
simulation and exploration, epsilon greedy is important!

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

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

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



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

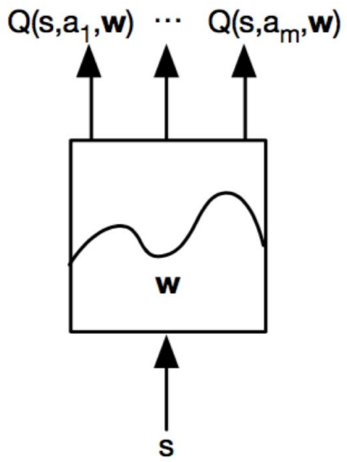
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$

Recap: Deep Q-learning

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

- 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

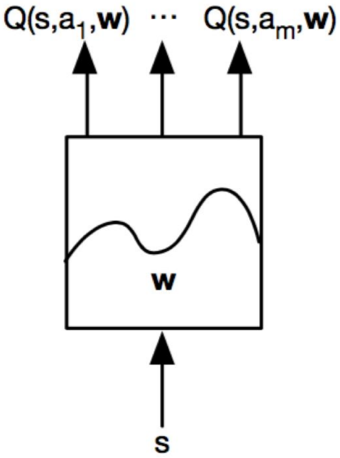
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$



- Use stochastic gradient descent

Update **w-** with updated **w** every ~1000 iterations

Recap: Deep Q-learning



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

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

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

Works for high-dimensional state and action spaces
Generalizes to unseen states

Recap: Obtaining the optimal policy

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

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

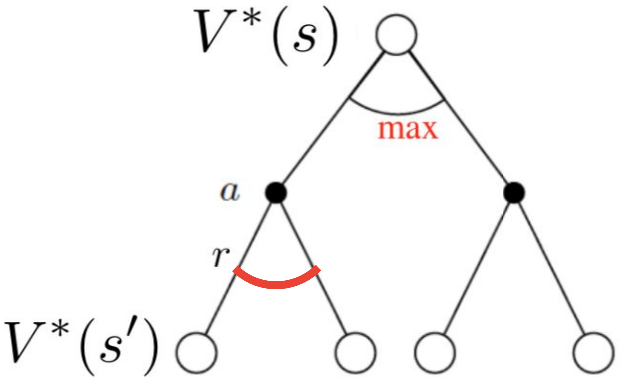
Recap: Obtaining the optimal policy

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

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

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

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



Contents

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

Value-based and Policy-based RL

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

State value functions

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

Action value functions

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

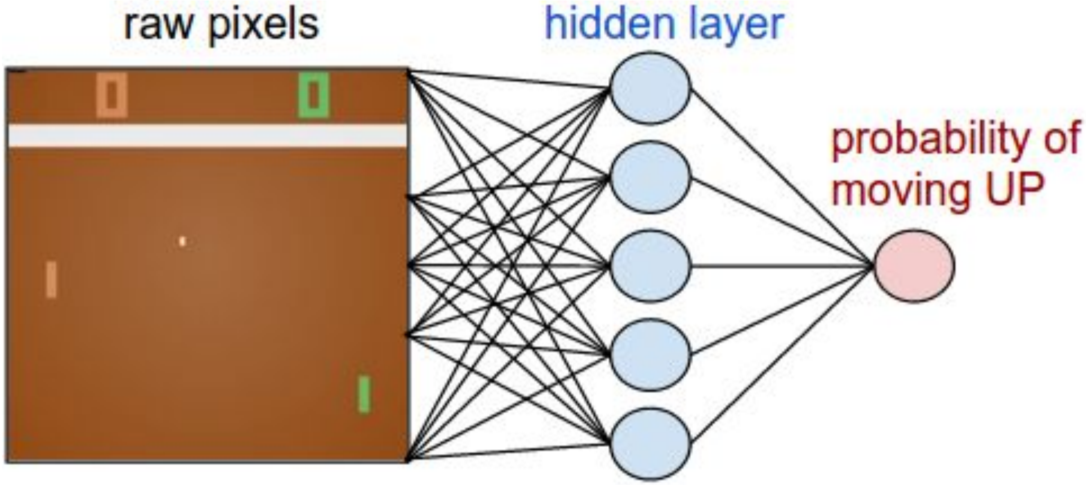
$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\ \epsilon, & \text{else} \end{cases}$$

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

Value-based and Policy-based RL

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

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



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'

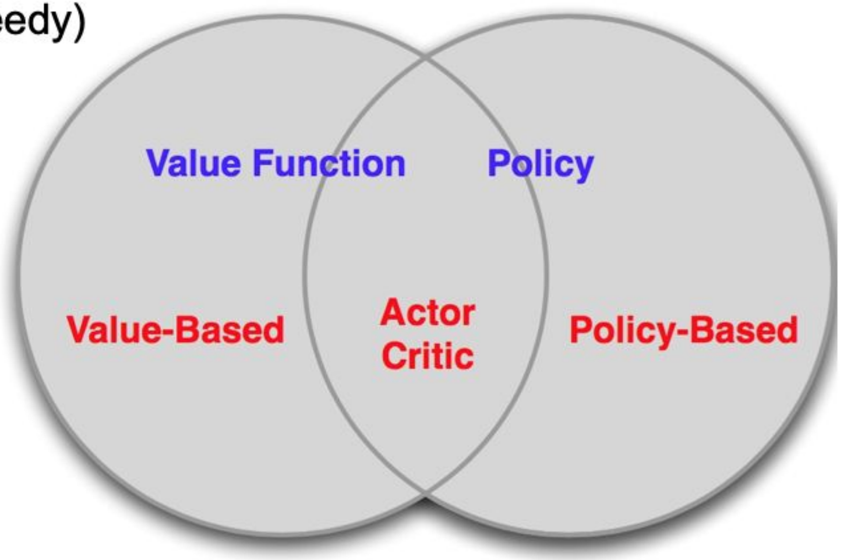
- V: doesn't prescribe actions
  - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg\max_u Q_\theta(s, u)$
  - Challenge for continuous / high-dimensional action spaces*

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

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

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



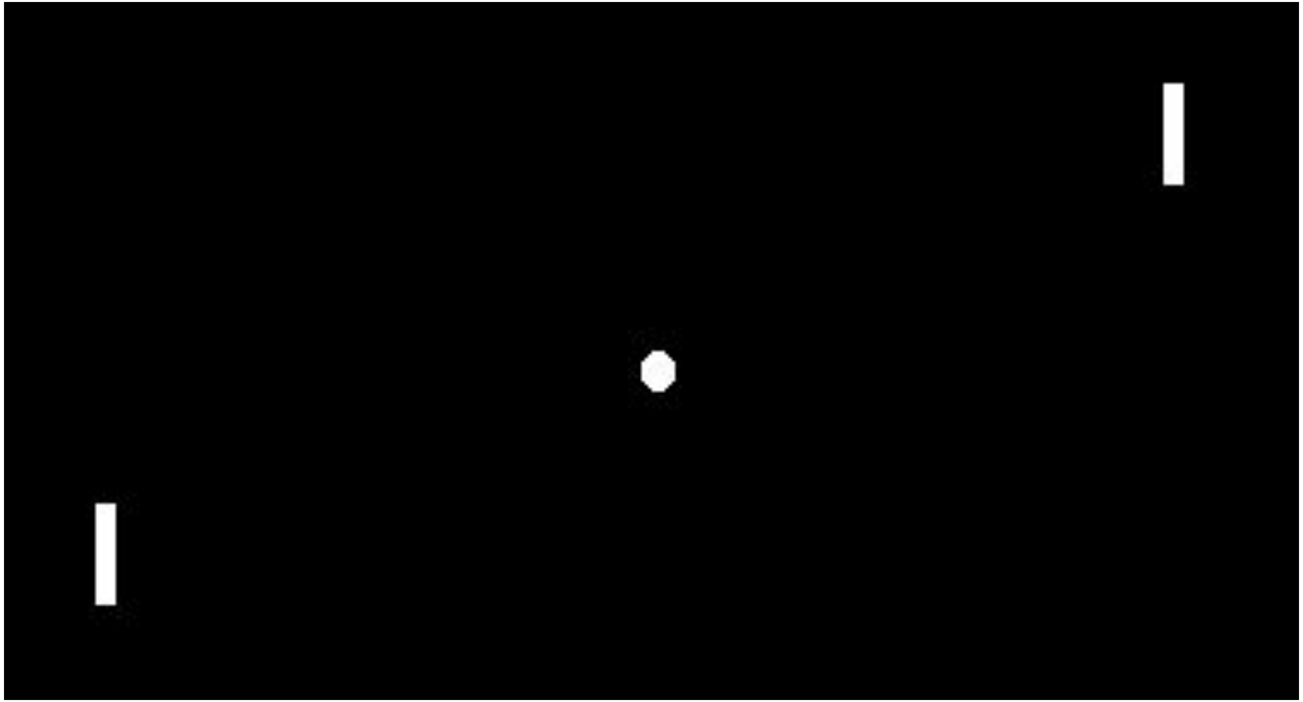
Value-based and Policy-based RL

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

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

Pong from pixels

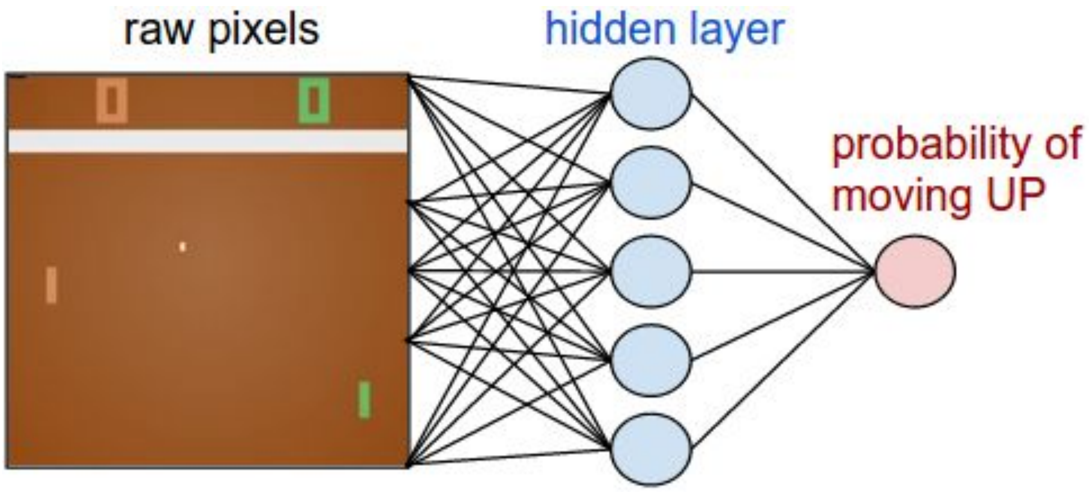


Pong from pixels

e.g.,

height width

[80 x 80]
array of



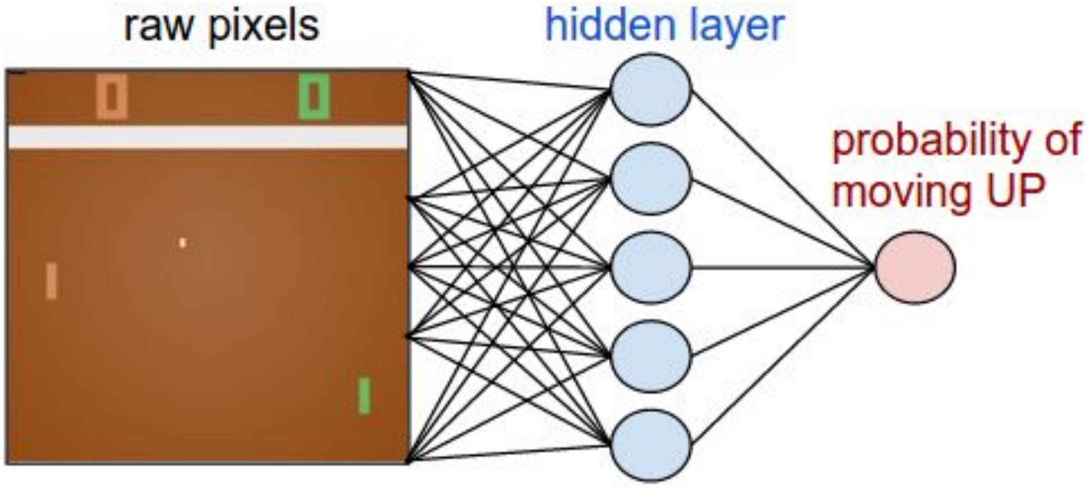
Pong from pixels

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?

Pong from pixels

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

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

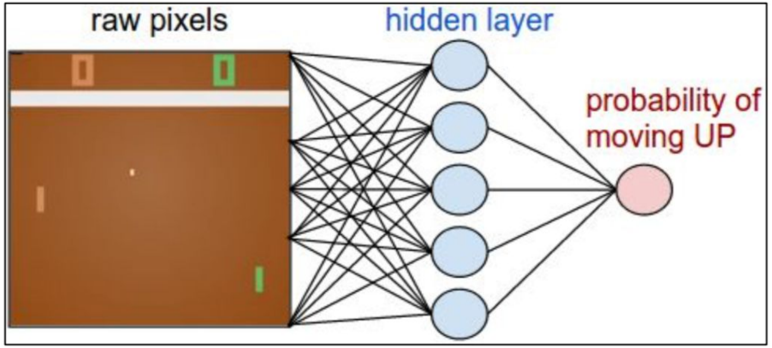
Pong from pixels

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

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

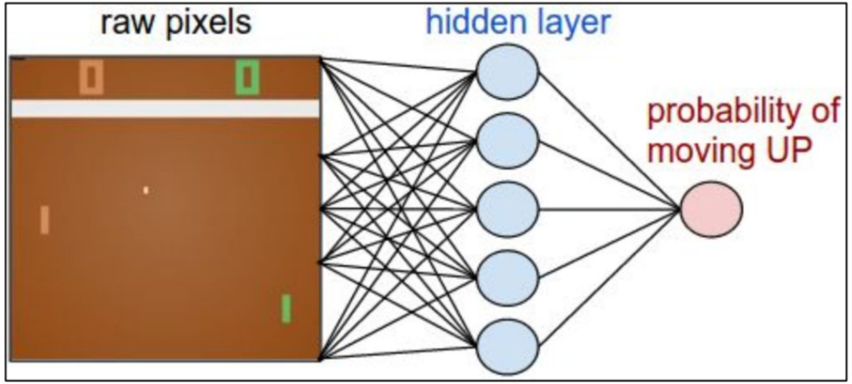
maximize:

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



Pong from pixels

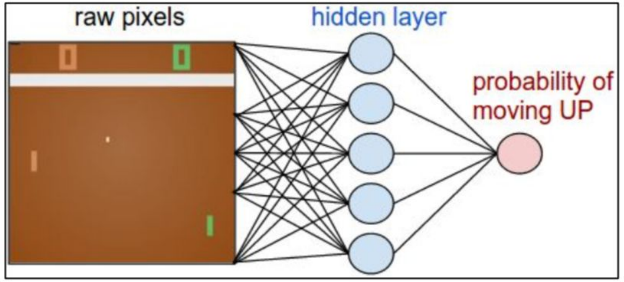
Except, we don’t have labels...



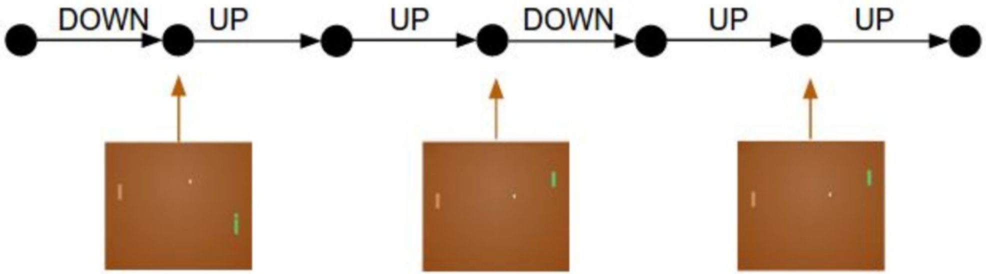
Should we go UP or DOWN?

Pong from pixels

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



Rollout the policy and collect an episode

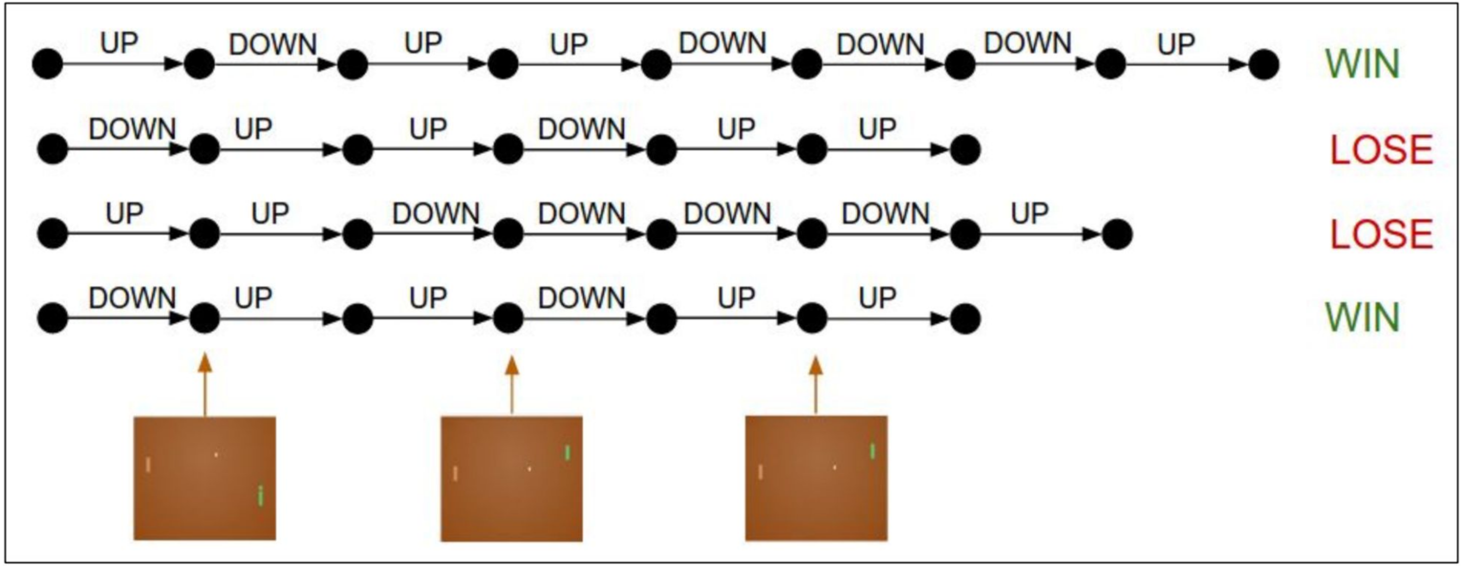


WIN

Pong from pixels

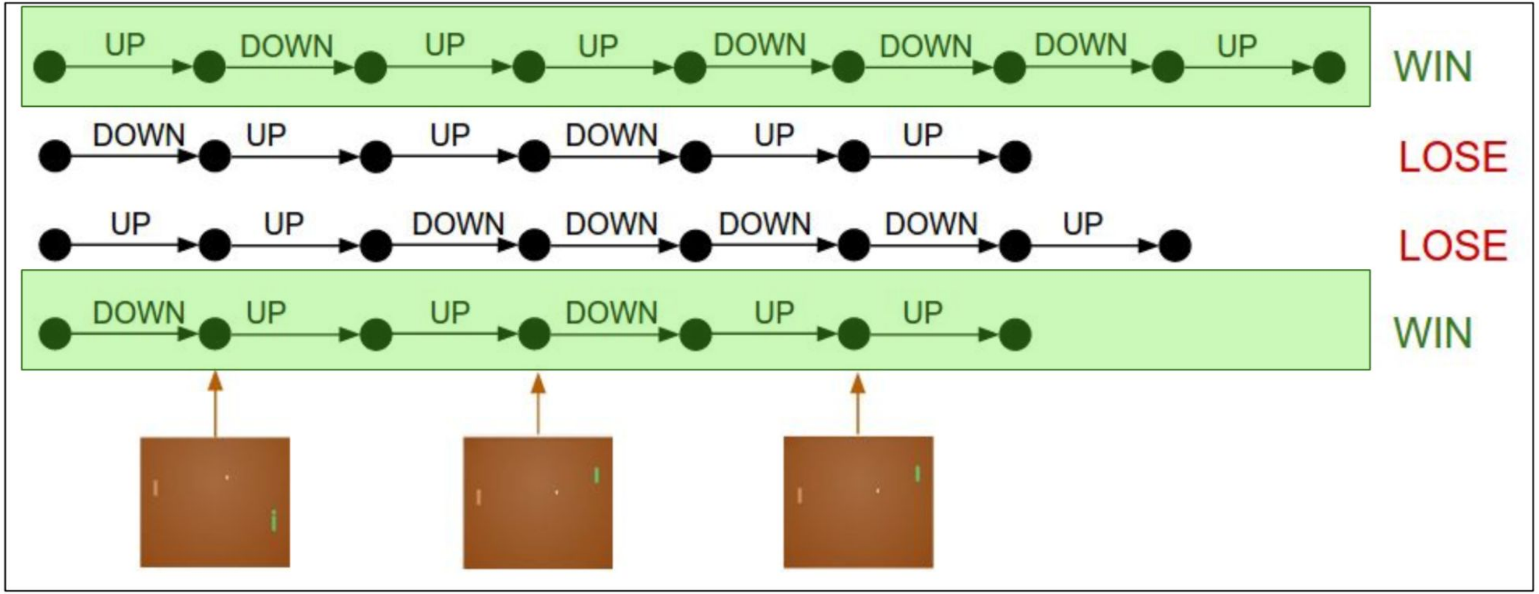
Collect many rollouts...

4 rollouts:



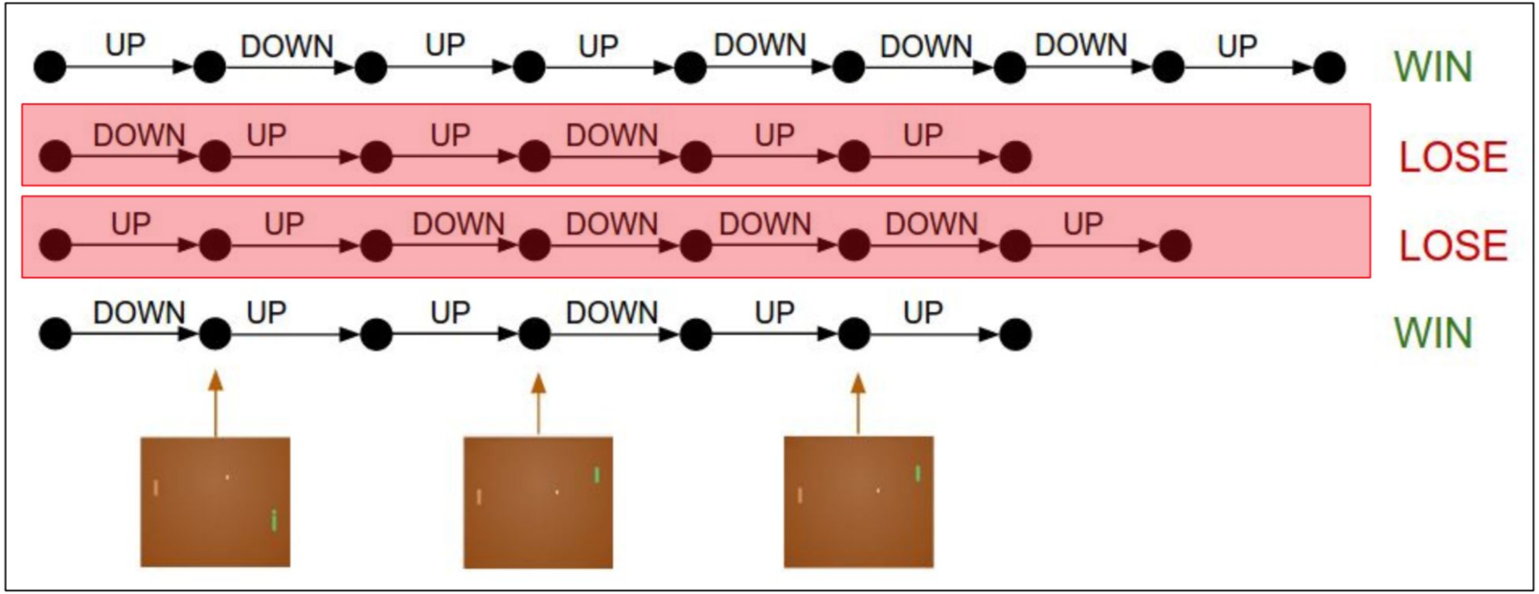
Pong from pixels

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



Pong from pixels

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



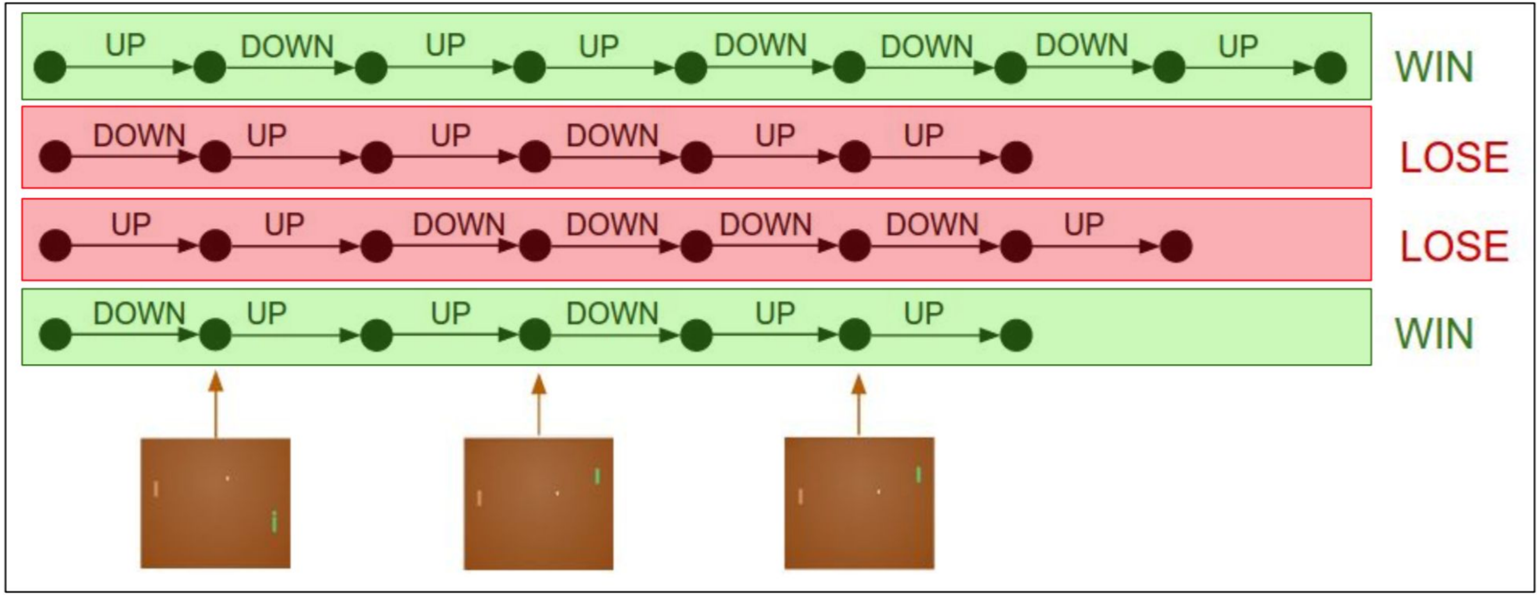
Pong from pixels

Pretend every action we took here was the correct label.

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

Pretend every action we took here was the wrong label.

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



Pong from pixels

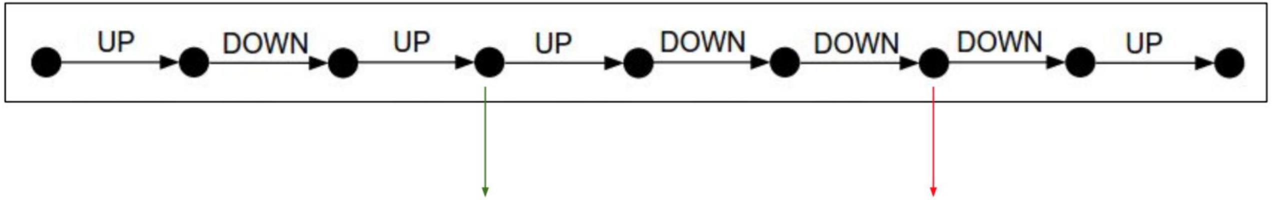
Discounting

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

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

Discounted rewards

0.21 0.24 0.27 -0.81 -0.9 -1 0 0



Reward +1.0

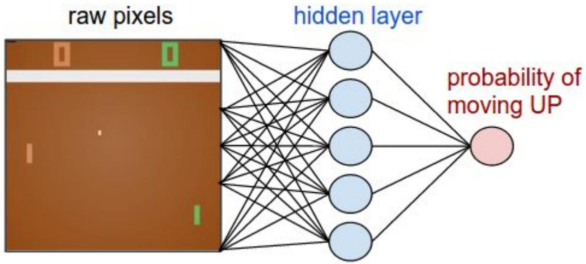
Reward -1.0

\gamma = 0.9

Pong from pixels

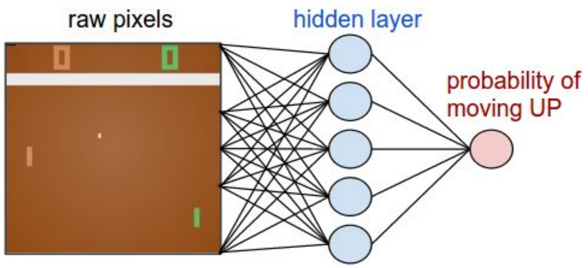
$\pi(a \mid s)$

1. `Initialize a policy network at random`



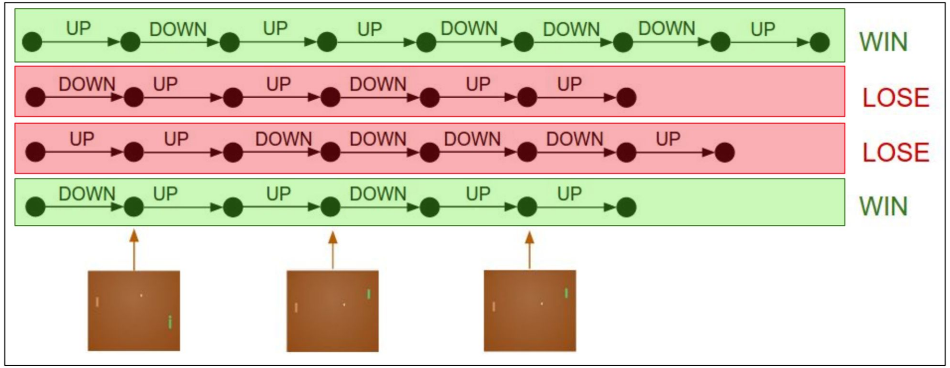
Pong from pixels

$\pi(a|s)$



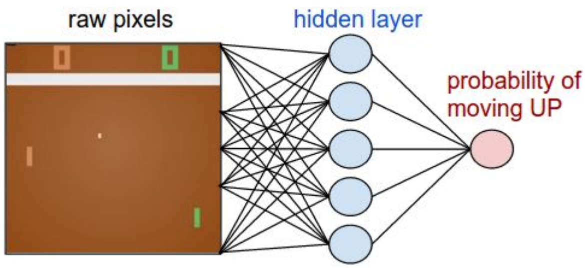

```
1.  Initialize a policy network at random
2.  Repeat Forever:
3.      Collect a bunch of rollouts with the policy
```

epsilon greedy!



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
```

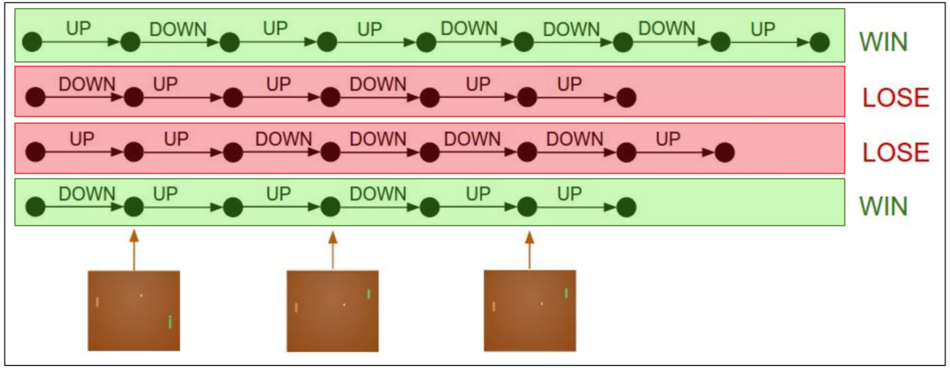
epsilon greedy!

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.

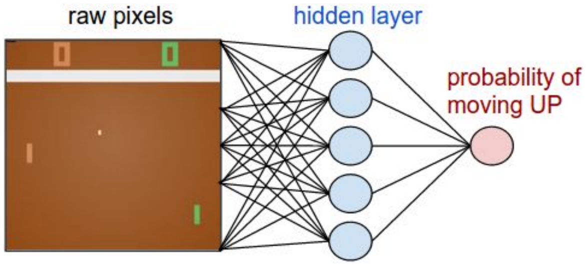
maximize: $(-1) * \log p(y_i \mid x_i)$



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

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
```

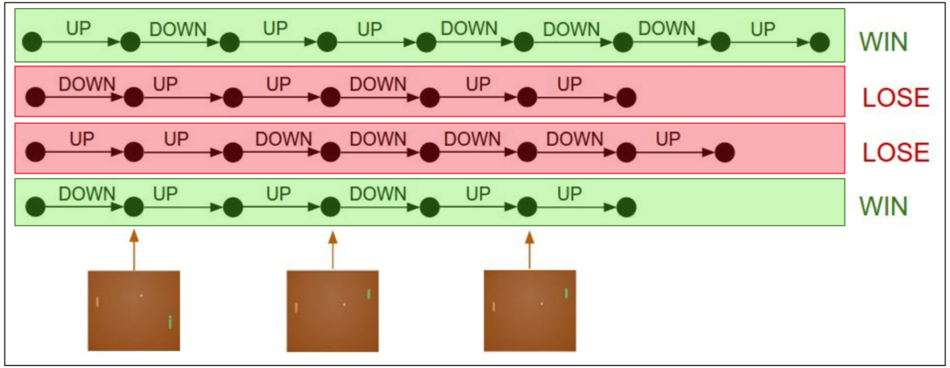
epsilon greedy!

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



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

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
```

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

Policy gradients

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

For each policy, define its value:

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

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 parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

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

Policy gradients

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

For each policy, define its value:

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

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

How can we do this?

Policy gradients

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

For each policy, define its value:

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

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

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

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

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: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\dfrac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

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

If we inject this back:

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

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

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 τ, we can estimate J(θ) with

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

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

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:

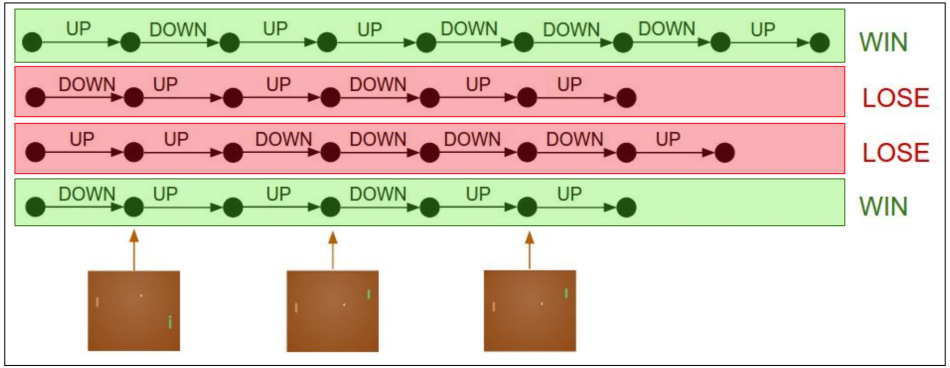
- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

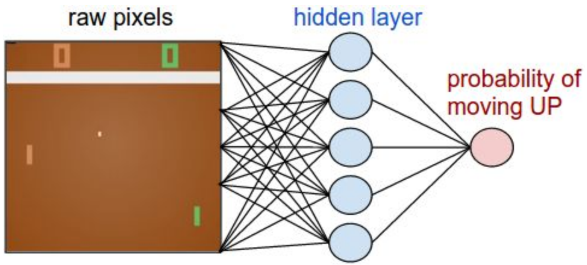
Pretend every action we took here was the correct label.

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

Pretend every action we took here was the wrong label.

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





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

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

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

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) 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
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

epsilon greedy

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

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

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

Variance reduction with a baseline

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

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

Variance reduction with a baseline

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

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

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

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

e.g. exponential moving average of the rewards. Provably reduces variance while remaining unbiased.

Actor-critic methods

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

Q: What does this remind you of?

Actor-critic methods

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

Q: What does this remind you of?

A: Q-function and value function!

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

Using this, we get the estimator:

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

Actor-critic methods

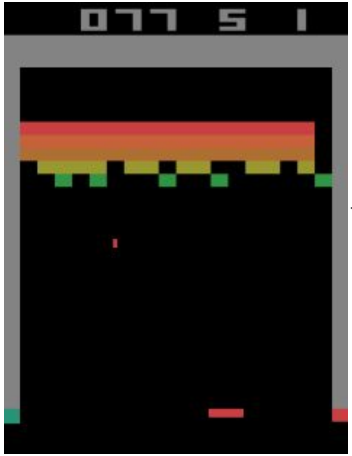
Problem: we don't know Q and V. Can we learn them?

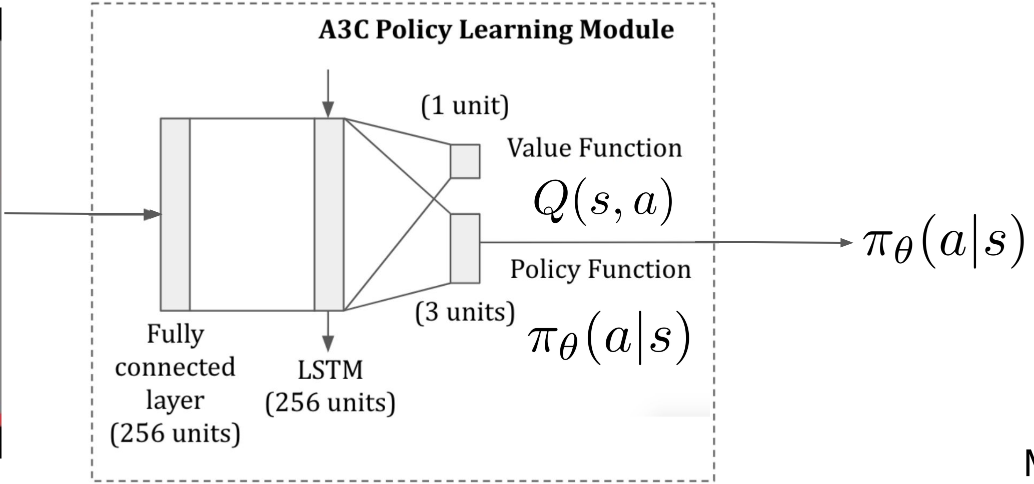
Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

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).





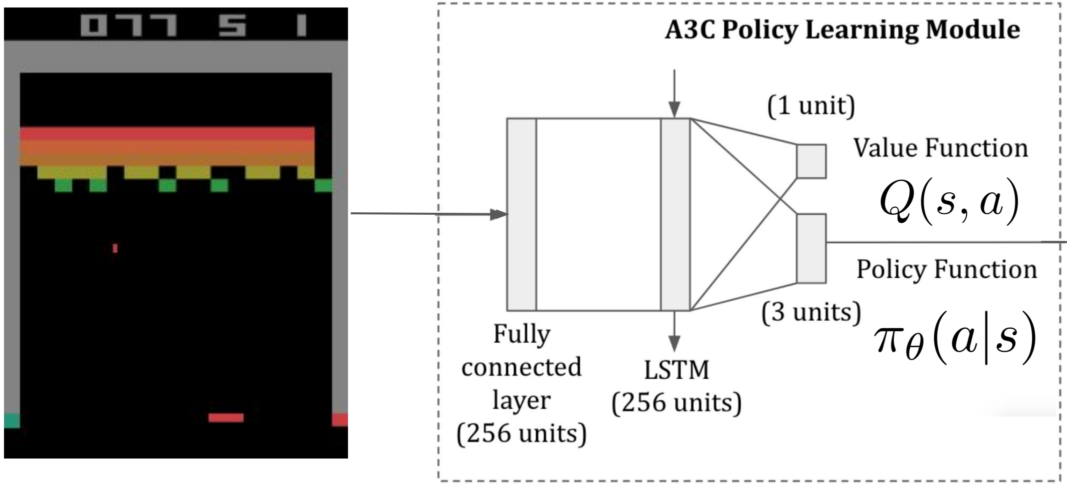
Minh et. al., ICML 2016

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



$\pi_\theta(a|s)$

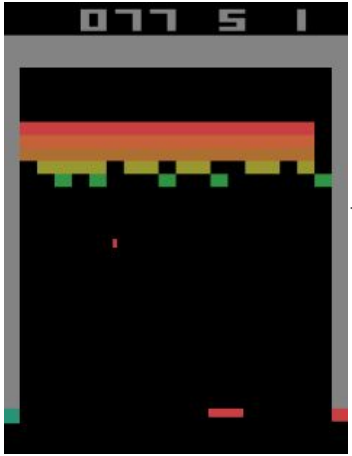
Actor: decides what actions to take

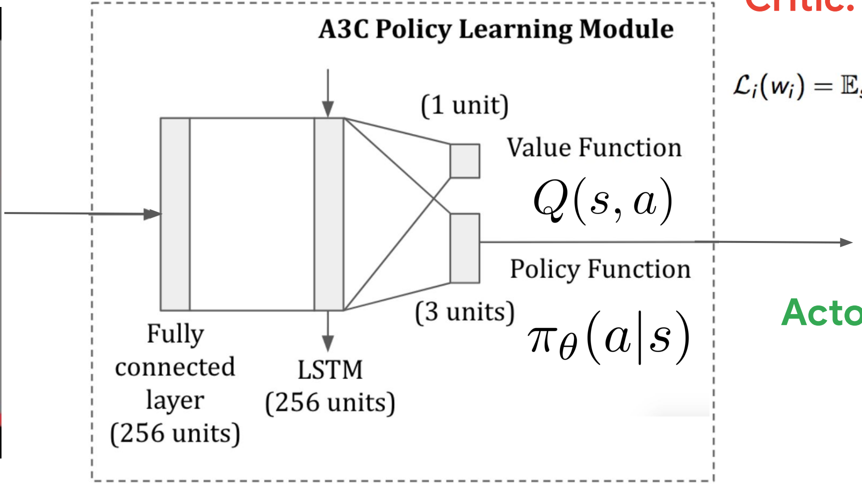
Minh et. al., ICML 2016

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

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

$\pi_\theta(a|s)$

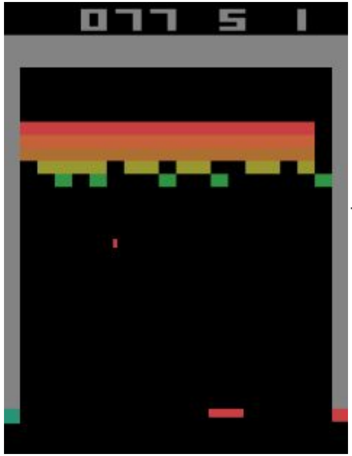
Actor: decides what actions to take

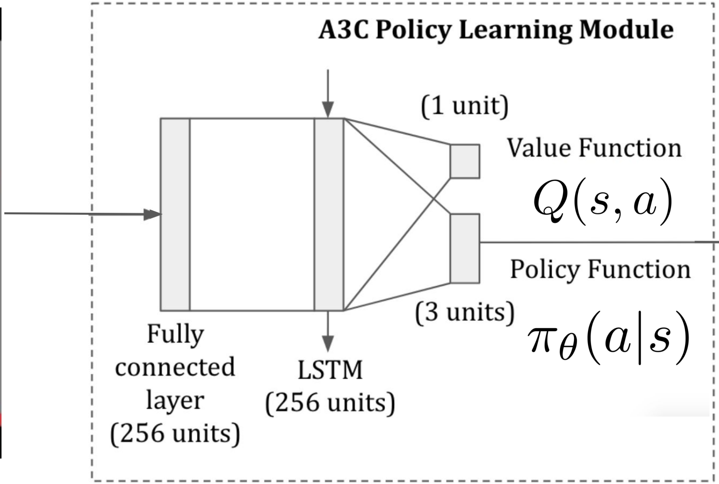
Minh et. al., ICML 2016

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).





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

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

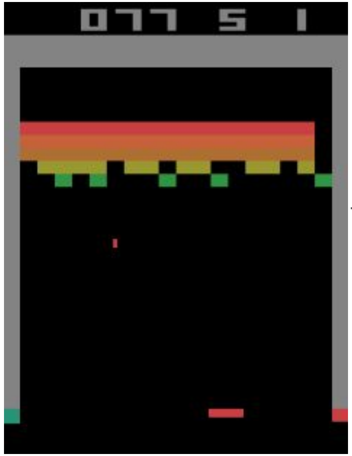
$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} (Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

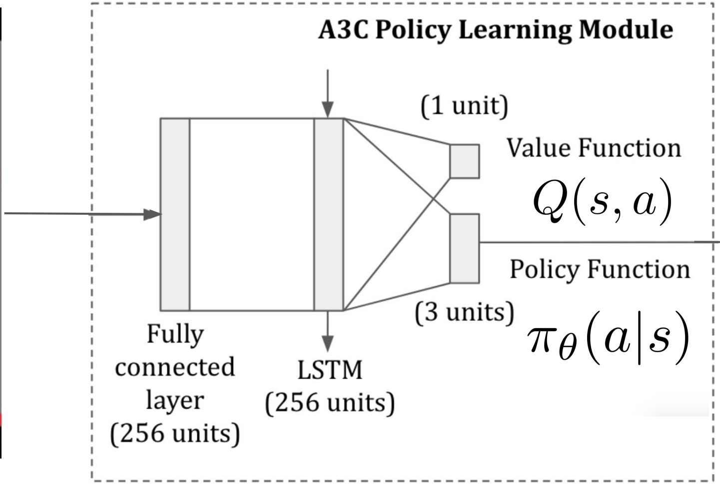
Minh et. al., ICML 2016

Actor-critic methods

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

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





$\pi_\theta(a|s)$

Exploration + experience replay
Decorrelate samples
Fixed targets

Minh et. al., ICML 2016

Summary of RL methods

Value iteration
Policy iteration
(Deep) Q-learning

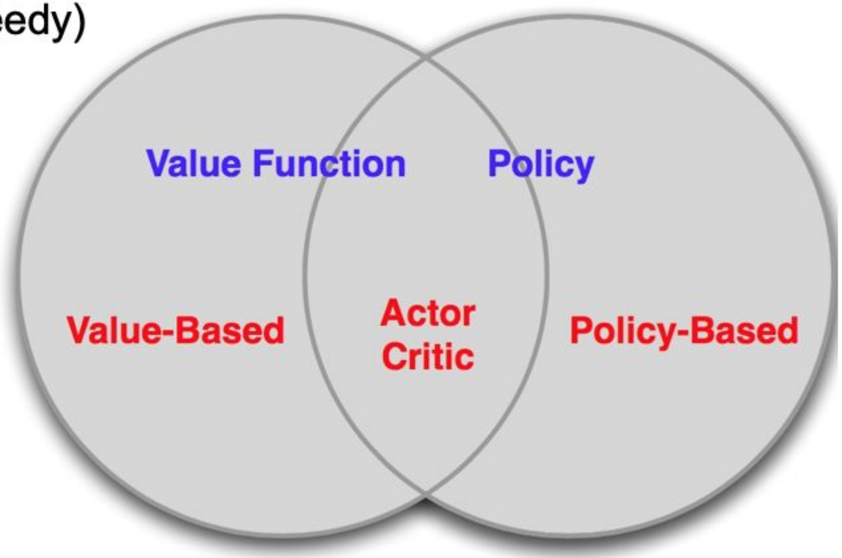
- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

Policy gradients

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

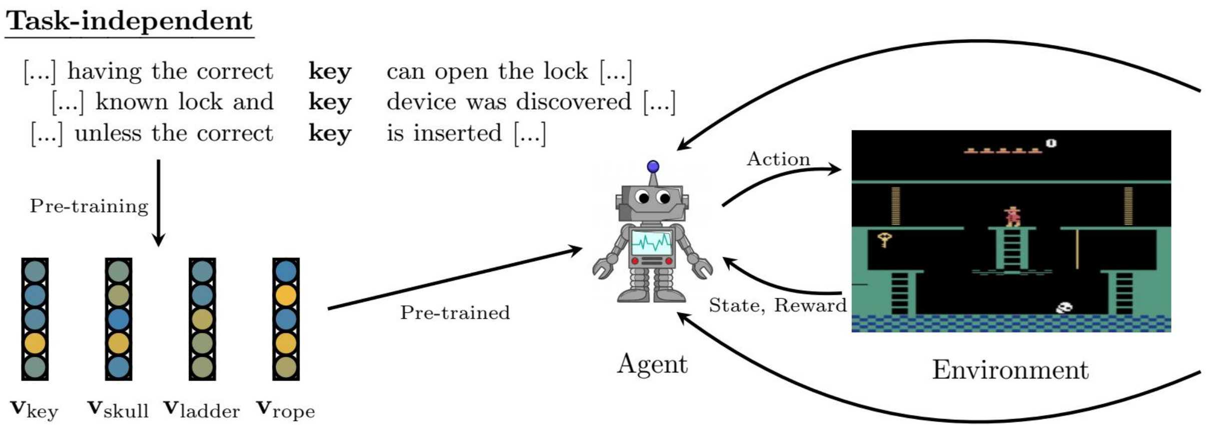
Actor (policy)
Critic (Q-values)

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



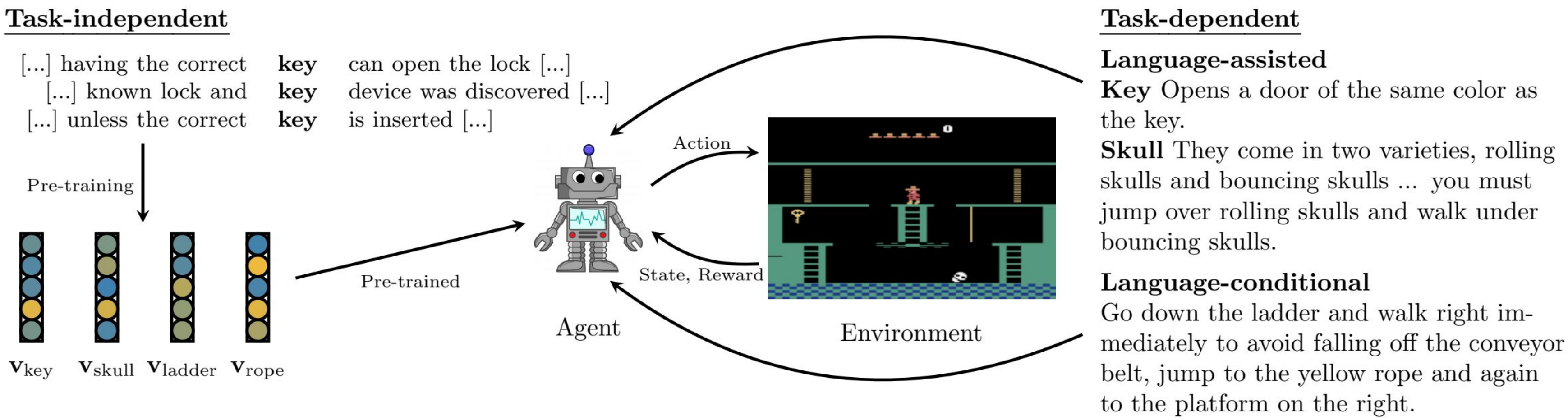
Applications: RL and Language

RL and Language



Luketina et. al., IJCAI 2019

RL and Language



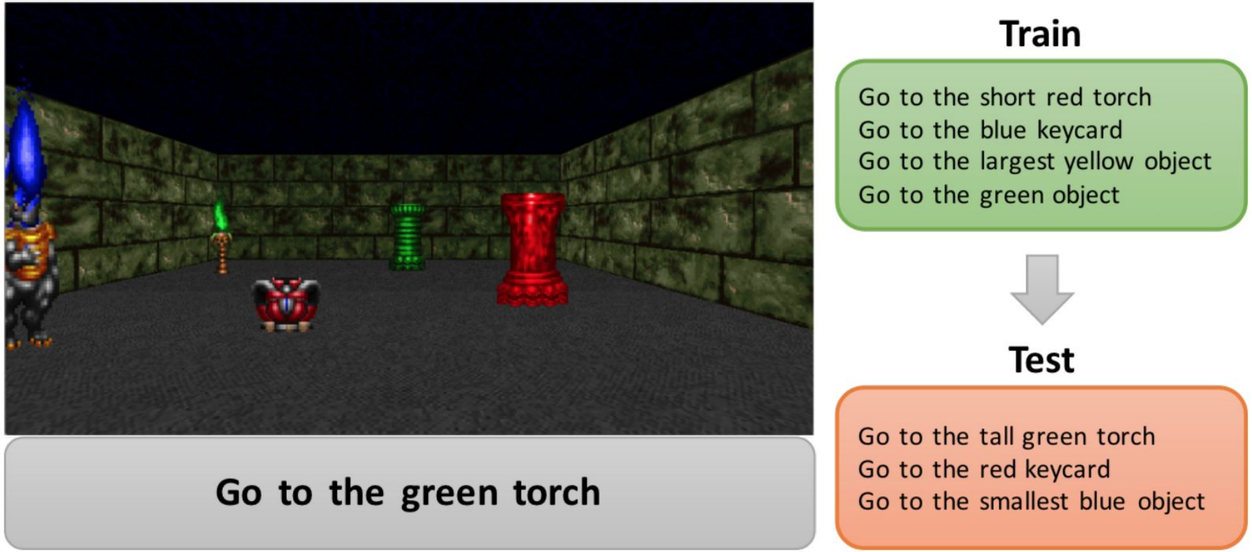
Luketina et. al., IJCAI 2019

Language-conditional RL

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

Language-conditional RL: Instruction following

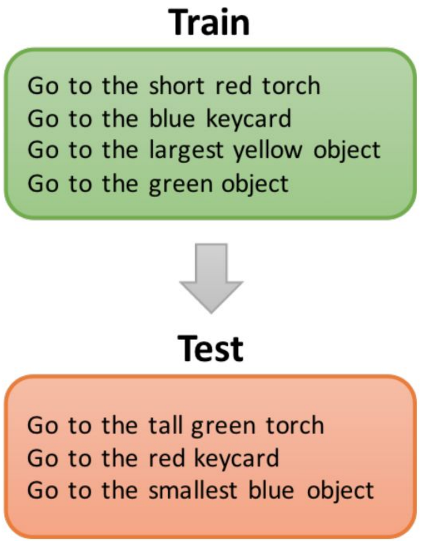
- Navigation via instruction following



Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017

Language-conditional RL: Instruction following

- Navigation via instruction following

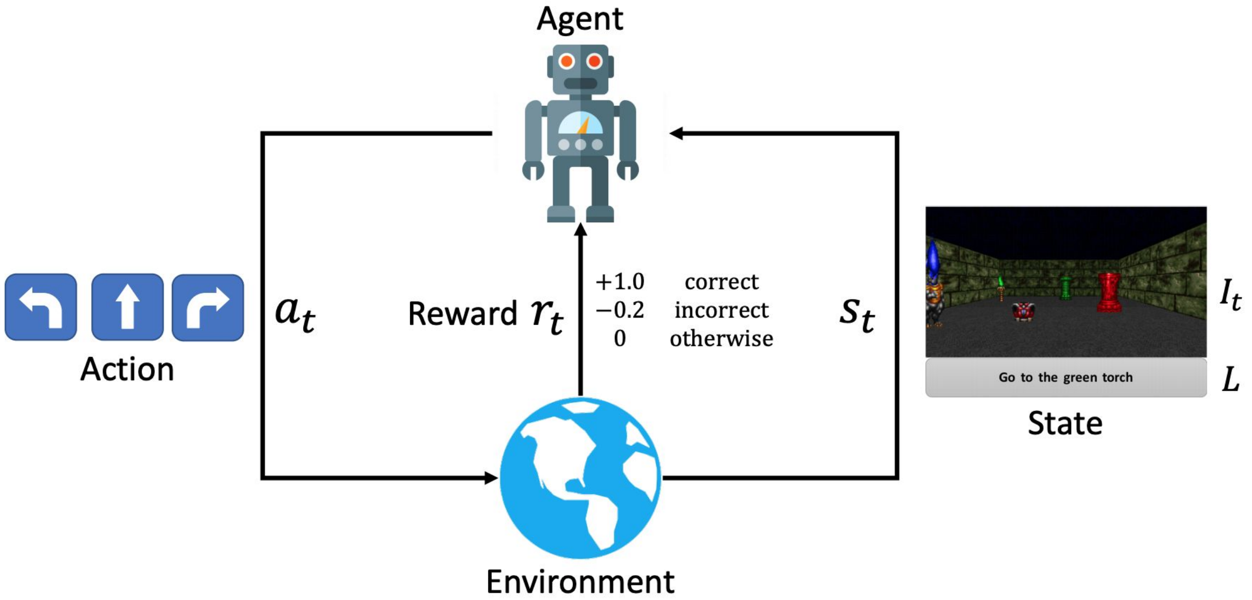


Fusion
Alignment
Ground language
Recognize objects
Navigate to objects
Generalize to unseen objects

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017

Language-conditional RL: Instruction following

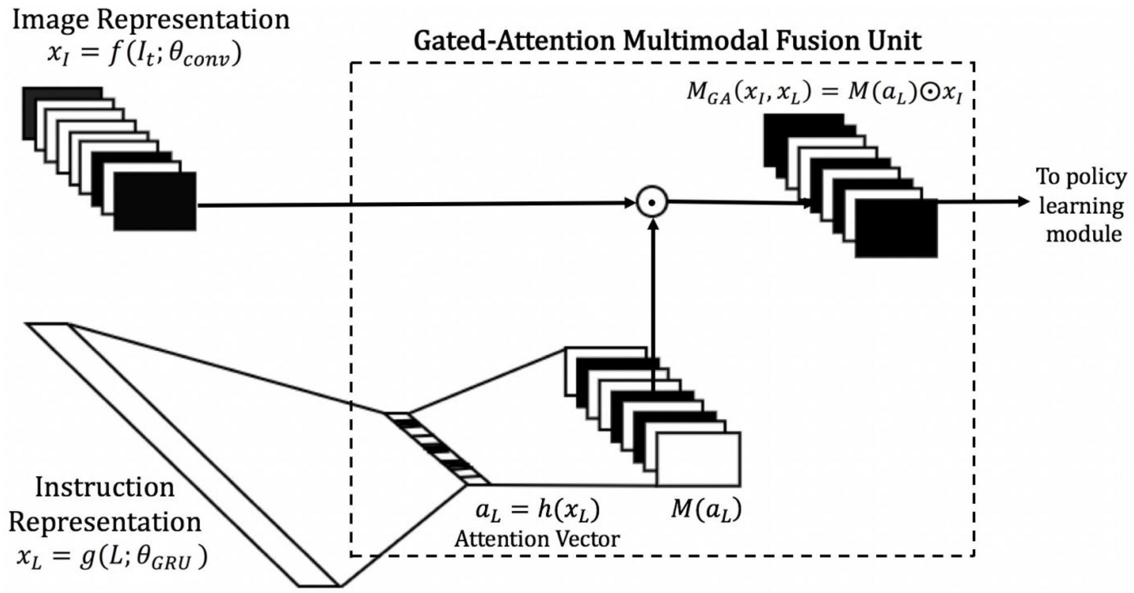
- Interaction with the environment



Chaplot et. al., AAAI 2018

Language-conditional RL: Instruction following

- Gated attention via element-wise product

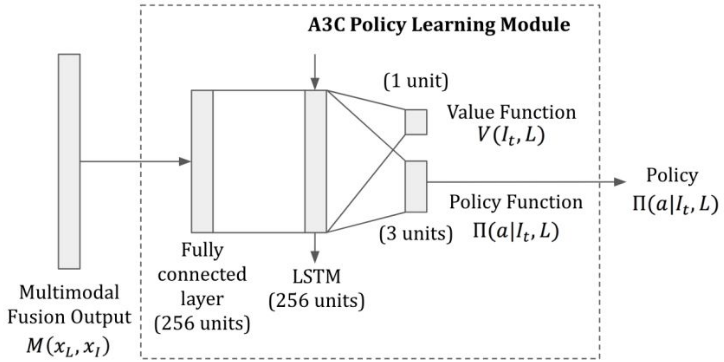


Fusion
Alignment
Ground language
Recognize objects

Chaplot et. al., AAAI 2018

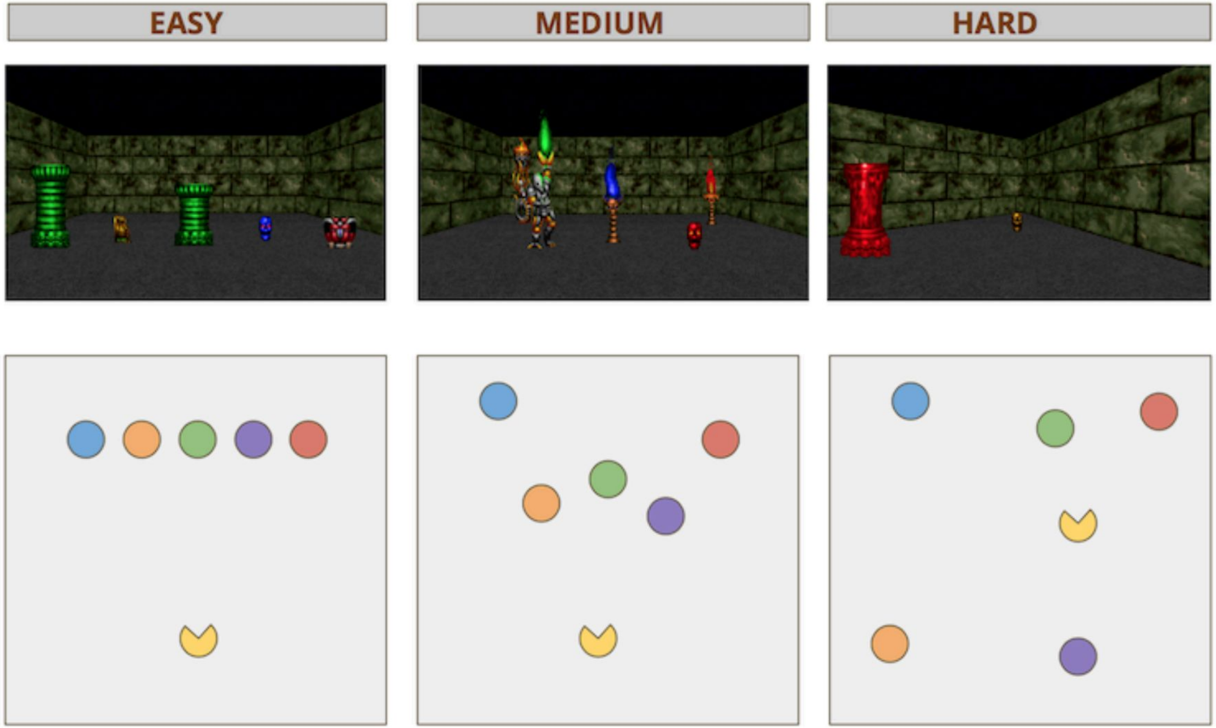
Language-conditional RL: Instruction following

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



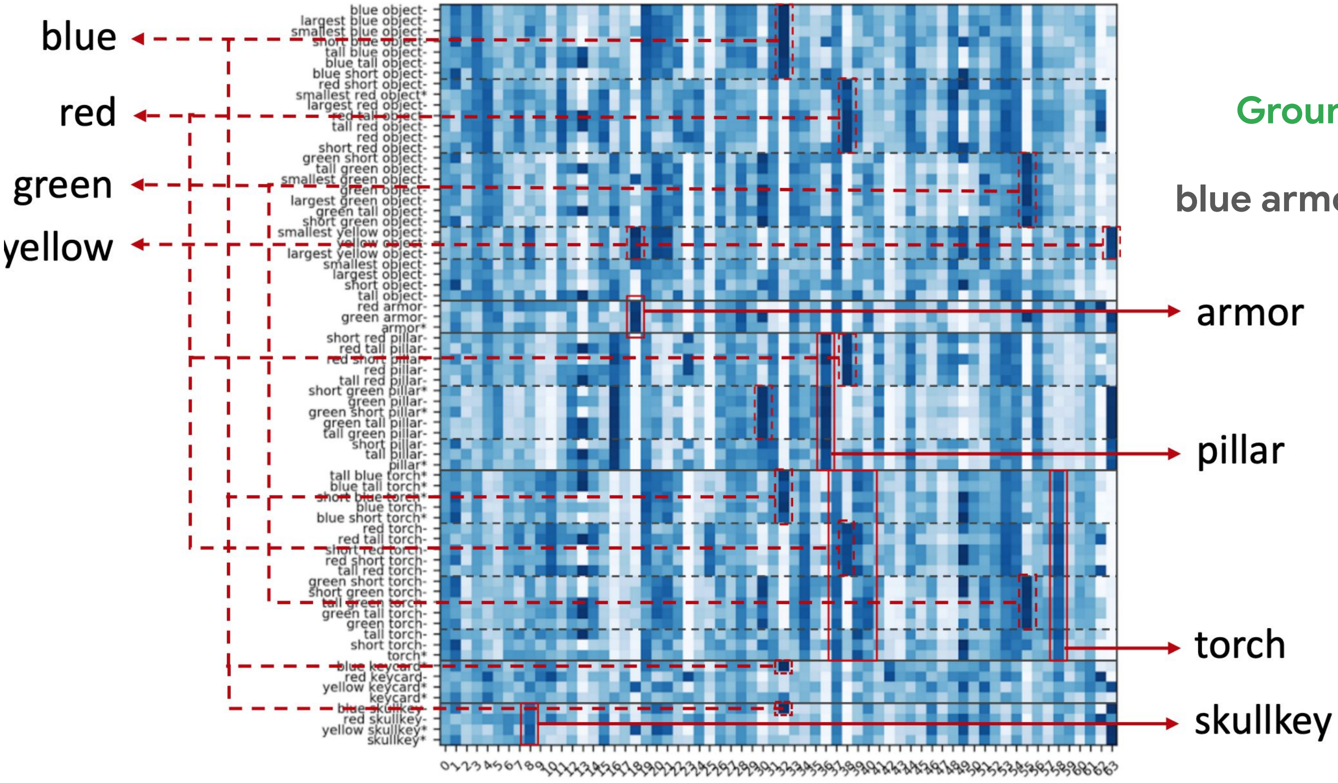
Chaplot et. al., AAAI 2018

Language-conditional RL: Instruction following



Chaplot et. al., AAAI 2018

Language-conditional RL: Instruction following

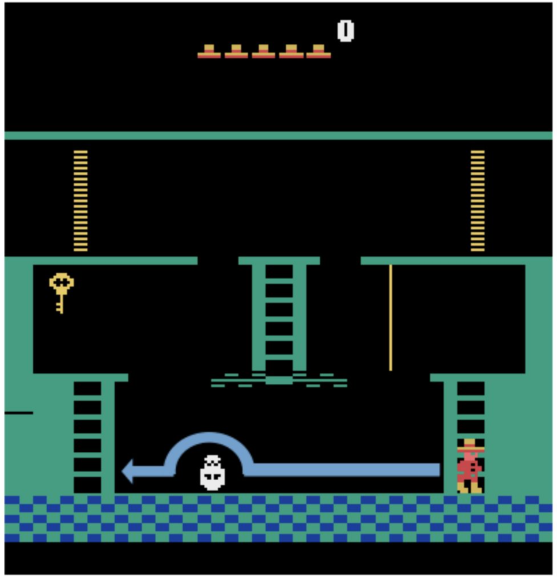


Grounding is important for generalization

blue armor, red pillar -> blue pillar

Chaplot et. al., AAAI 2018

Language-conditional RL: Rewards from instructions

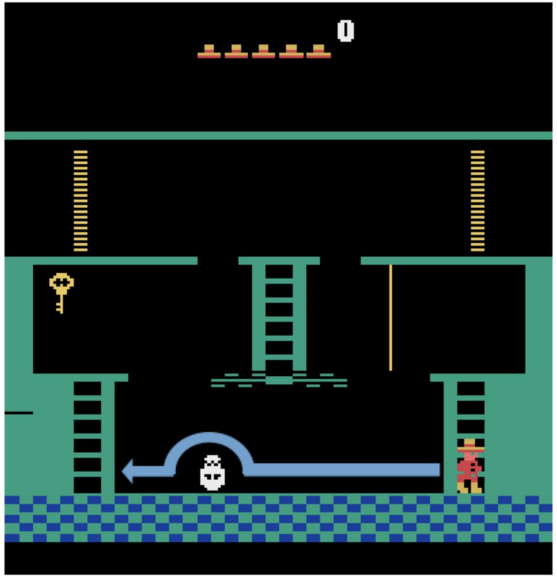


Montezuma's revenge

Sparse, long-term reward problem

General solution: reward shaping via auxiliary rewards

Language-conditional RL: Rewards from instructions



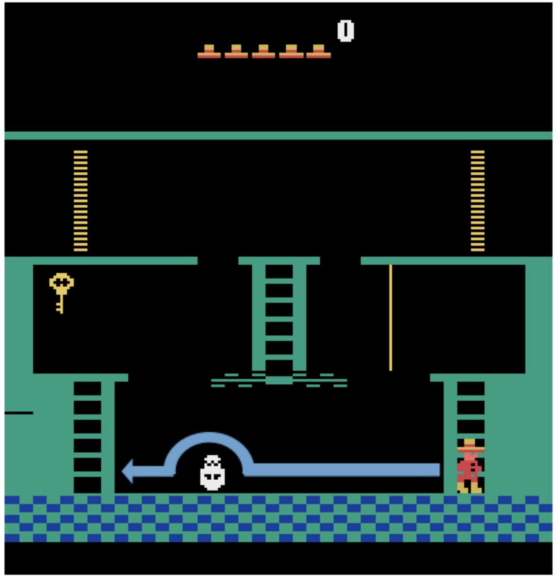
Montezuma's revenge

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

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

Pathak et. al., ICML 2017
Burda et. al., ICLR 2019

Language-conditional RL: Rewards from instructions



Montezuma's revenge

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

Natural language for reward shaping

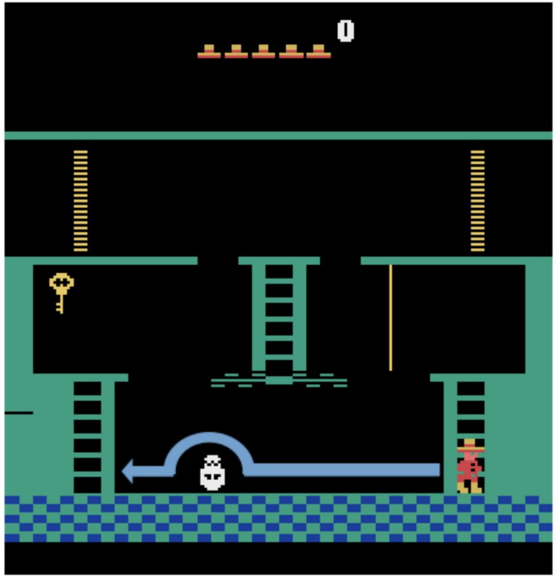
"Jump over the skull while going to the left"

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

Intermediate rewards to speed up learning

Goyal et. al., IJCAI 2019

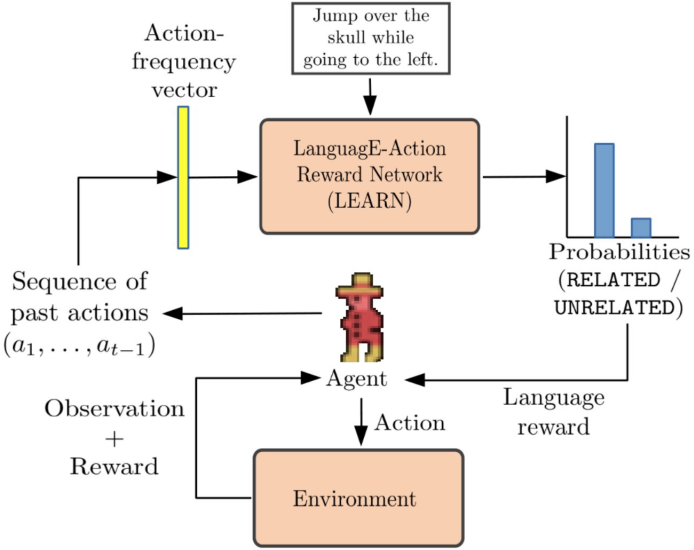
Language-conditional RL: Rewards from instructions



Montezuma's revenge

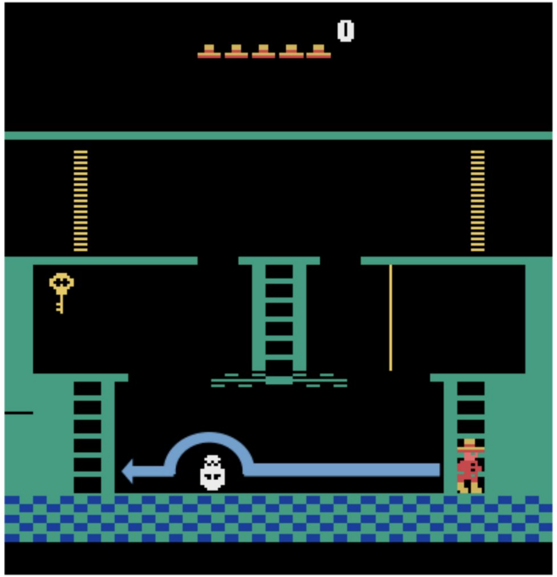
Natural language for reward shaping

Encourages agent to take actions related to the instructions



Goyal et. al., IJCAI 2019

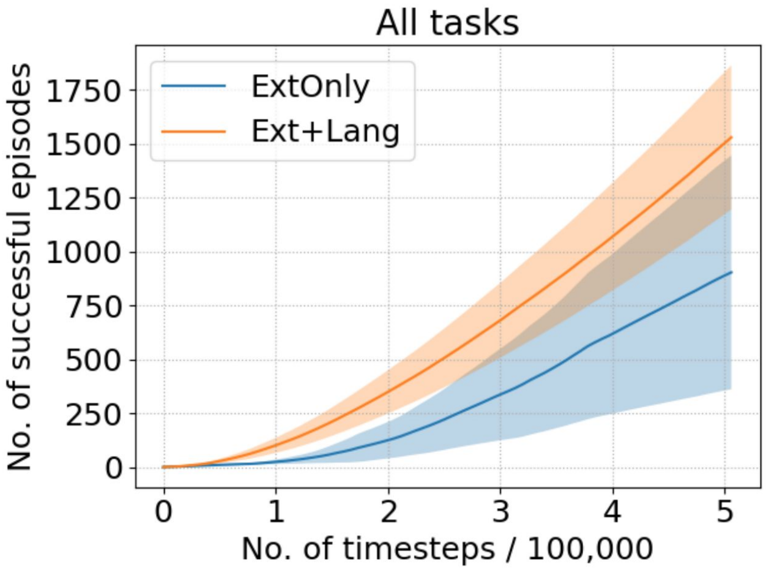
Language-conditional RL: Rewards from instructions



Montezuma's revenge

Natural language for reward shaping

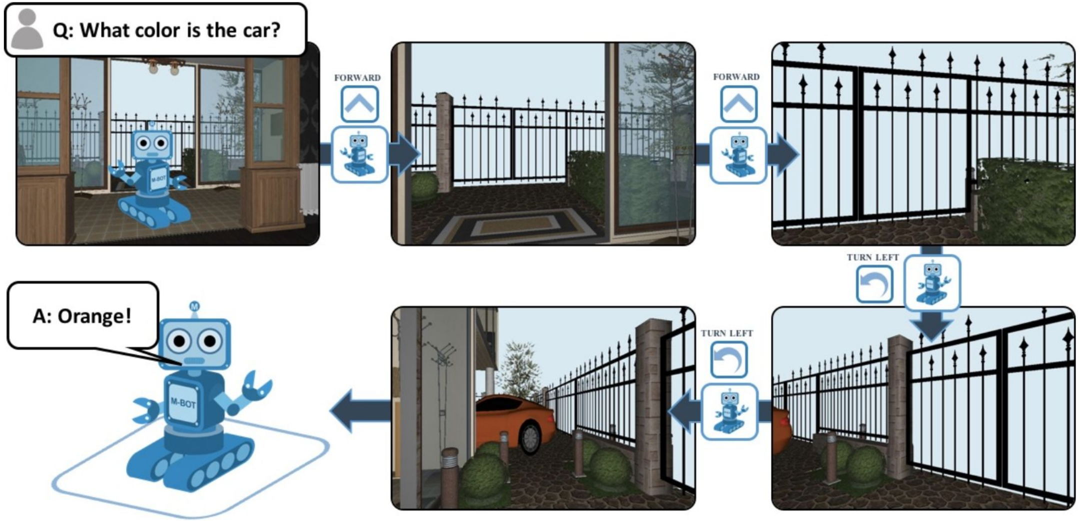
Encourages agent to take actions related to the instructions



Goyal et. al., IJCAI 2019

Language-conditional RL: Language in S and A

- Embodied QA: Navigation + QA



Most methods similar to instruction following

Das et. al., CVPR 2018

Language-assisted RL

- Language for communicating domain knowledge
- Language for structuring policies

Language-assisted RL: Domain knowledge

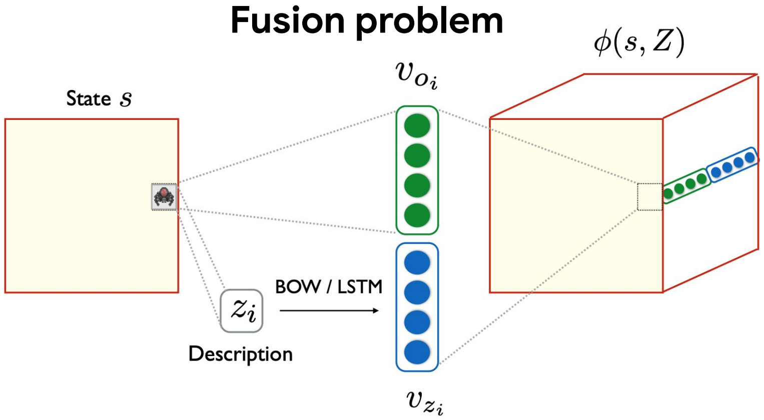
- Properties of entities in the environment are annotated by language

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

Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

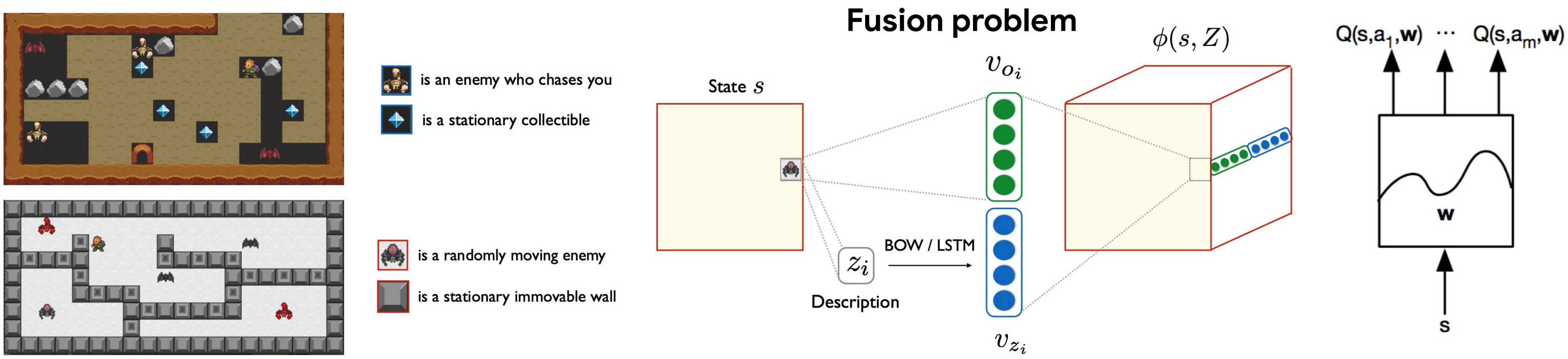
- Properties of entities in the environment are annotated by language



Narasimhan et. al., JAIR 2018

Language-assisted RL: Domain knowledge

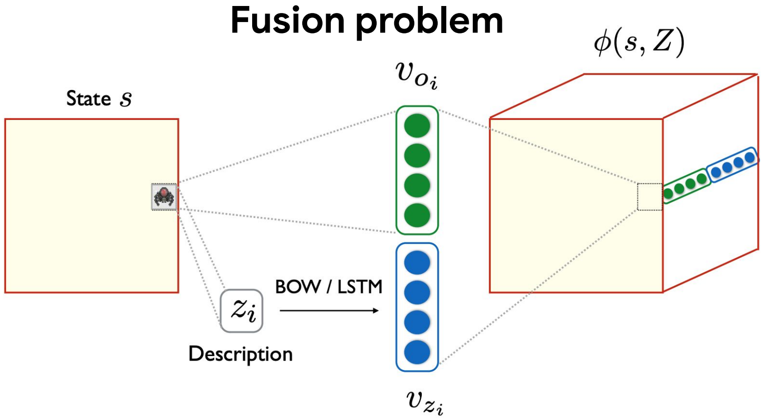
- Properties of entities in the environment are annotated by language

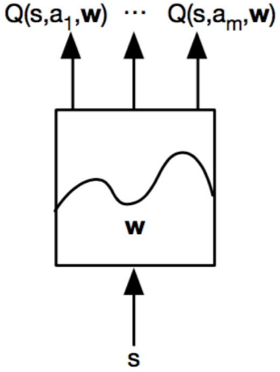


Narasimhan et. al., JAIR 2018

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

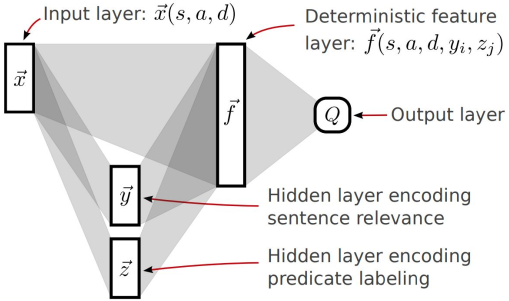
The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

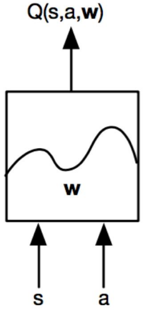
Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

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

- 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

Relevant sentences

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

A: action-description
S: state-description

Branavan et. al., JAIR 2012

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

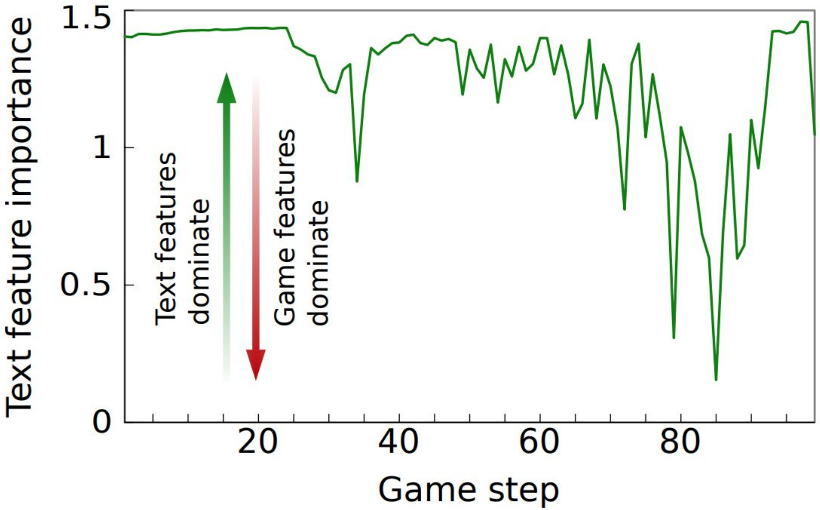
Method	% Win	% Loss	Std. Err.
Random	0	100	—
Built-in AI	0	0	—
Game only	17.3	5.3	± 2.7
Sentence relevance	46.7	2.8	± 3.5
Full model	**53.7**	5.9	± 3.5
Random text	40.3	4.3	± 3.4
Latent variable	26.1	3.7	± 3.1

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

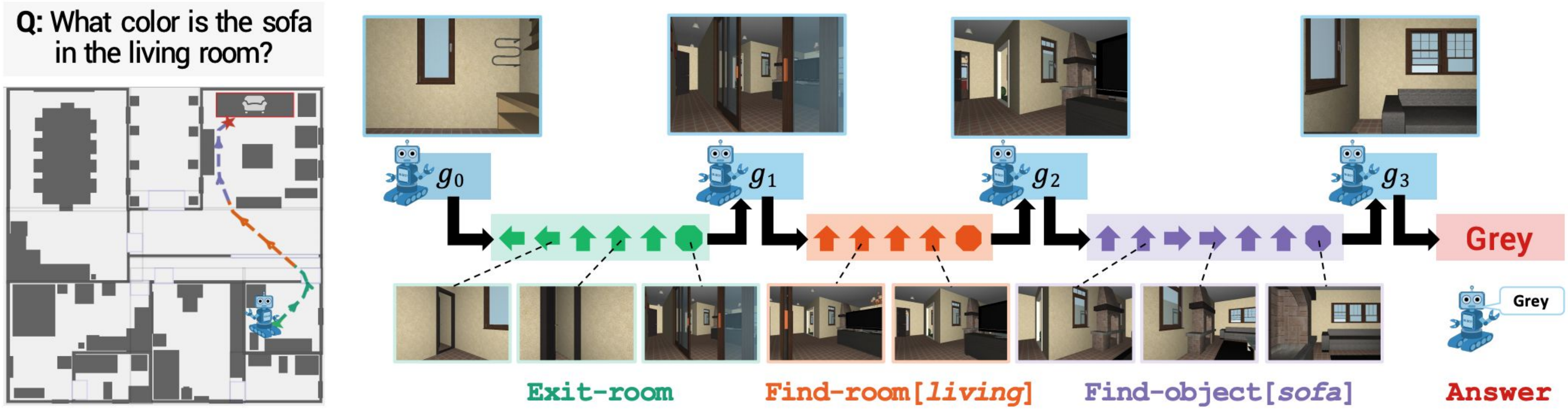


Language is most important at the start when you don't have a good policy
Afterwards, the model relies on game features

Branavan et. al., JAIR 2012

Language for structuring policies

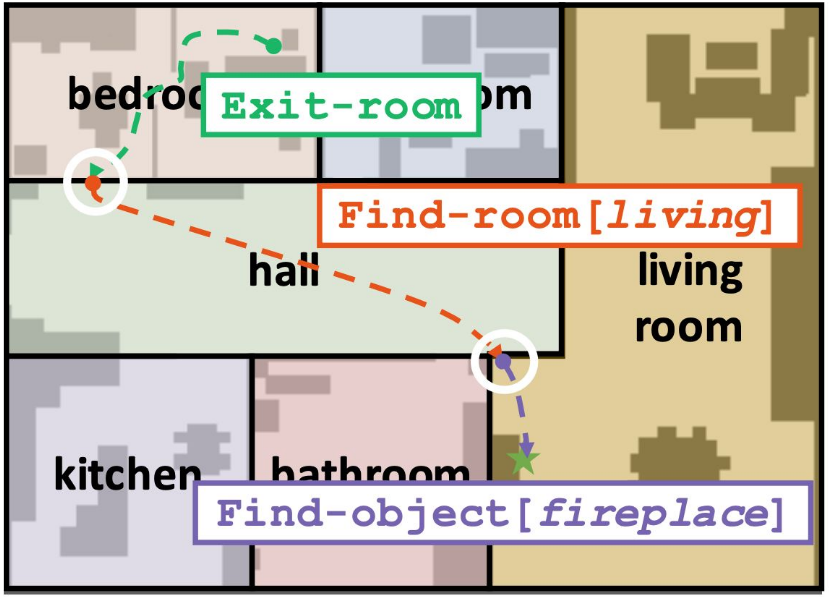
- Composing modules for Embodied QA

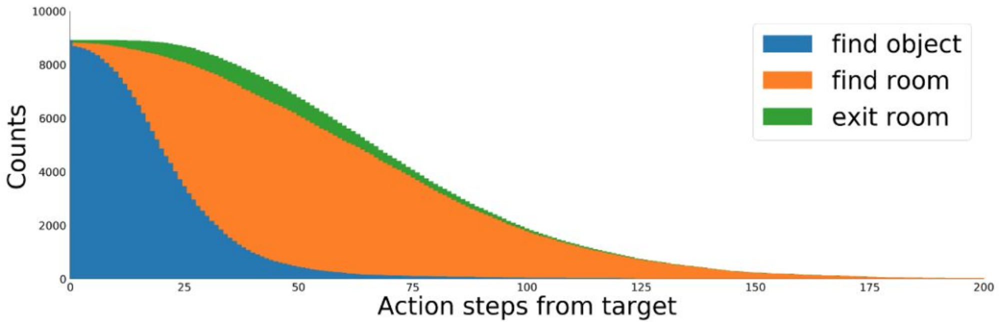


Das et. al., CoRL 2018

Language for structuring policies

- Composing modules for Embodied QA





Das et. al., CoRL 2018

Summary of RL methods

Value iteration
Policy iteration
(Deep) Q-learning

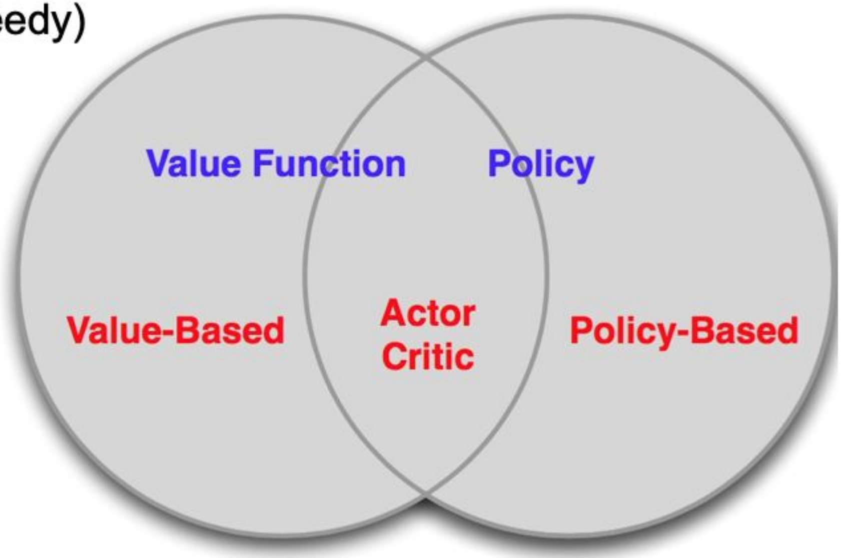
- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

Policy gradients

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

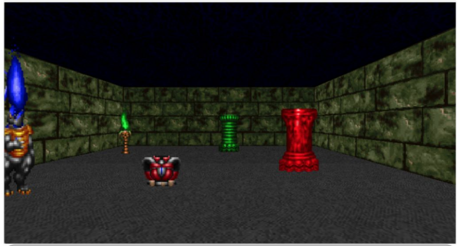
Actor (policy)
Critic (Q-values)

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



Summary of applications

Instruction following



Go to the green torch

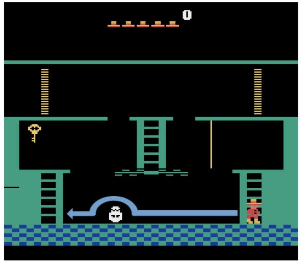
Train

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

Test

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

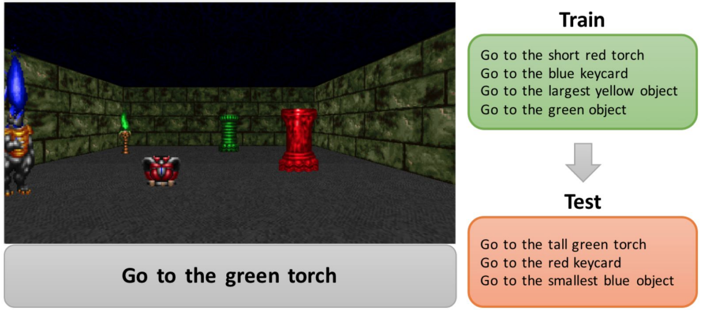
Language for rewards



“Jump over the skull while going to the left”

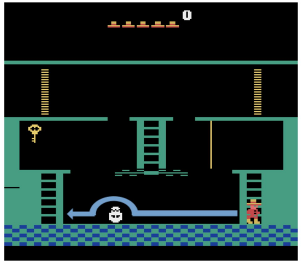
Summary of applications

Instruction following



Language as domain knowledge

Language for rewards



"Jump over the skull while going to the left"

Language to structure policies