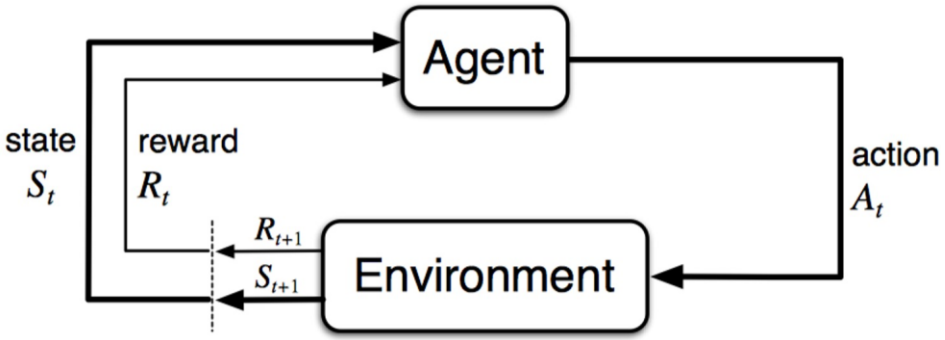
Reinforcement Learning

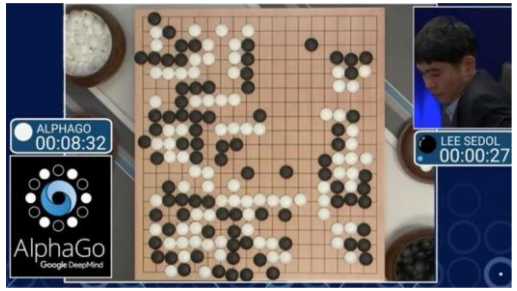
Dam Quang Tuan

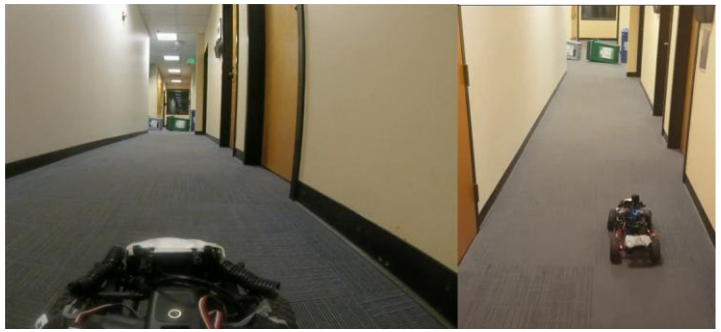
Learning a Policy – RL basics

Reinforcement learning

- Introduction to RL
- Markov Decision Processes (MDPs)
- Solving known MDPs using value and policy iteration
- Solving unknown MDPs using function approximation and Q-learning



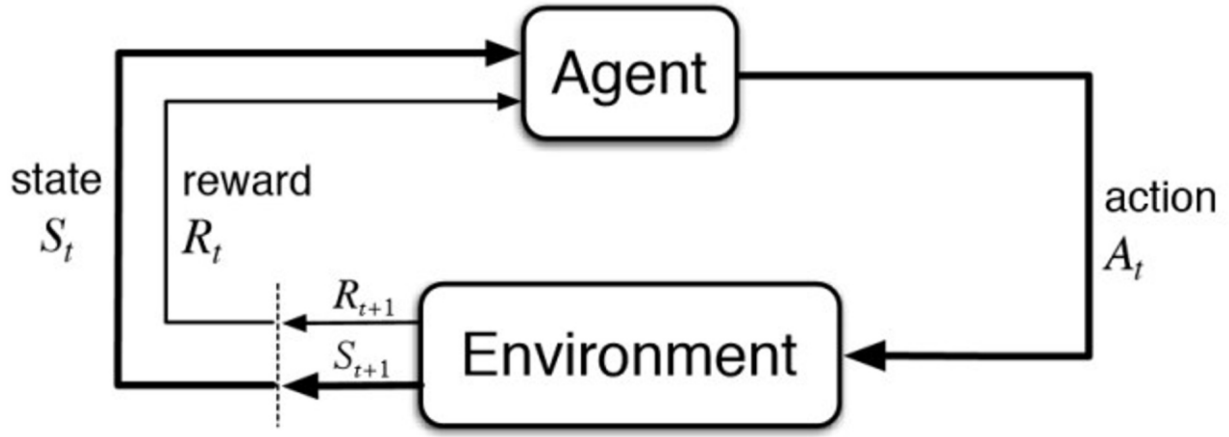




Learning a Policy – RL basics

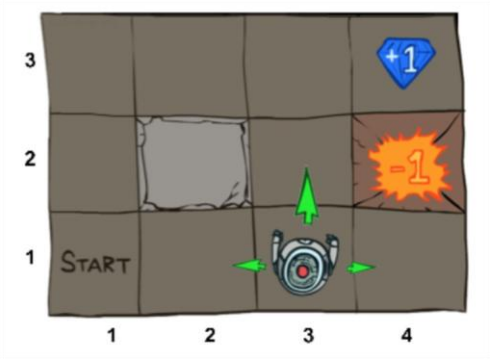
An MDP is defined by:

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

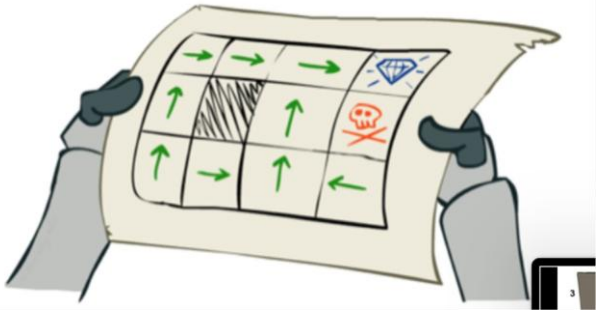


Return:

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



π:



Policy: $\pi(a|s) = \Pr(A_t = a|S_t = s) \quad \forall t$

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

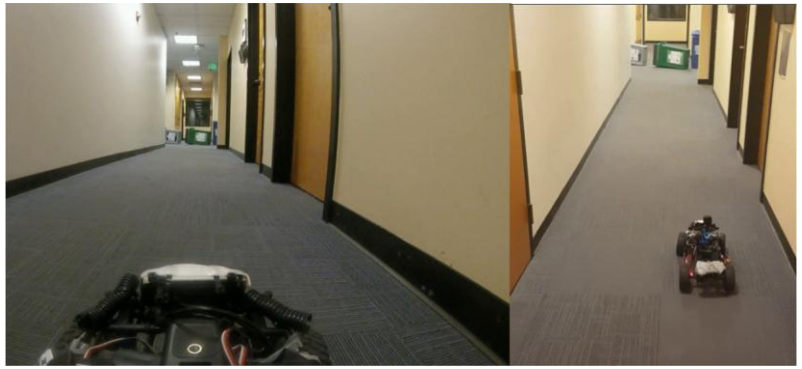
RL vs Supervised Learning

Reinforcement Learning

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

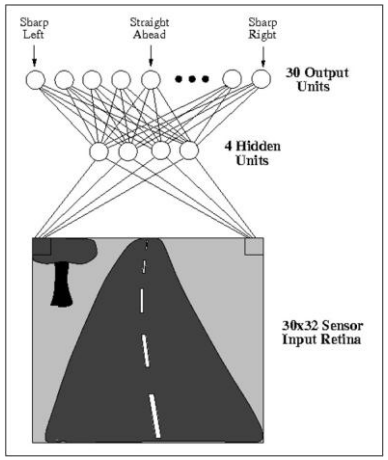
Supervised Learning

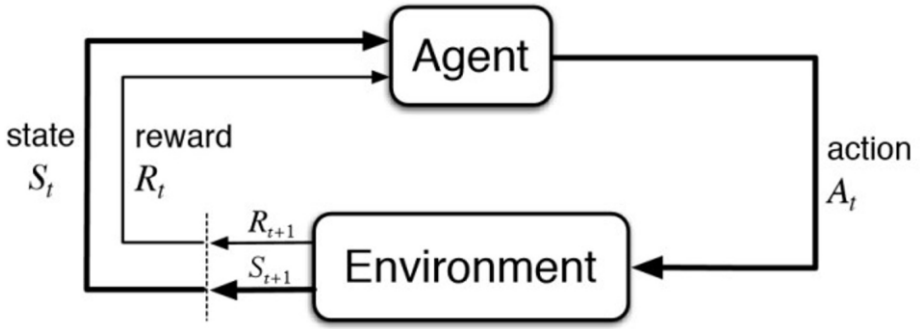
- One-step decision making
- Maximize immediate reward
- Dense supervision
- Environment always known



Intersection Between RL and Supervised Learning

Imitation learning





Obtain expert trajectories (e.g. human driver/video demonstrations):

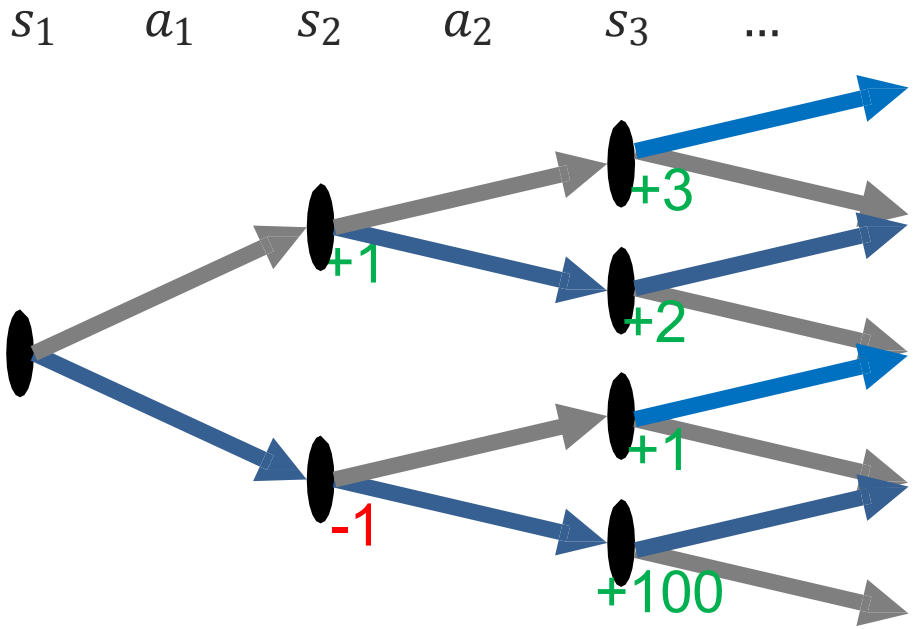
$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Perform supervised learning by predicting expert action

$$D = \{(s_0, a_0^*), (s_1, a_1^*), (s_2, a_2^*), \ldots\}$$

But: distribution mismatch between training and testing
Hard to recover from sub-optimal states
Sometimes not safe/possible to collect expert trajectories

RL as Exploring a Tree



π which action to take from each s

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | S_t = s\right] \qquad V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

State-value function: how much total reward should I expect following π from s?

$V^{\pi}(s_1) = 99$ $V^{\pi}(s_1) = 99$

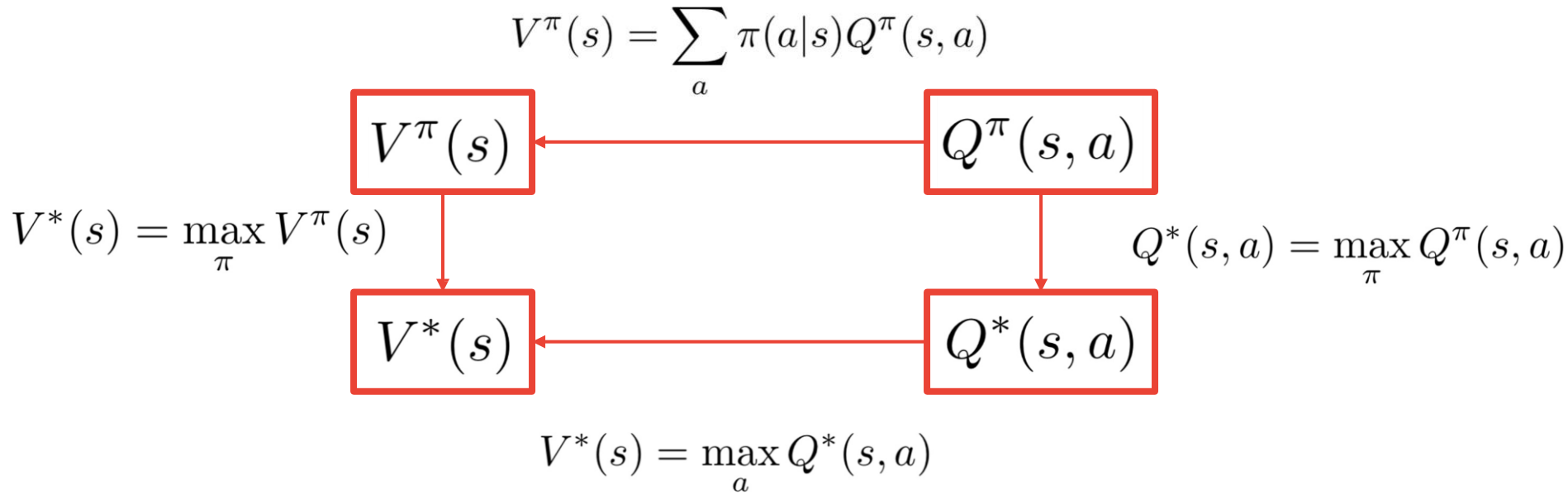
$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right] \quad Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Action-value function: how much total reward should I expect taking a, then following π, from s?

$Q^{\pi}(s_1, up) = 3$ $Q^{*}(s_1, up) = 4$

Relationships Between State and Action Values

State value functions **Action value functions**



Value-based Methods

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

State value functions

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

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

Action value functions

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

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

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

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

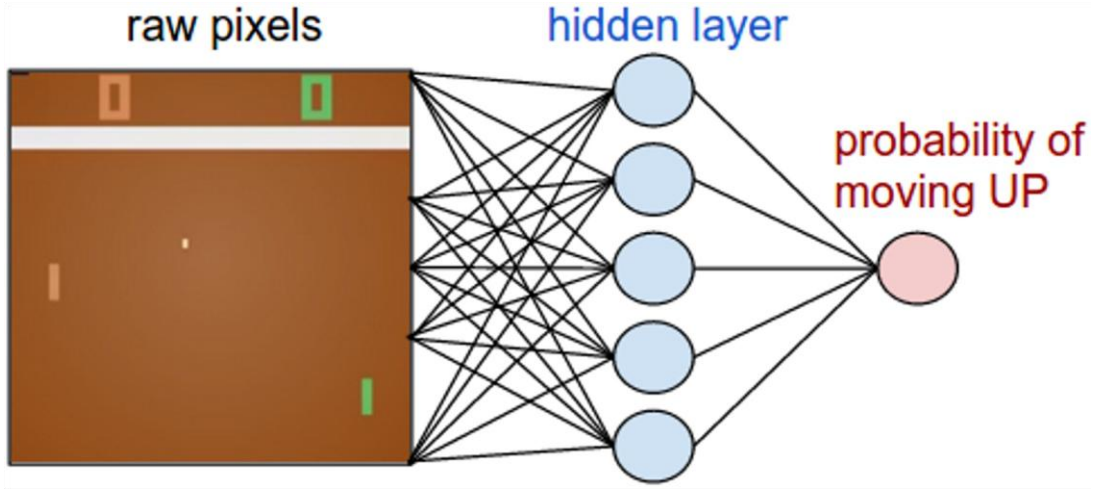
Optimal policy can also be found by maximizing over V*(s')
with **one-step look ahead**

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

Policy-based Methods

- Policy Based
  - No Value Function
  - Learned Policy

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



- 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 $\arg\max_a \ Q^*(s, a)$
  - Challenge for continuous / high-dimensional action spaces

Value-based vs Policy-based

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

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

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

Value-based

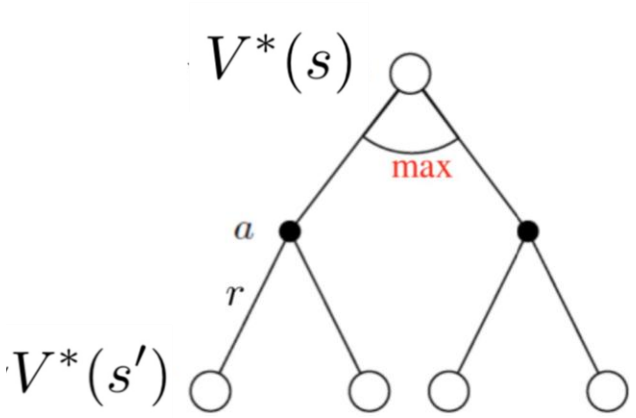
- More sample efficient, respects MDP structure
- Easier to add human knowledge about states and actions
- More complex algorithm
- Can't handle continuous argmax, harder to understand, sometimes values are more complex than policies

Policy-based

- Less sample efficient, more akin to trial-and-error
- Harder to add human knowledge
- Simpler algorithm
- Directly learns policy, can be more interpretable

Policy-based RL in 15 minutes

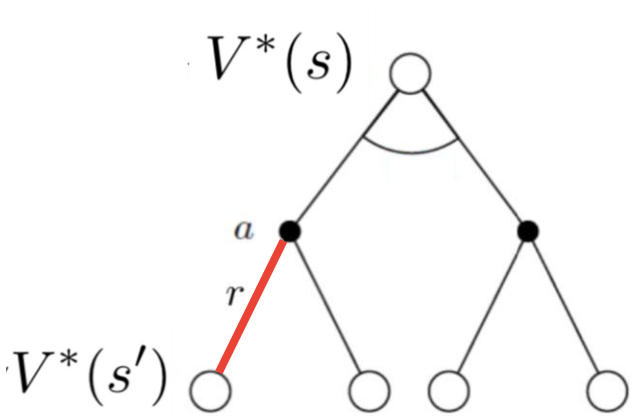
Recursive definition



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

Bellman Optimality for State Value Functions

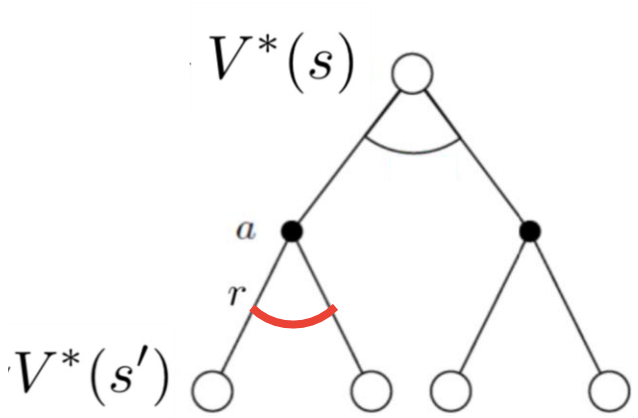
Recursive definition



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

Bellman Optimality for State Value Functions

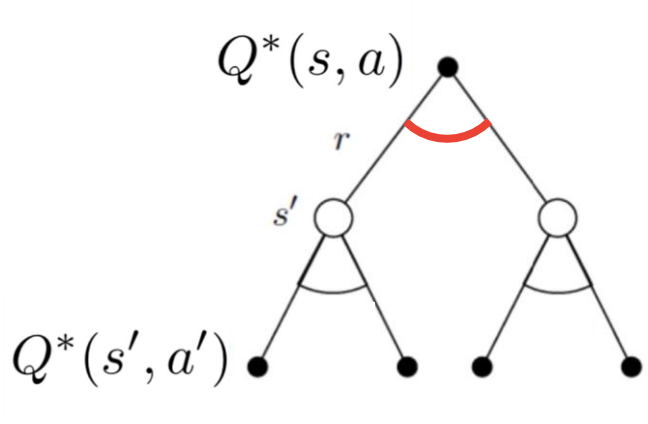
Recursive definition



$$
\begin{aligned}
V^*(s) &= \max_a Q^*(s,a) \\
&= \max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\
&= \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s'))\right]
\end{aligned}
$$

Bellman Optimality for Action Value Functions

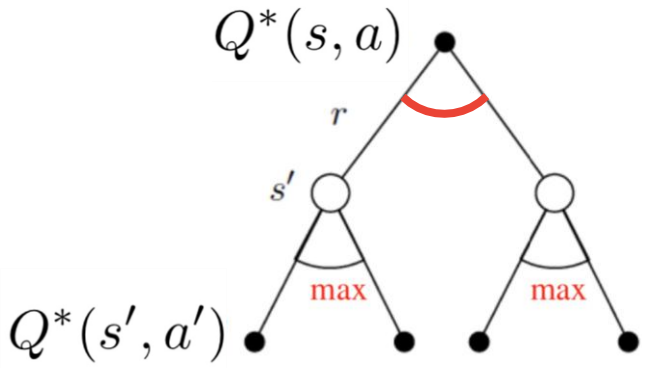
Recursive definition



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

Bellman Optimality for Action Value Functions

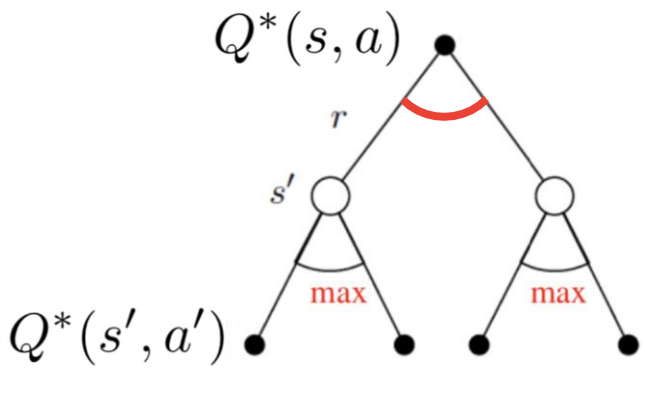
Recursive definition



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

Bellman Optimality for Action Value Functions

Recursive definition



$$
\begin{aligned}
Q^*(s,a) &= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\
&= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] \\
&= \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)
\end{aligned}
$$

Solving the Bellman Optimality Equations

Recursive definition

$$V^*(s) = \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s')) \right]$$

Solve by iterative methods

$$V^*_{[k+1]}(s) = \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*_{[k]}(s')) \right]$$

[Slides from Fragkiadaki, 10-703 CMU]

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

[Slides from Fragkiadaki, 10-703 CMU]

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

Find the best action according to one-step look ahead

This is called a value update or Bellman update/back-up

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

[Slides from Fragkiadaki, 10-703 CMU]

Q-Value Iteration

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

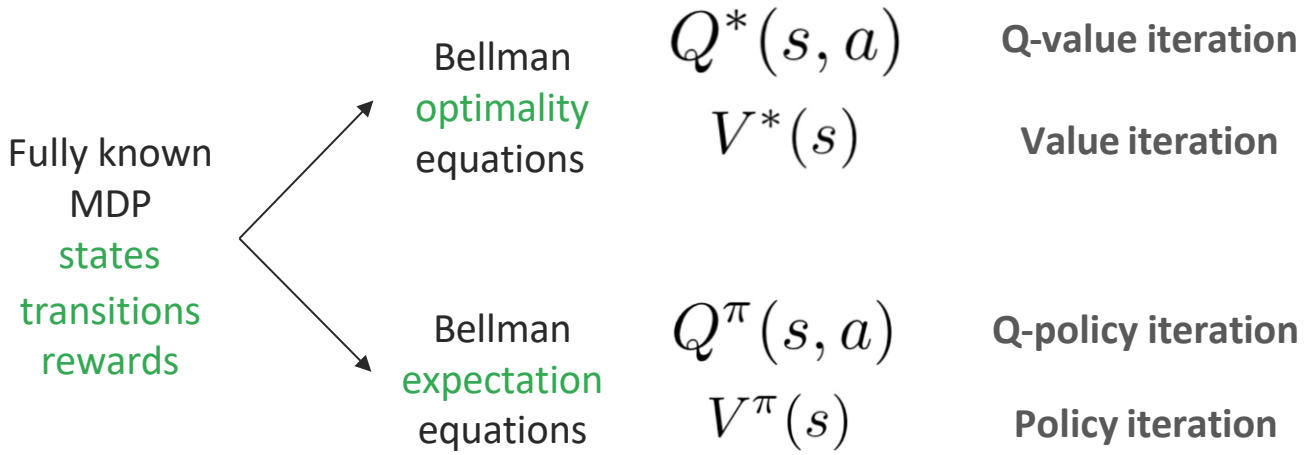
$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

[Slides from Fragkiadaki, 10-703 CMU]

Summary: Exact Methods



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

Limitations:

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

Unknown MDPs?

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

This is problematic when do not know the transitions

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$ **simulation and exploration**
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

$$\text{target}(s') = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

$$\text{error}(s') = \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right)$$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

learning rate

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

Key idea: implicitly estimate the transitions via simulation

Tabular Q-learning

Bellman optimality

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

Algorithm:

Start with $Q_0(s,a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = r(s,a,s')$$

Sample new initial state s'

else:

$$\text{target} = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

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

$$s \leftarrow s'$$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = r(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

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

$$s \leftarrow s'$$

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

[Slides from Fragkiadaki, 10-703 CMU]

Exploration and Exploitation

Poor estimates of Q(s,a) at the start:

Bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further.

$$\pi(s) = \begin{cases} \max_a \hat{Q}(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

Gradually decrease epsilon as policy is learned.

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = r(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

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

$$s \leftarrow s'$$

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?

- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

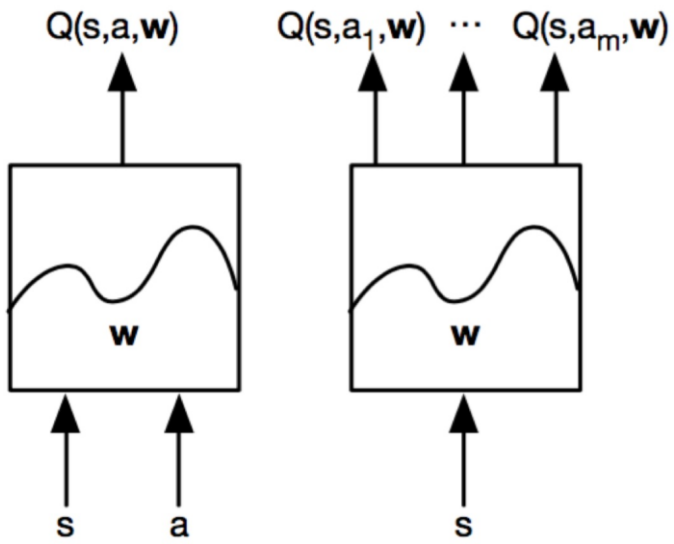
[Slides from Fragkiadaki, 10-703 CMU]

Deep Q-learning

Q-learning with function approximation to **extract informative features** from **high-dimensional** input states.

Represent value function by Q network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



+ high-dimensional, continuous states
+ generalization to new states

[Slides from Fragkiadaki, 10-703 CMU]

Deep Q-learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as as a target

- Minimize MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

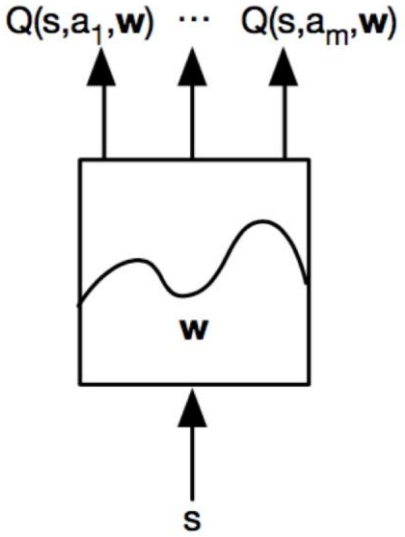
Deep Q-learning Challenges

- Minimize MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_a Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

- Converges to Q* using table lookup representation

- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets



Deep Q-learning: Experience Replay

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

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

$\rightarrow \quad s, a, r, s'$

exploration, epsilon greedy is important!

- Sample random mini-batch of transitions (s,a,r,s') from **D**

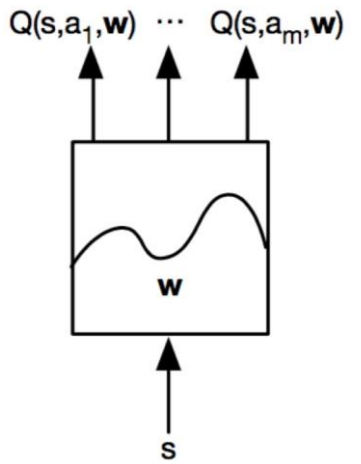
Deep Q-learning: Fixed Q-targets

- Sample random mini-batch of transitions (s,a,r,s') from D
- Compute Q-learning targets w.r.t. old fixed parameters **w-**

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

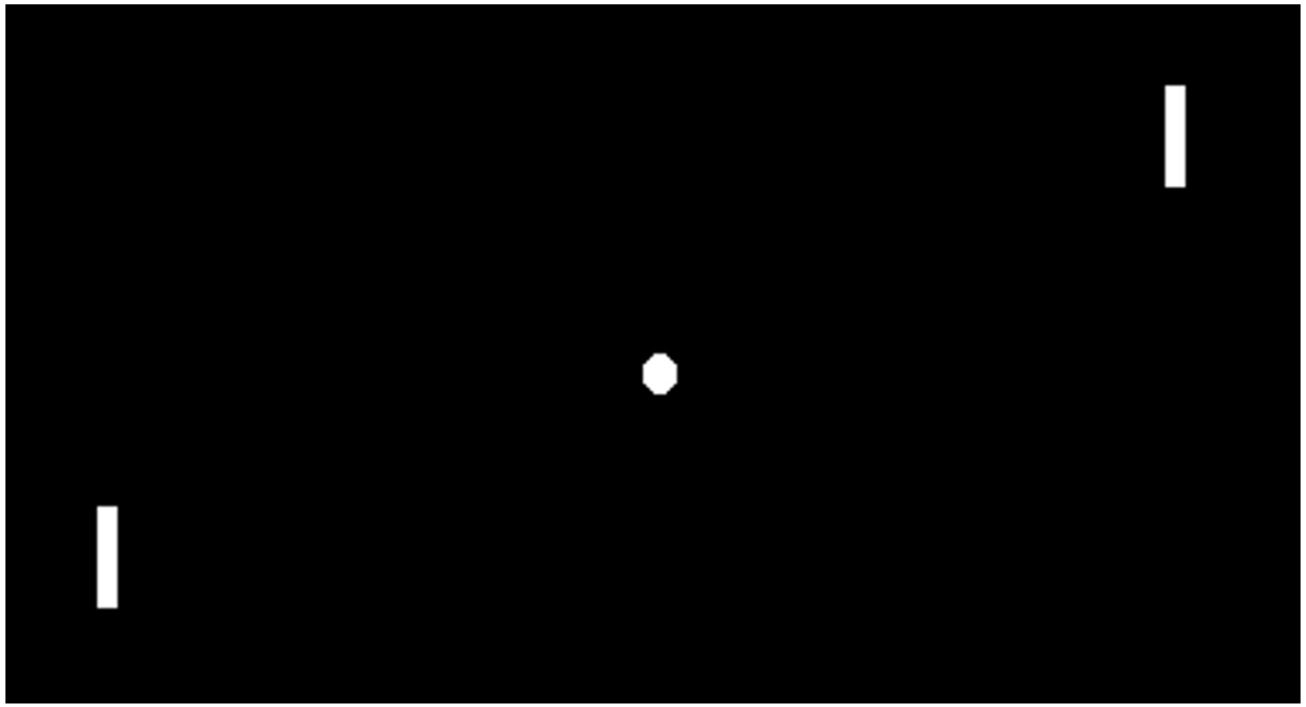
- Optimize MSE between Q-network and Q-learning targets

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



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

Policy-based RL in 15 minutes

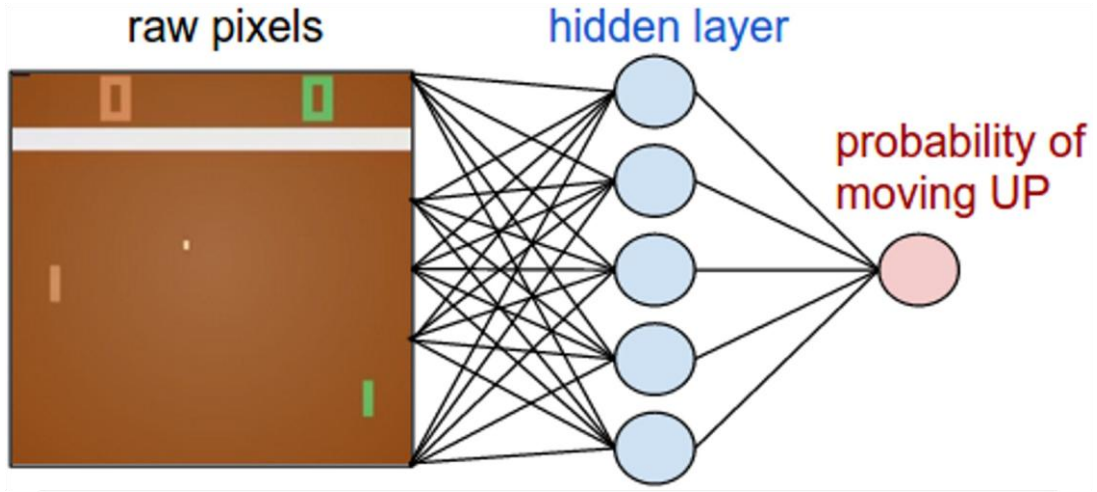


Pong from Pixels

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?

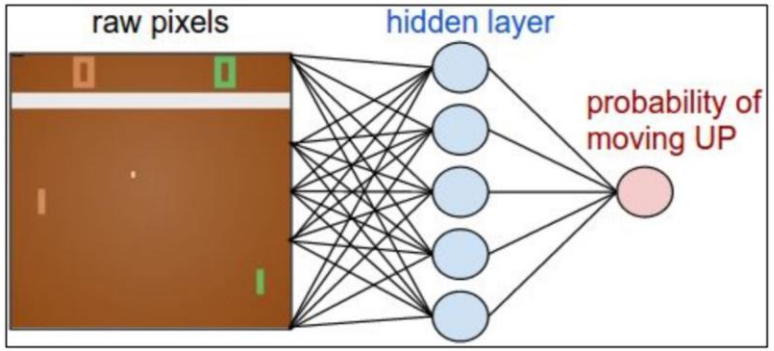
Pong from Pixels

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

(x1,UP)
(x2,DOWN)
(x3,UP)
…

maximize:

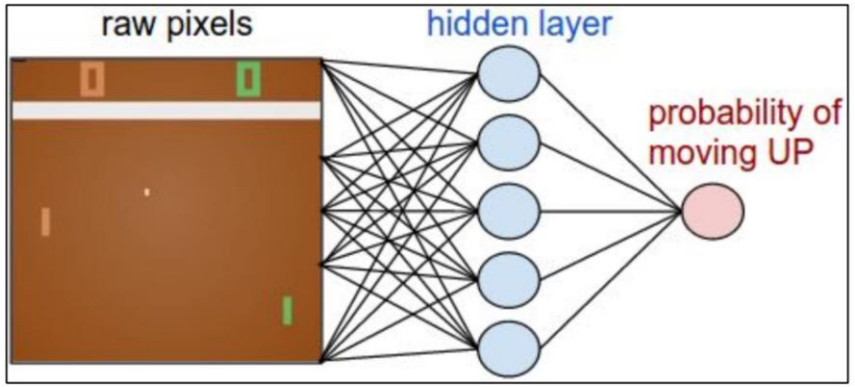
$$\sum_i \log p(y_i | x_i)$$



[Slides from Karpathy]

Pong from Pixels

Except, we don't have labels...

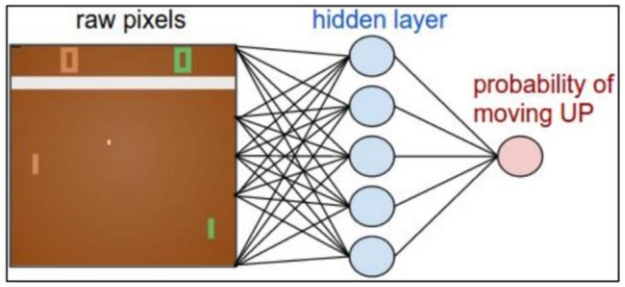


Should we go UP or DOWN?

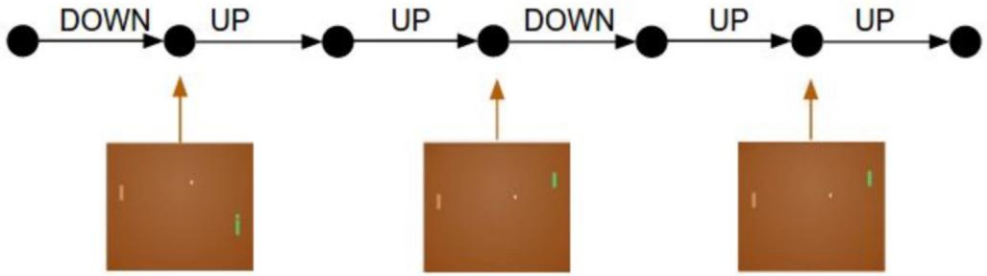
[Slides from Karpathy]

Pong from Pixels

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



Rollout the policy and collect an episode



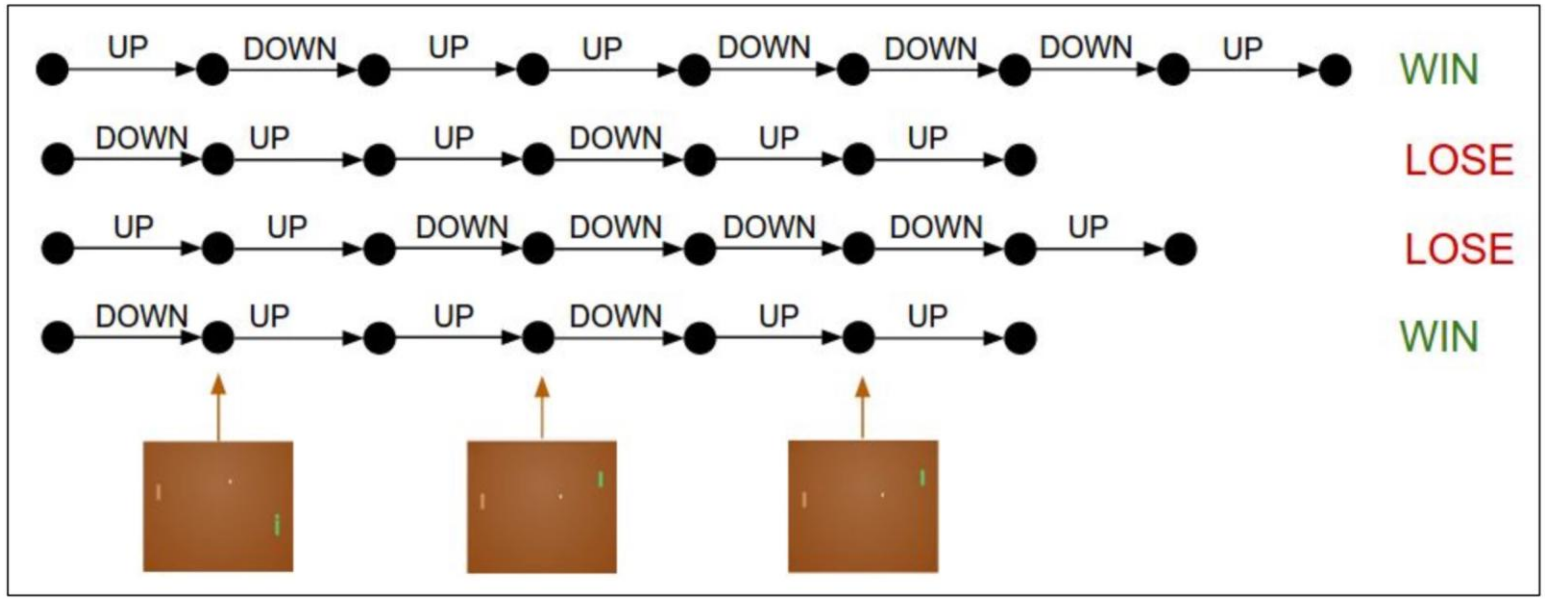
WIN

[Slides from Karpathy]

Pong from Pixels

Collect many rollouts...

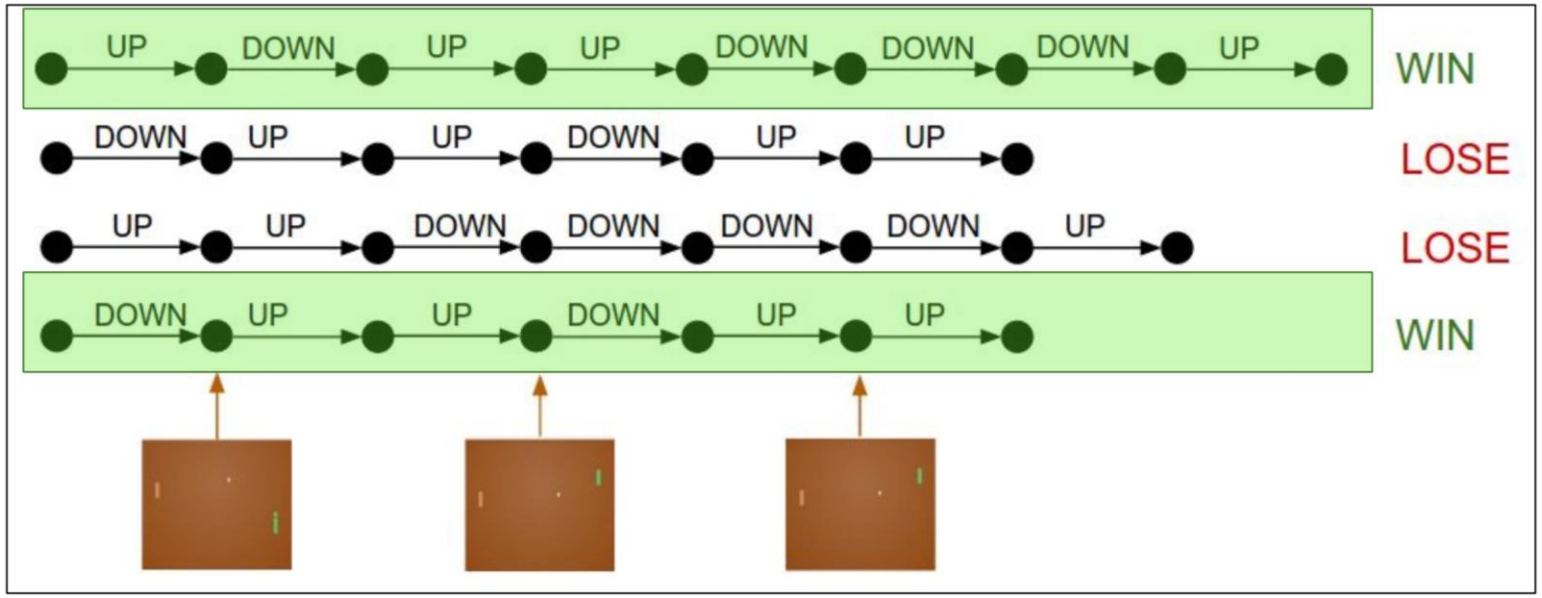
4 rollouts:



[Slides from Karpathy]

Pong from Pixels

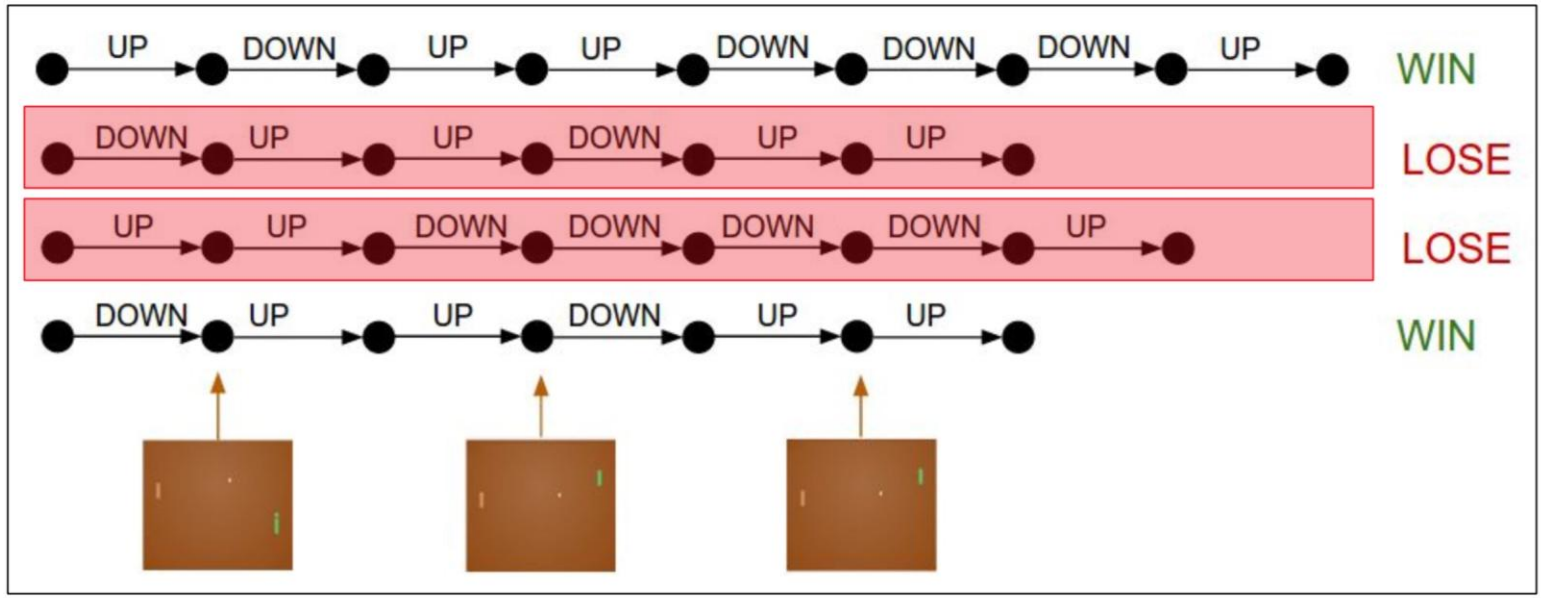
Not sure whatever we did here, but apparently it was good.



[Slides from Karpathy]

Pong from Pixels

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



[Slides from Karpathy]

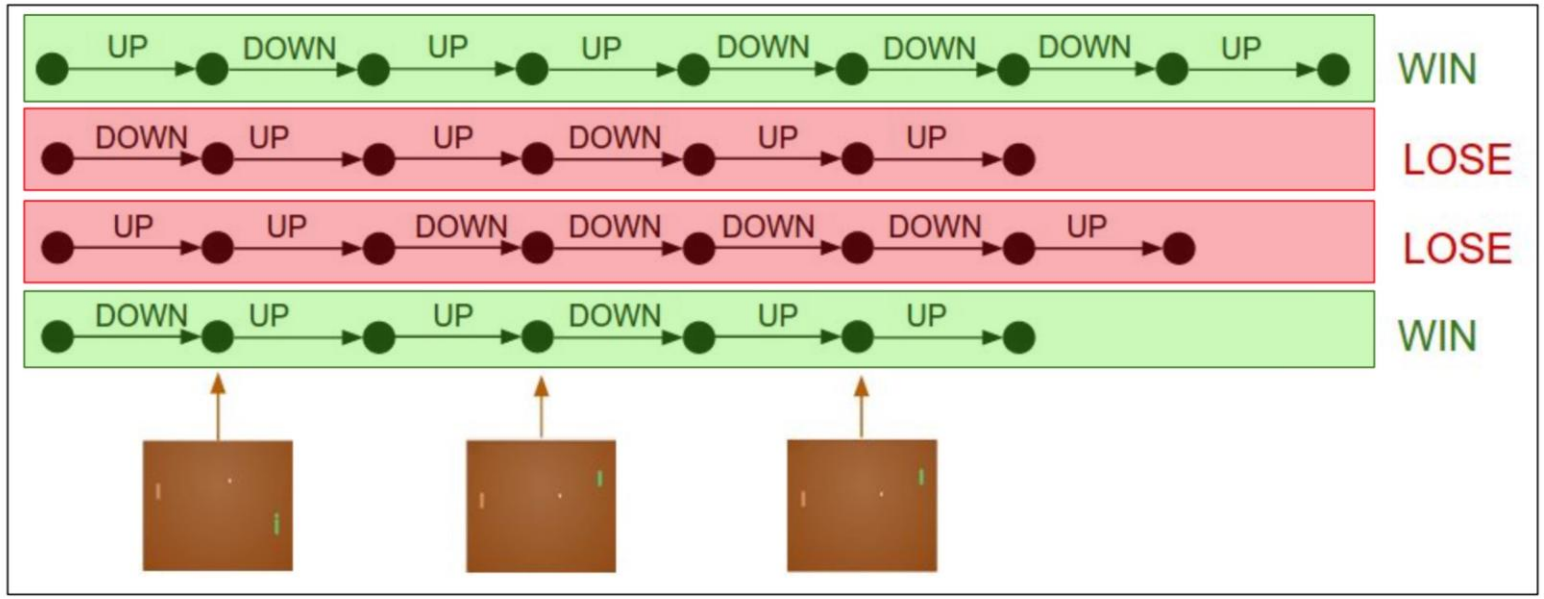
Pong from Pixels

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

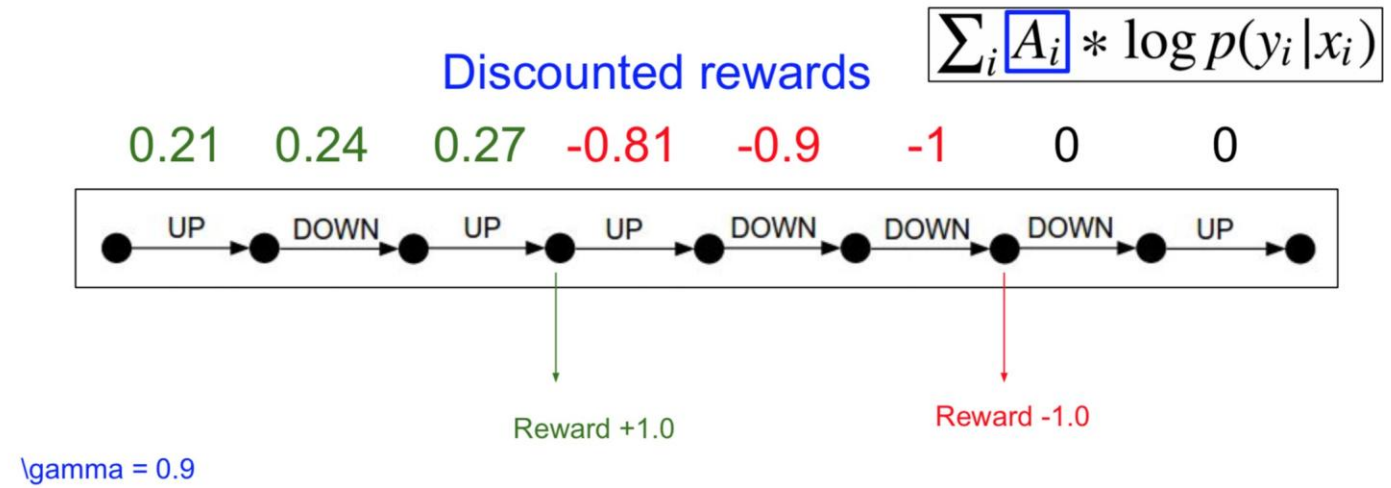


[Slides from Karpathy]

Pong from Pixels

Discounting

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

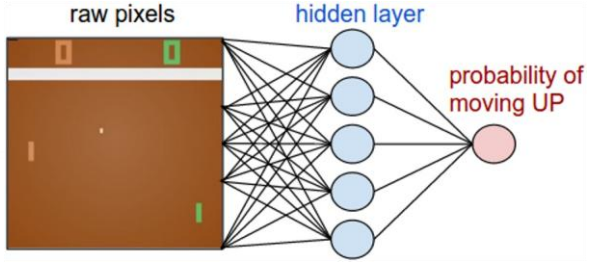


[Slides from Karpathy]

Pong from Pixels

$\pi(a \mid s)$

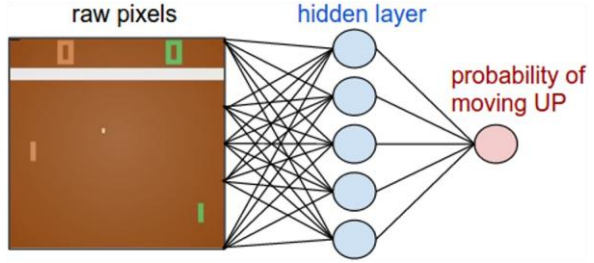
1. Initialize a policy network at random



[Slides from Karpathy]

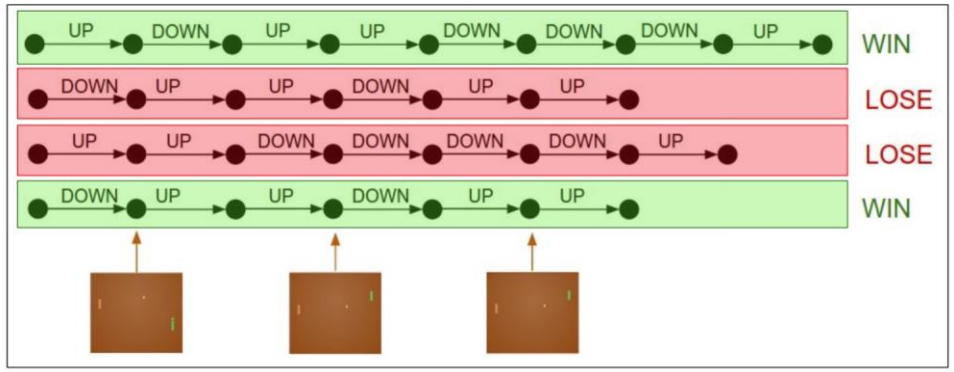
Pong from Pixels

$\pi(a \mid s)$



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

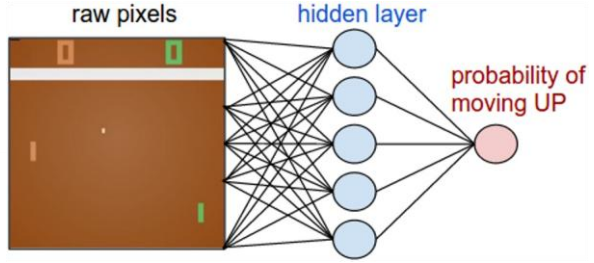
epsilon greedy!



[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

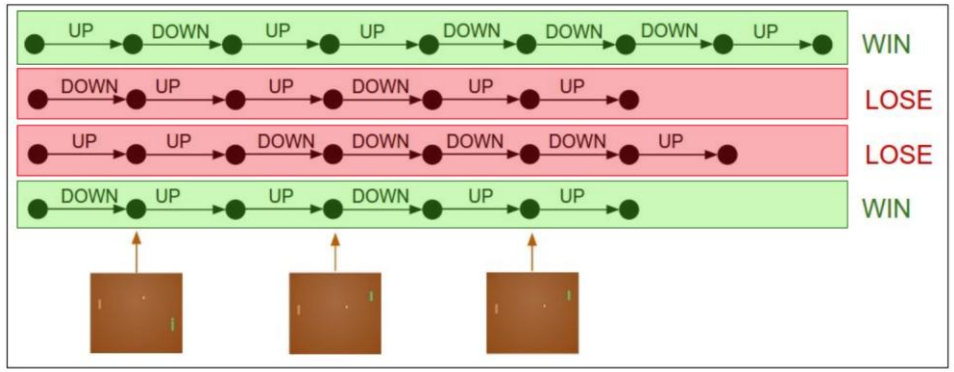
epsilon greedy!

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



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

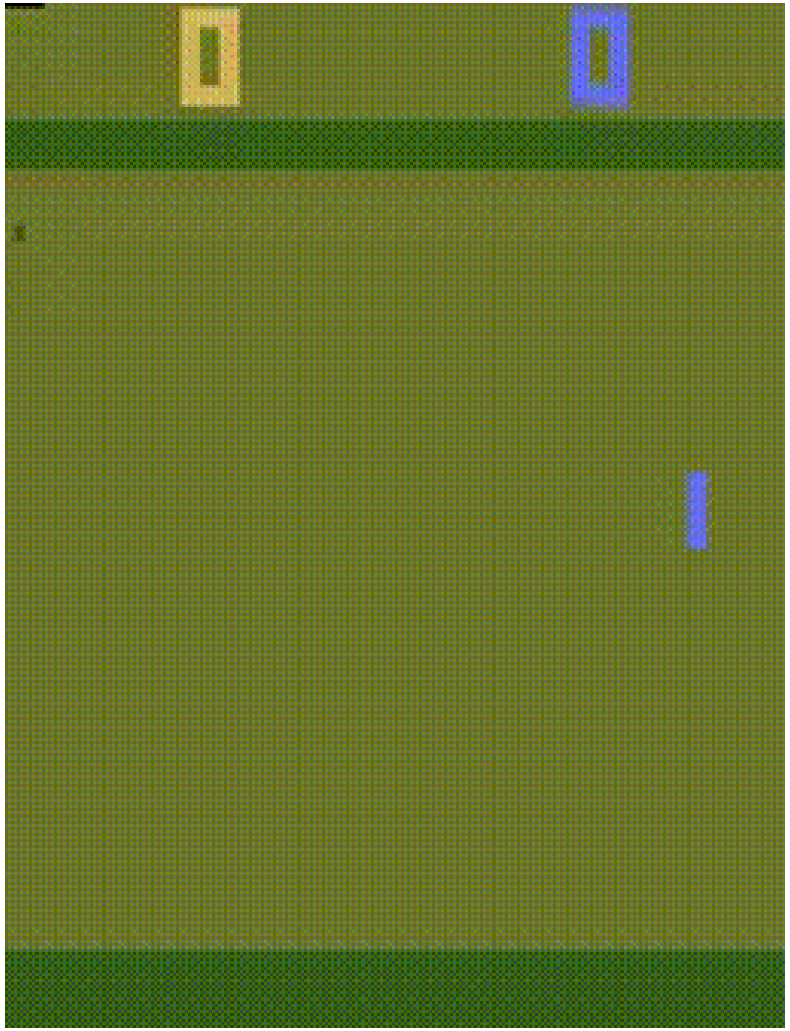
Does not require transition probabilities

Does not estimate Q(), V()

Predicts policy directly

[Slides from Karpathy]

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

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

[Slides from Karpathy]

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

Probability of a trajectory

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

Reward of a trajectory

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

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

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:

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

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:

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

REINFORCE Algorithm

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

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

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

Intractable

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

REINFORCE Algorithm

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

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

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

Intractable

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

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

REINFORCE Algorithm

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

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

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

Intractable

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

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

If we inject this back:

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

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

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

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

Doesn't depend on transition probabilities

REINFORCE Algorithm

Can we compute these without knowing the transition probabilities?

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

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

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

Doesn't depend on transition probabilities

Therefore when sampling a trajectory, we can estimate gradients:

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

Policy Gradients

Gradient estimator:

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

Interpretation:

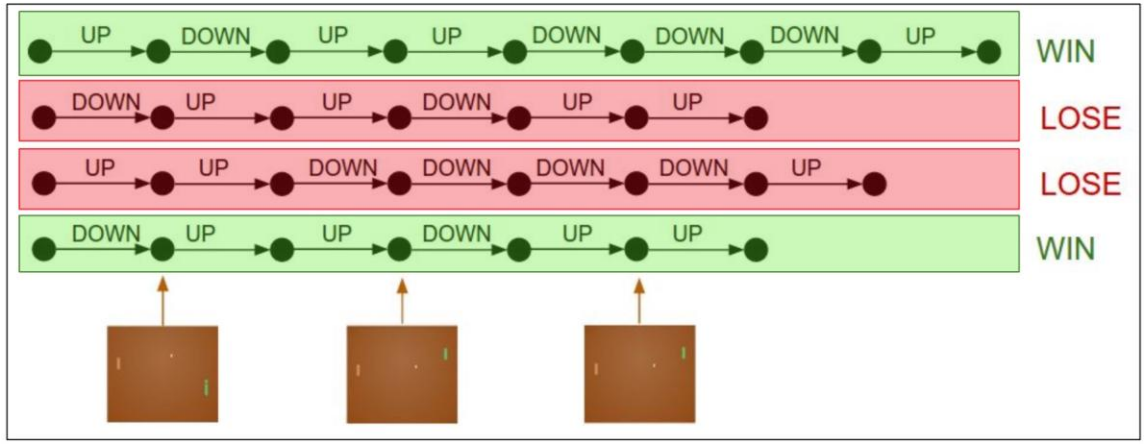
- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(trajectory)** is low, push down the probabilities of the actions seen

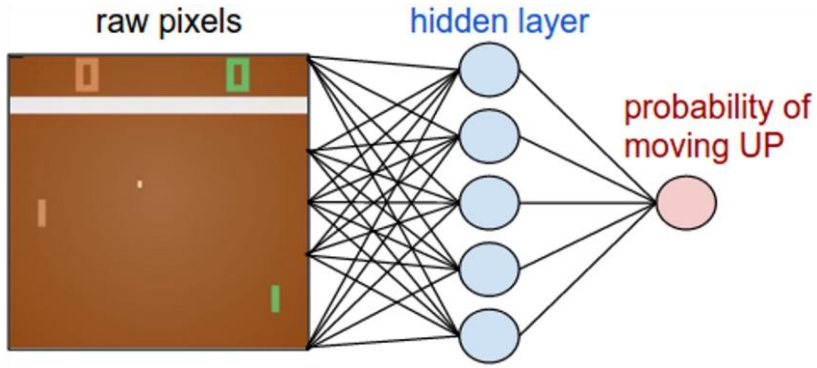
Pretend every action we took here was the correct label.

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

Pretend every action we took here was the wrong label.

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





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

Policy Gradients

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
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t | S_t, \boldsymbol{\theta})$

epsilon greedy

[Slides from Fragkiadaki, 10-703 CMU]

Policy Gradients

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?

[Slides from Fragkiadaki, 10-703 CMU]

Variance Reduction with a Baseline

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

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

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

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

e.g. exponential moving average of the rewards.

[Slides from Fragkiadaki, 10-703 CMU]

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 and state value functions!**

We are happy with an action **a** in a state **s** if **Q(s,a) - V(s)** is large.
Otherwise we are unhappy with an action if it's small.

Using this, we get the estimator:

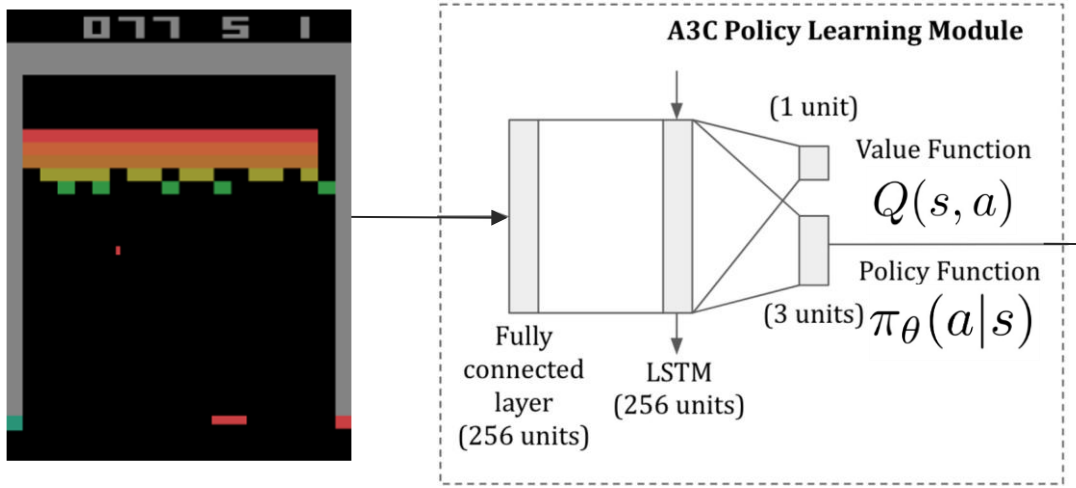
$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

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)

Exploration + experience replay
Decorrelate samples
Fixed targets



Critic: evaluates how good the action is

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

Q-learning target — Q-network

$\pi_\theta(a|s)$

Actor: decides what actions to take

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

Variance reduction with a baseline

[Minh et al., Asynchronous Methods for Deep Reinforcement Learning. ICML 2016]

Summary: RL Methods

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

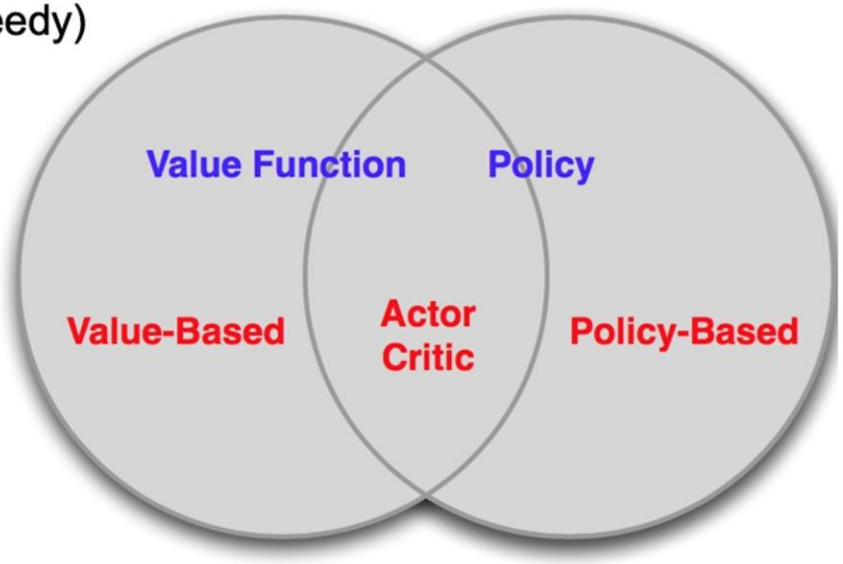
Value iteration
Policy iteration
(Deep) Q-learning

- Policy Based
  - No Value Function
  - Learned Policy

Policy gradients

- Actor-Critic
  - Learned Value Function
  - Learned Policy

Actor (policy)
Critic (Q-values)



[Slides from Fragkiadaki, 10-703 CMU]

Summary: RL Methods

Epsilon greedy + exploration
Experience replay
Decorrelate samples
Fixed targets

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

Value iteration
Policy iteration
(Deep) Q-learning

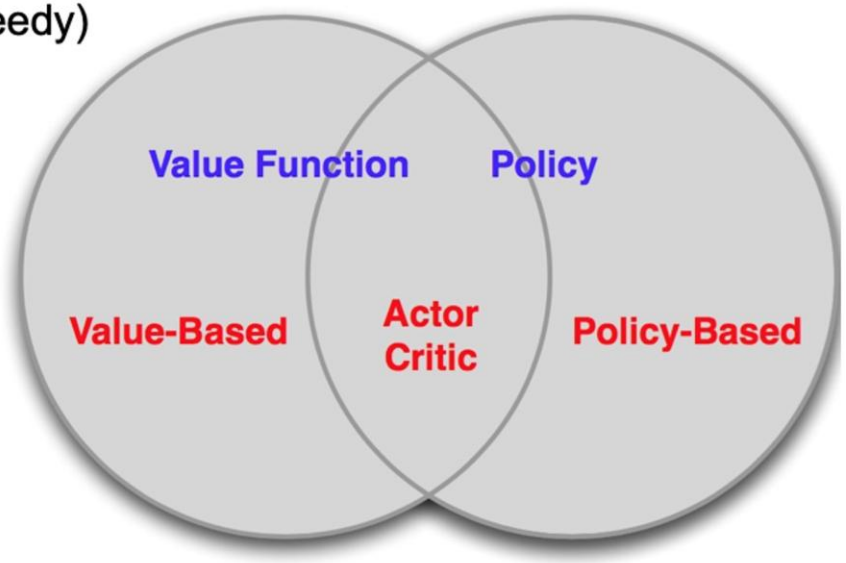
- Policy Based
  - No Value Function
  - Learned Policy

Policy gradients

Variance reduction with a baseline

- Actor-Critic
  - Learned Value Function
  - Learned Policy

Actor (policy)
Critic (Q-values)

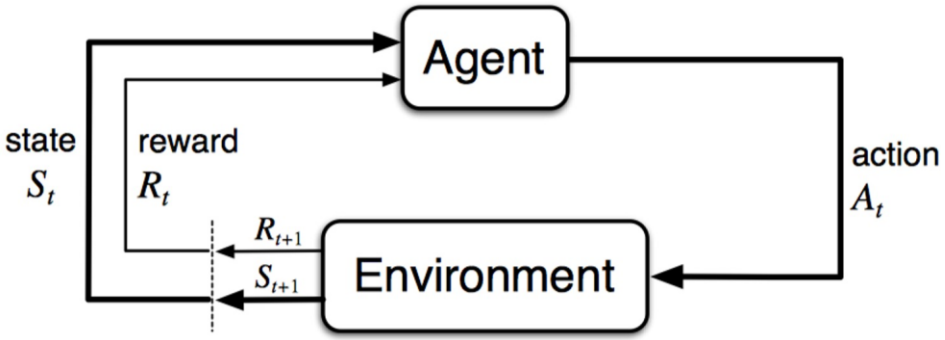


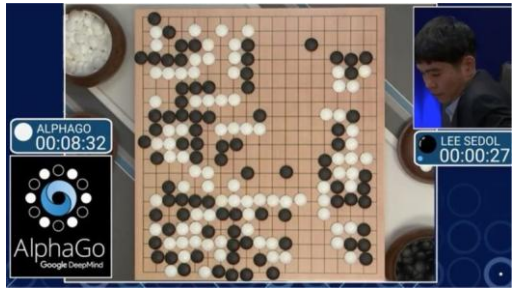
[Slides from Fragkiadaki, 10-703 CMU]

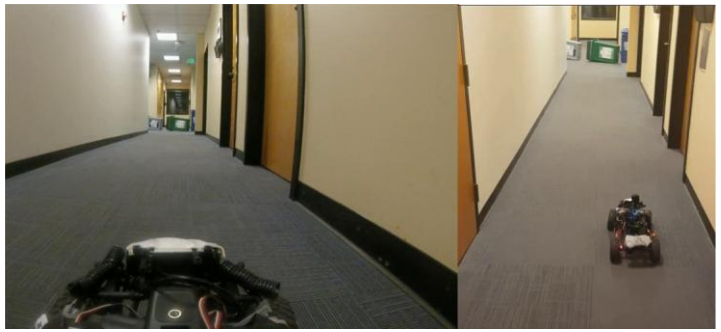
Exercises

1) What is the primary objective in Reinforcement Learning (RL)?

A. Minimize prediction error on labeled data
B. Maximize the immediate reward at each step
C. Maximize the expected cumulative return over time
D. Estimate the transition probabilities exactly







Exercises

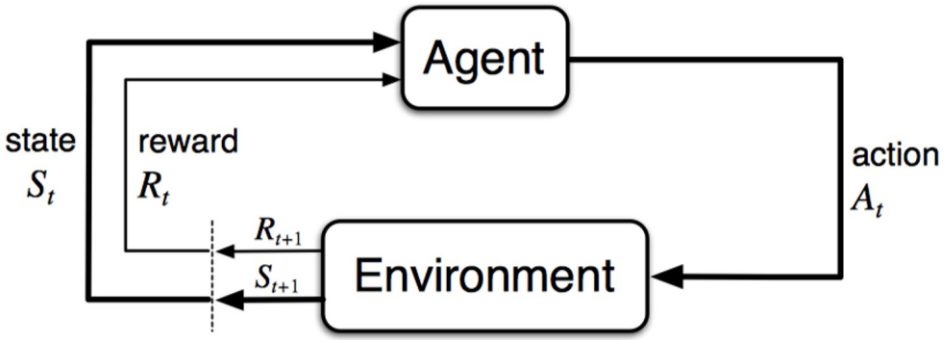
2) Which of the following is not a component of a Markov Decision Process (MDP)?

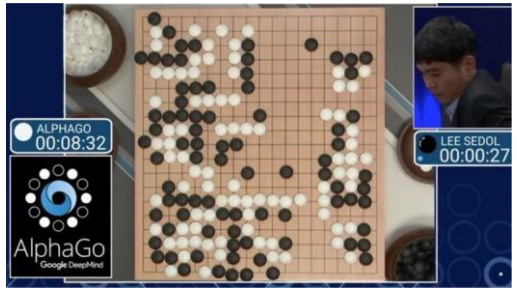
A. Set of states S

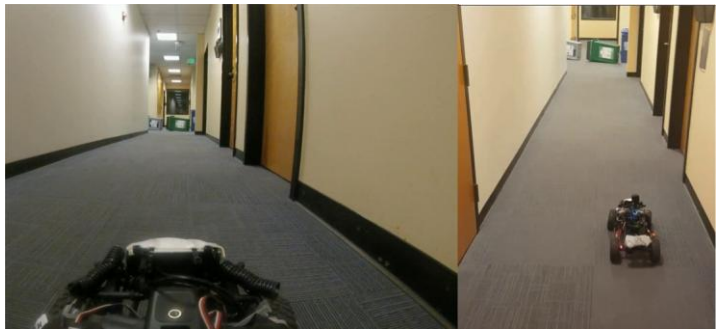
B. Reward function $r(s, a, s')$

C. Discount factor γ

D. Learning rate α







Exercises

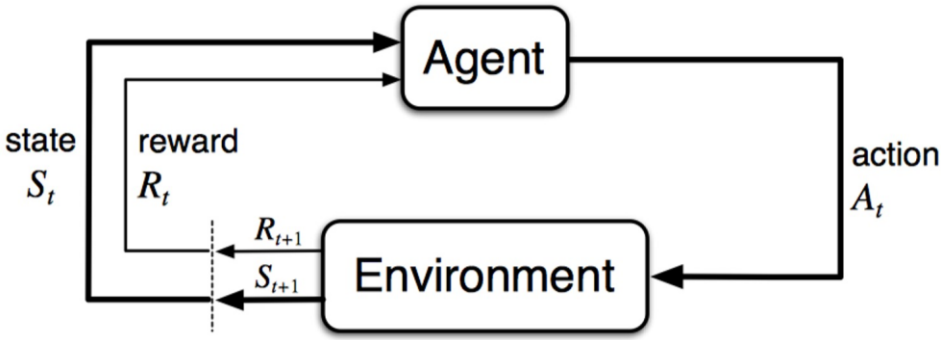
3) Compared with supervised learning, RL typically involves:

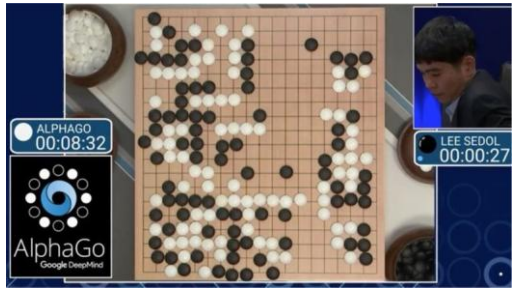
A. One-step decisions with dense labels

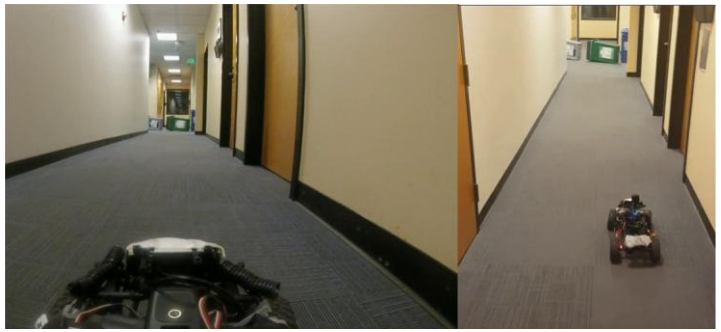
B. A fully known environment model

C. Sequential decisions with possibly sparse rewards and an unknown environment

D. Minimizing cross-entropy loss on actions







Exercises

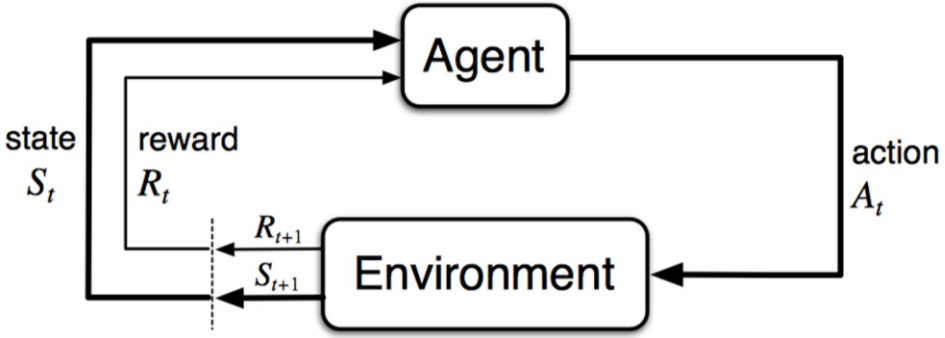
4) A key drawback of pure imitation learning highlighted in the slides is:

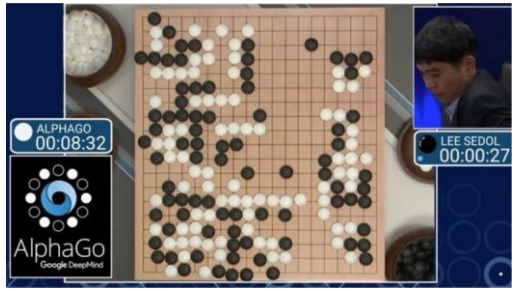
A. It eliminates distribution shift

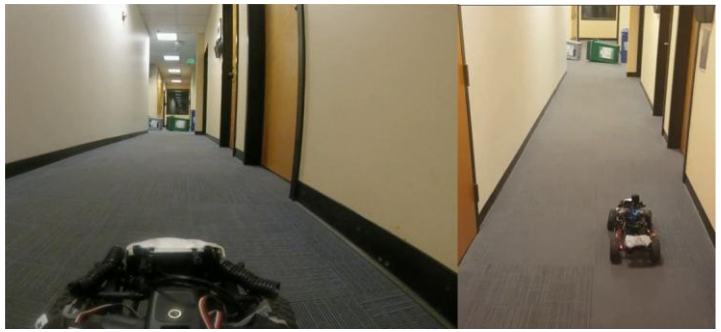
B. It never needs expert trajectories

C. It suffers from train–test distribution mismatch and may fail to recover from sub-optimal states

D. It always improves safety







Exercises

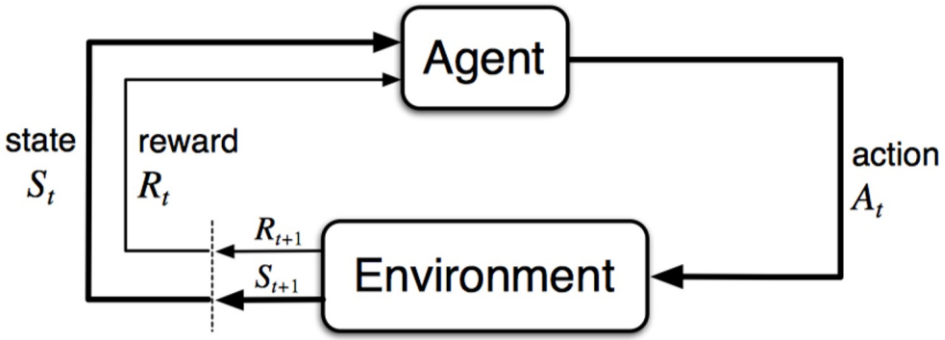
5) The state-value function $V^{\pi}(s)$ is best described as:

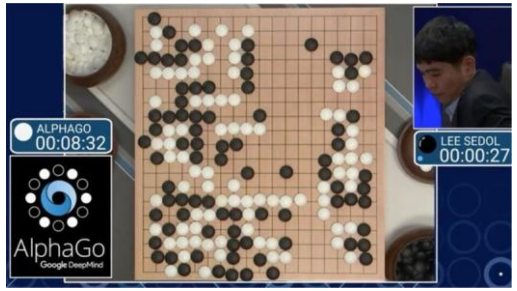
A. Immediate reward at state s

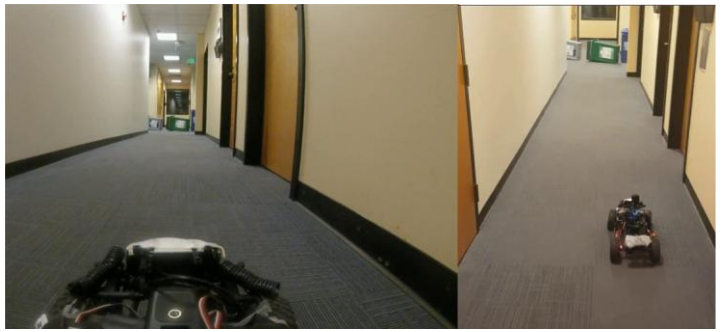
B. Expected total reward from s when following policy π

C. Best possible one-step reward from s

D. Expected reward of taking action a in s







Exercises

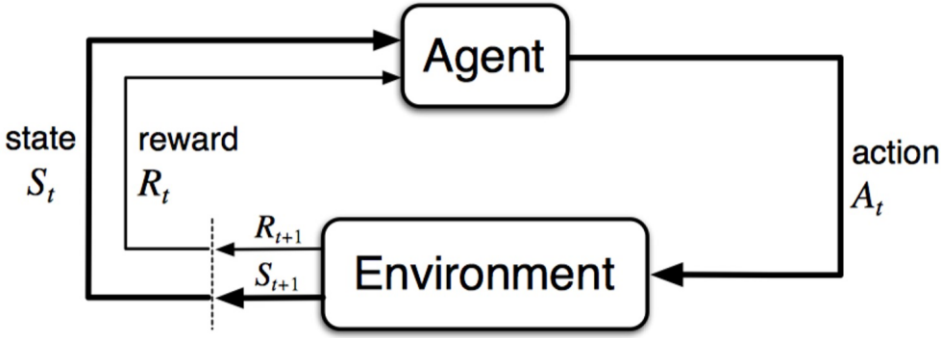
6) The action-value function $Q^{\pi}(s, a)$ gives:

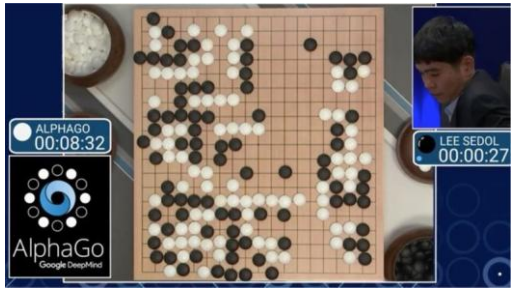
A. Immediate reward for action a in s

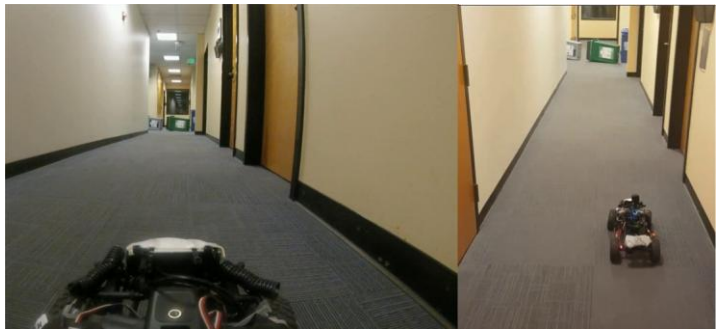
B. Expected return after taking a in s and then following π

C. Max return over all policies from s

D. Return assuming random actions







Exercises

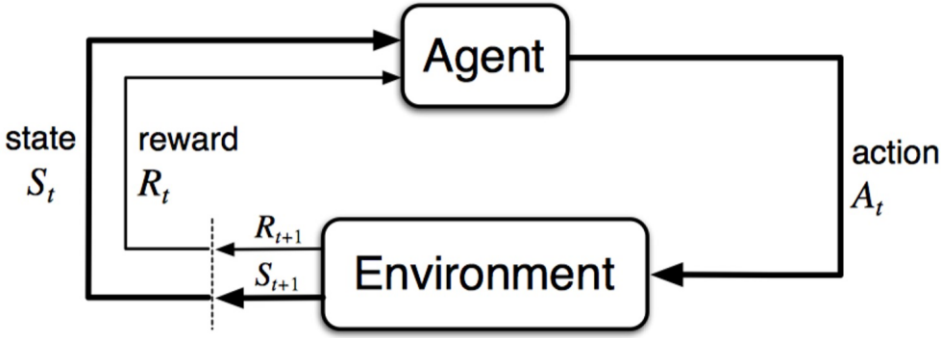
7) An optimal policy can be obtained by:

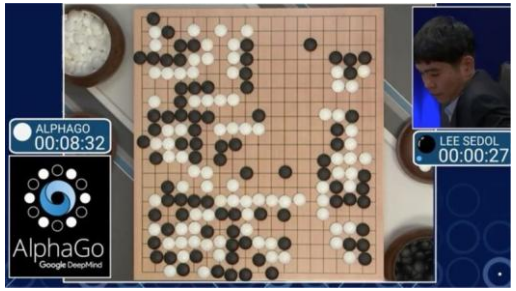
A. Minimizing $V^*(s)$

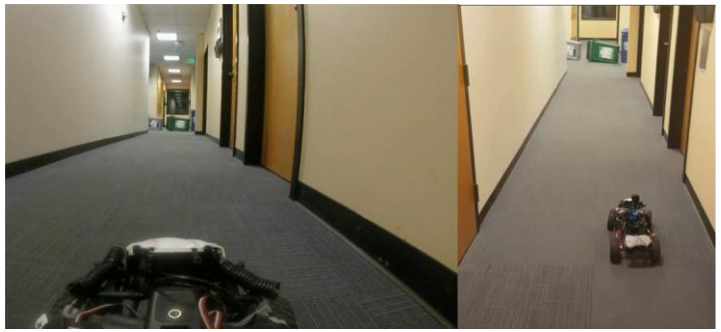
B. Randomly exploring until convergence

C. Choosing actions that maximize $Q^*(s, a)$ (or via one-step lookahead using V^*)

D. Greedy choice w.r.t. immediate reward







Exercises

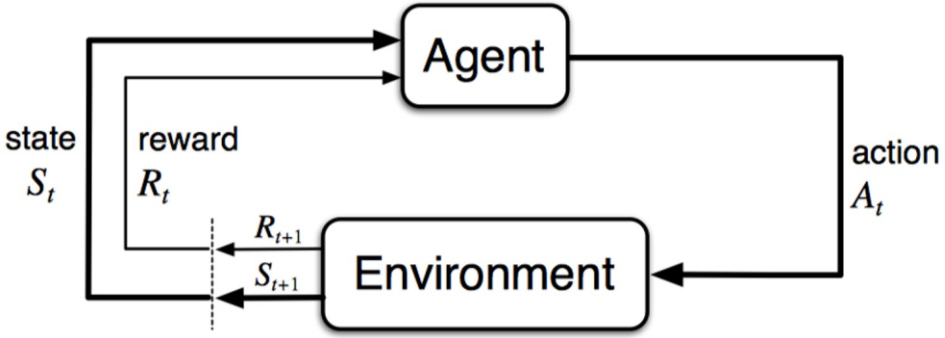
8) Which statement about value-based vs policy-based methods is correct?

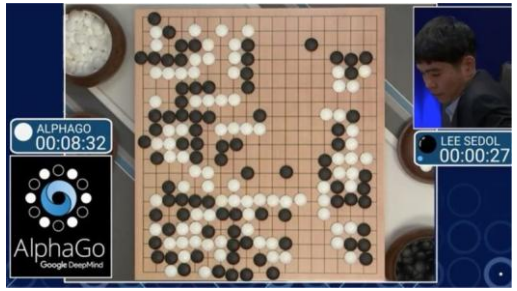
A. Value-based methods can struggle with continuous $\arg\max$

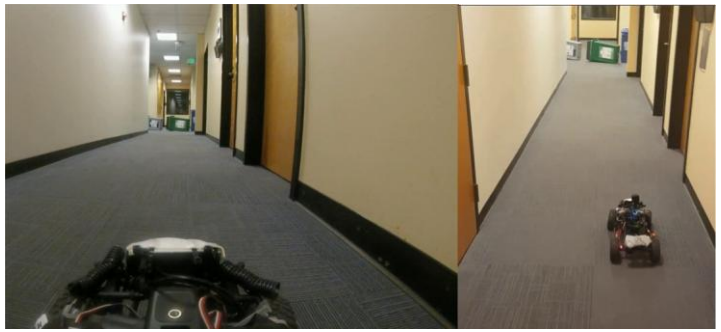
B. Policy-based methods cannot handle continuous actions

C. Value-based methods are always simpler

D. Policy-based methods always use fewer samples



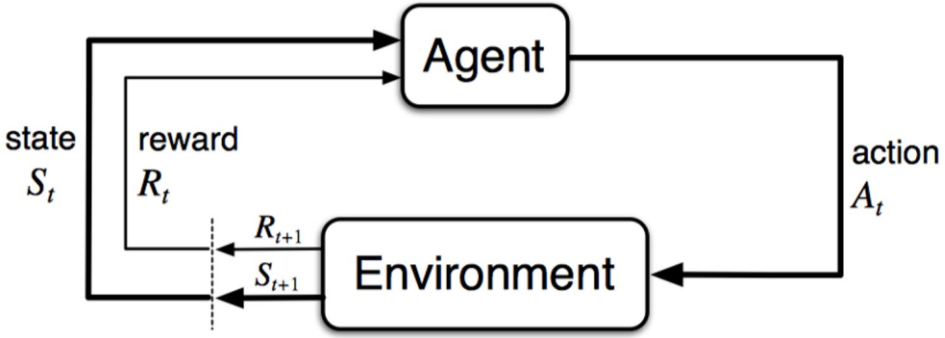


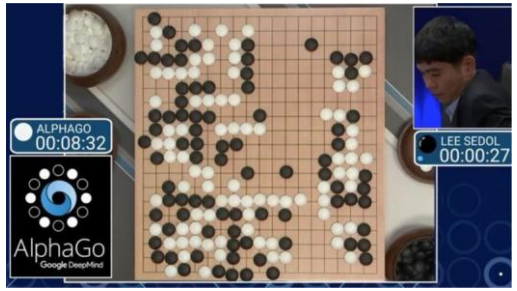


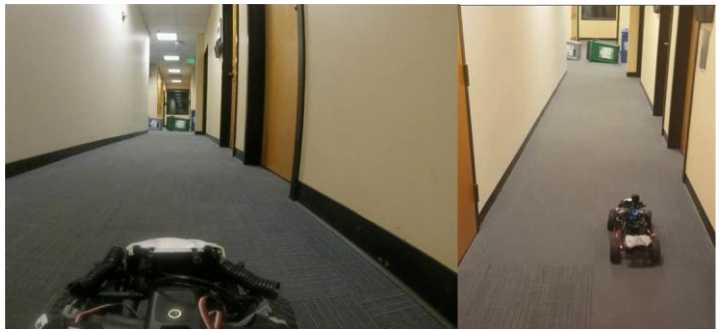
Exercises

9) A benefit of policy-based methods mentioned in the slides is:

A. Always higher sample efficiency than value-based
B. Directly learn the policy, which can be more interpretable
C. Guaranteed convergence without variance
D. No need for exploration







Exercises

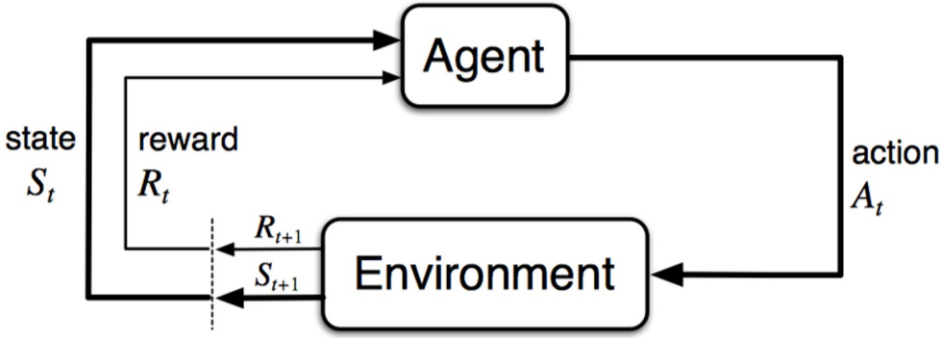
10) Bellman optimality equations are typically solved using:

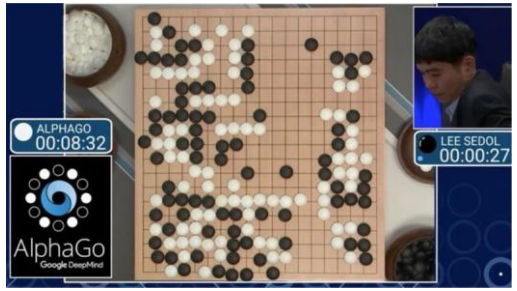
A. Maximum likelihood estimation

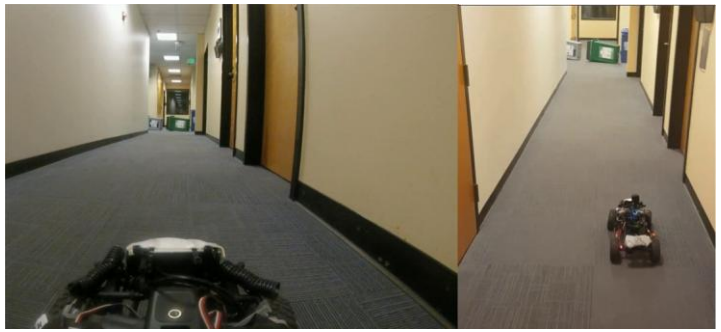
B. Iterative dynamic programming methods (e.g., value iteration, policy iteration)

C. Linear regression

D. Kernel density estimation







Exercises

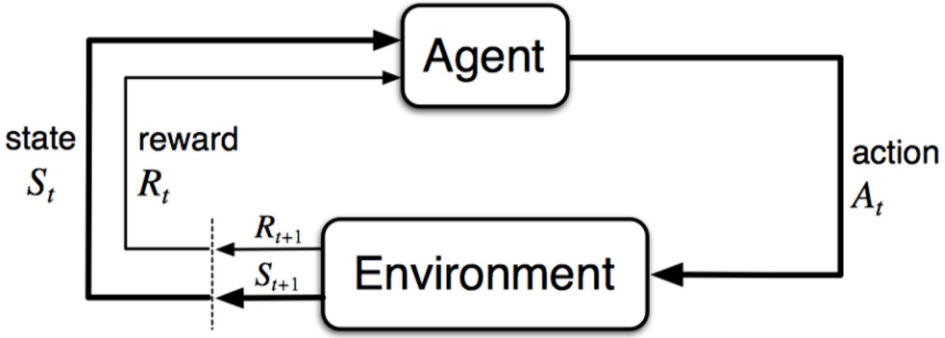
11) Value Iteration uses one-step lookahead to:

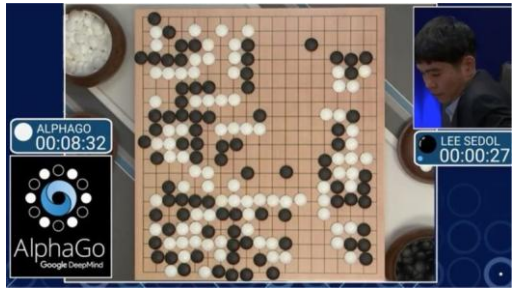
A. Compute a supervised loss

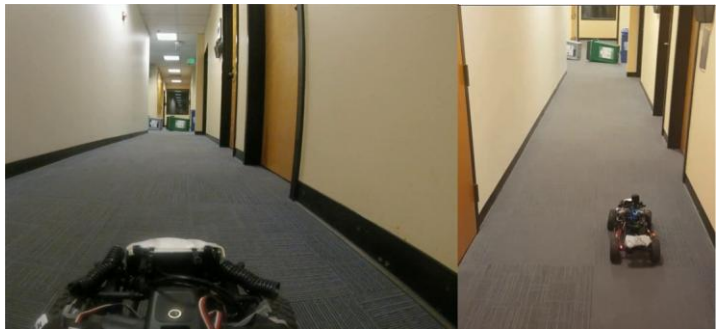
B. Back up estimates of V and pick the best action by maximizing the backed-up value

C. Estimate the learning rate

D. Remove correlation between samples



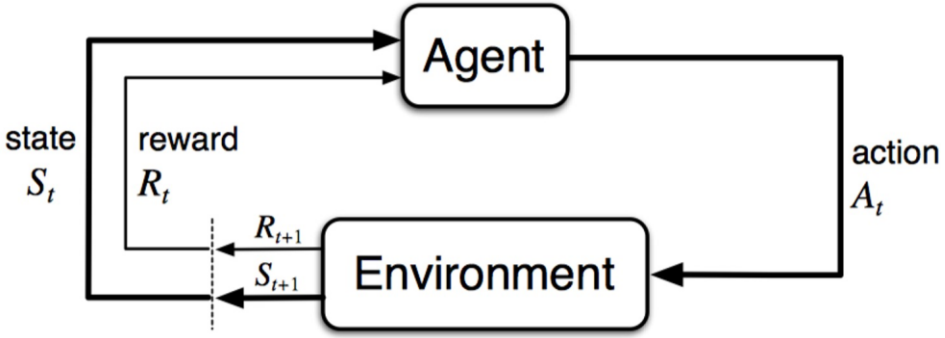


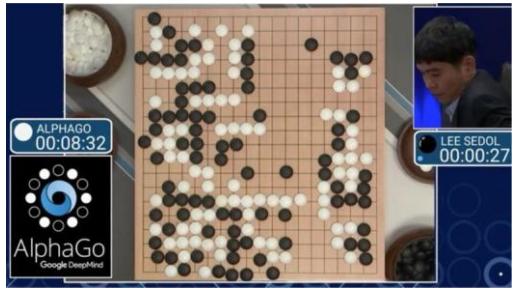


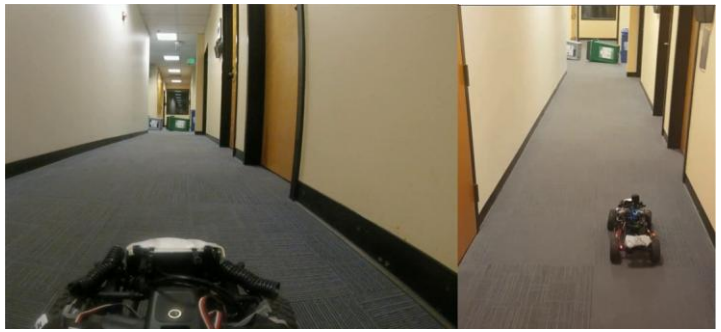
Exercises

12) A limitation of exact dynamic programming (value/policy iteration) emphasized in the slides is:

A. Inability to handle any discrete states
B. Requires known transitions and iterating/storing all states/actions (small, discrete spaces)
C. Requires on-policy rollouts only
D. Cannot compute an optimal policy







Exercises

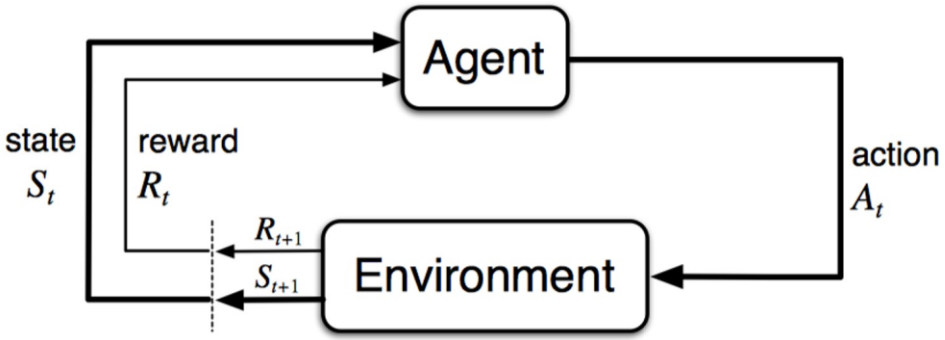
13) Why introduce Q-learning for unknown MDPs?

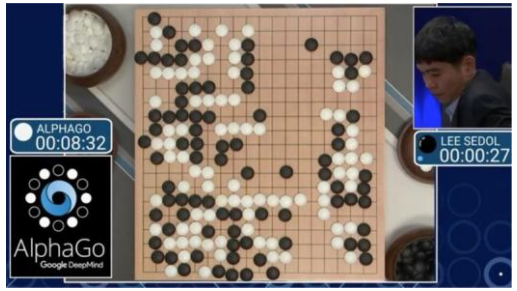
A. To avoid rewards

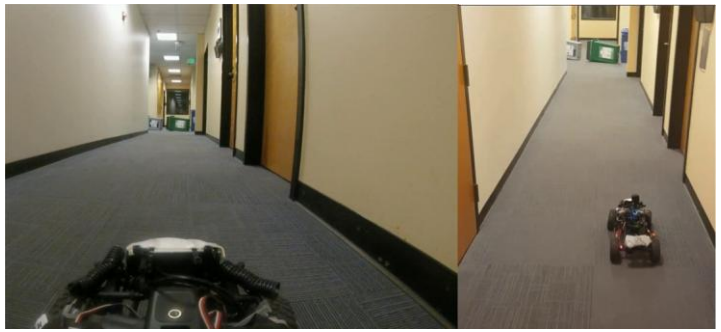
B. To learn optimal action values from interaction without knowing the transition model

C. To make targets stationary automatically

D. To eliminate exploration







Exercises

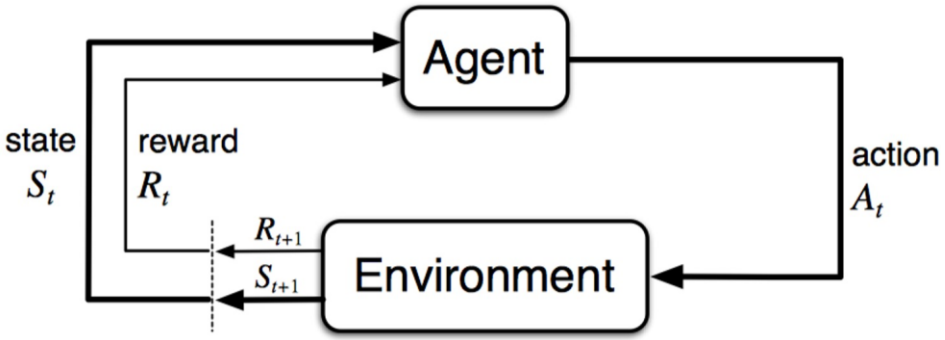
14) The standard tabular Q-learning update is:

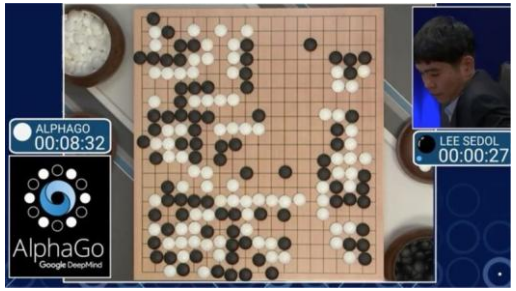
A. $Q \leftarrow Q + \alpha[r - Q]$

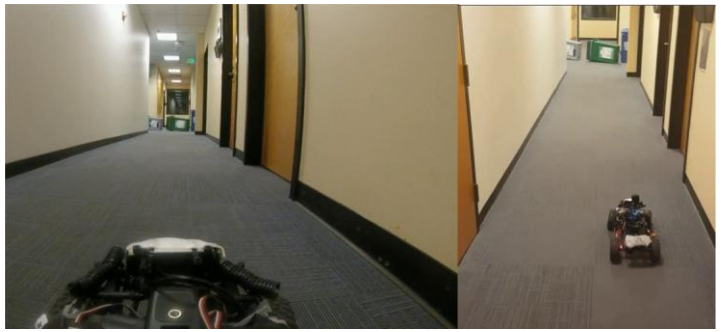
B. $Q \leftarrow r + \gamma \min_{a'} Q(s', a')$

C. $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right]$

D. $Q(s, a) \leftarrow Q(s, a) - \alpha \nabla Q$







Exercises

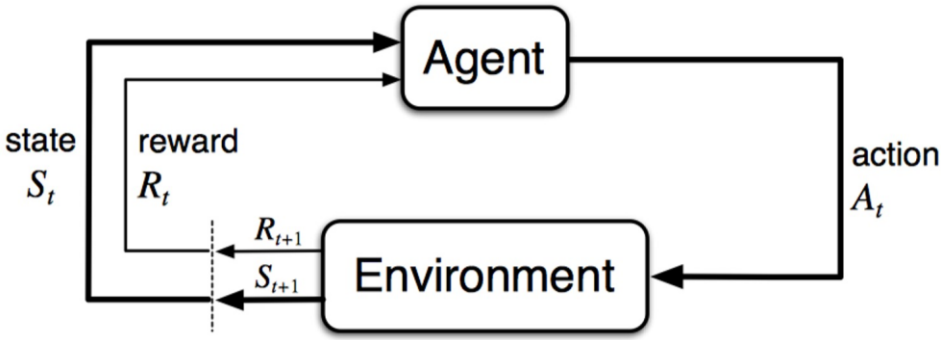
15) In ϵ-greedy exploration, the slides recommend:

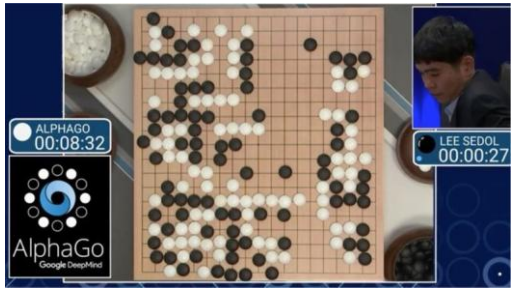
A. Keep ϵfixed at 1

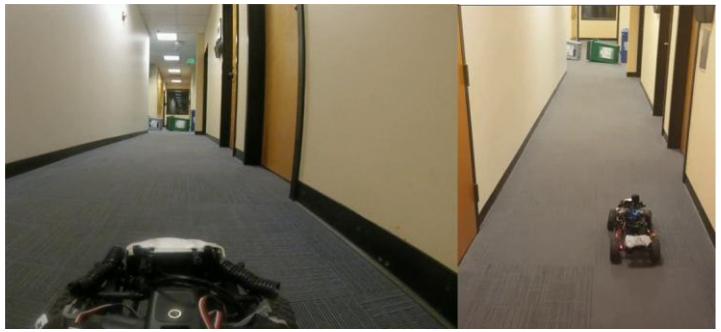
B. Gradually decrease ϵas learning progresses

C. Set $\epsilon = 0$from the start

D. Use ϵonly for terminal states







Exercises

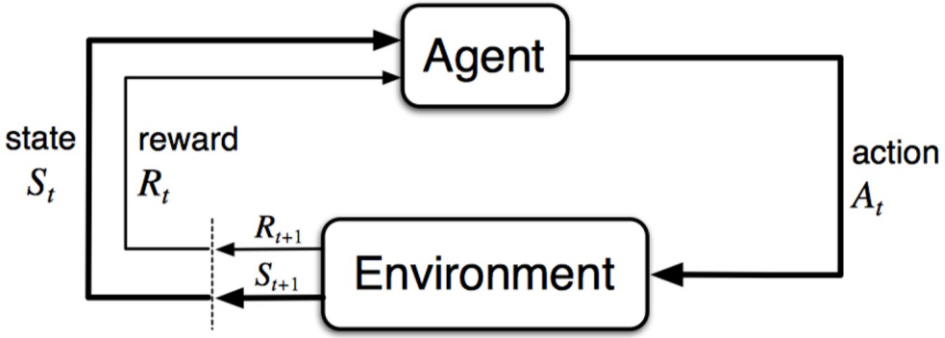
16) A limitation of tabular Q-learning is:

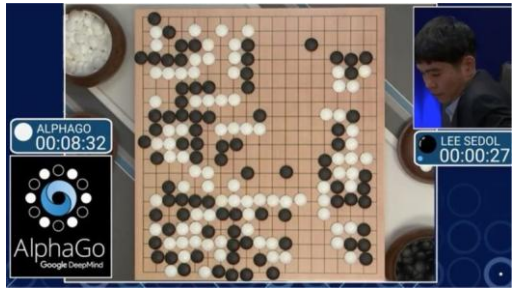
A. It cannot learn from rewards

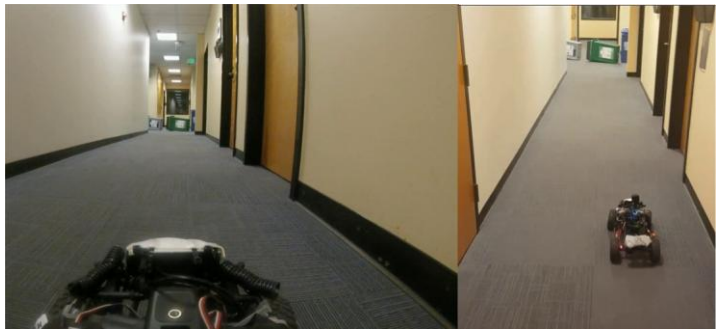
B. It still requires small, discrete $S$and A; generalization to unseen states is poor

C. It requires known transitions

D. It cannot converge







Exercises

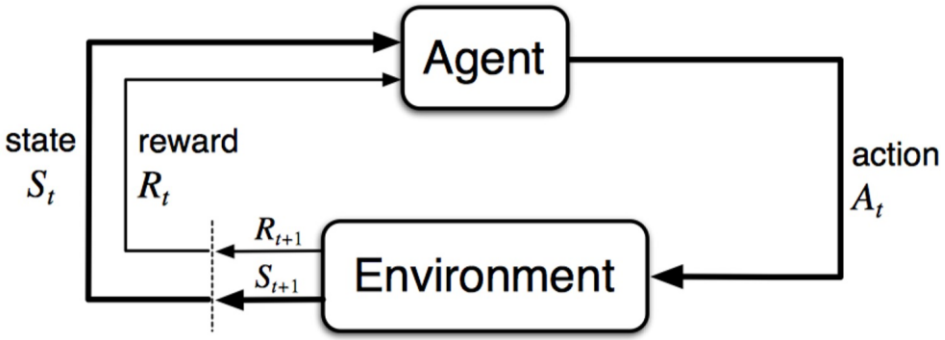
17) Deep Q-Learning (DQN) replaces the table with:

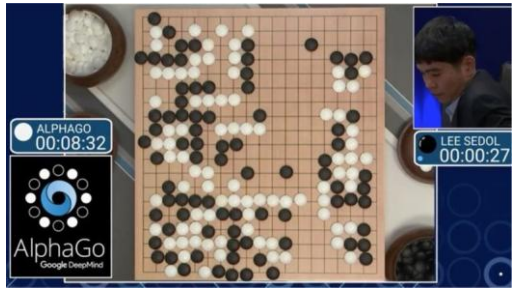
A. A reward model

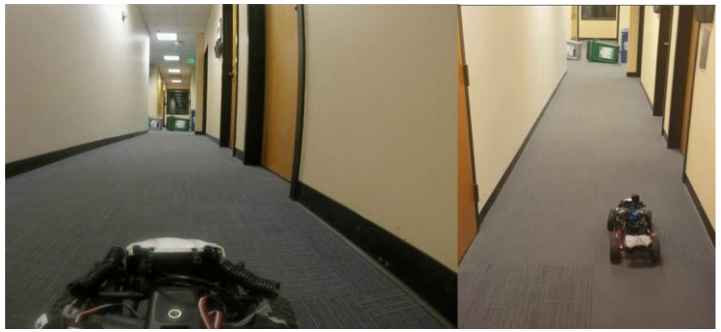
B. A policy gradient estimator

C. A neural network $Q_w(s, a)$ for high-dimensional/continuous states and generalization

D. A handcrafted feature dictionary



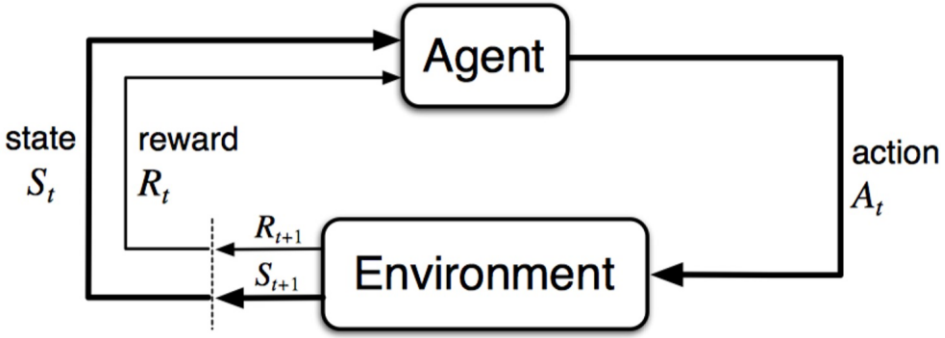


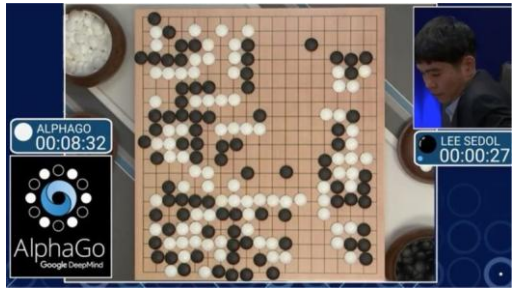


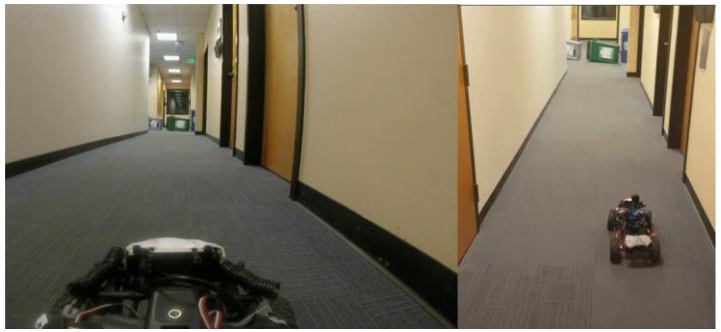
Exercises

18) Two major stability challenges for DQN listed in the slides are:

A. Overfitting and vanishing labels
B. Sparse gradients and exploding logits
C. Correlated samples and non-stationary targets
D. Lack of GPU memory and slow I/O







Exercises

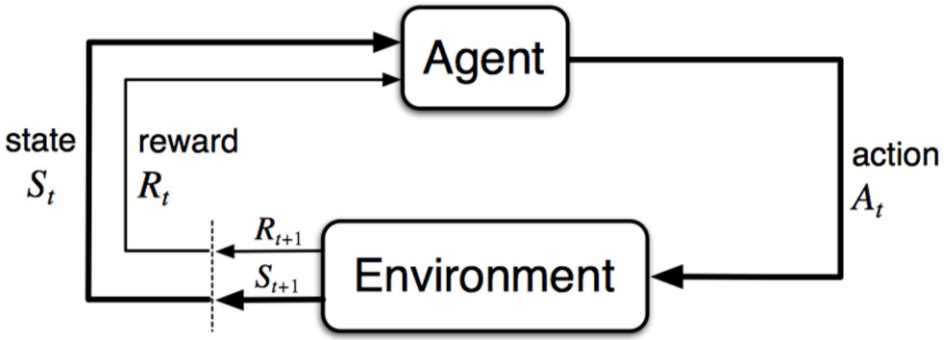
19) Experience replay primarily helps by:

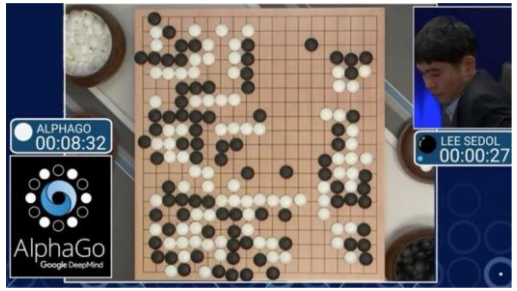
A. Making targets unbiased

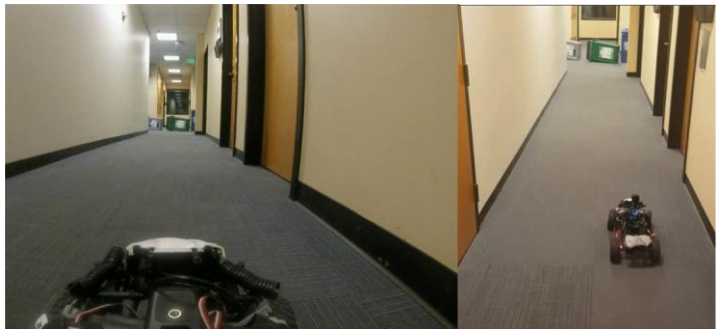
B. Decorrelating samples by replaying randomly drawn past transitions

C. Forcing on-policy learning

D. Increasing the discount factor







Exercises

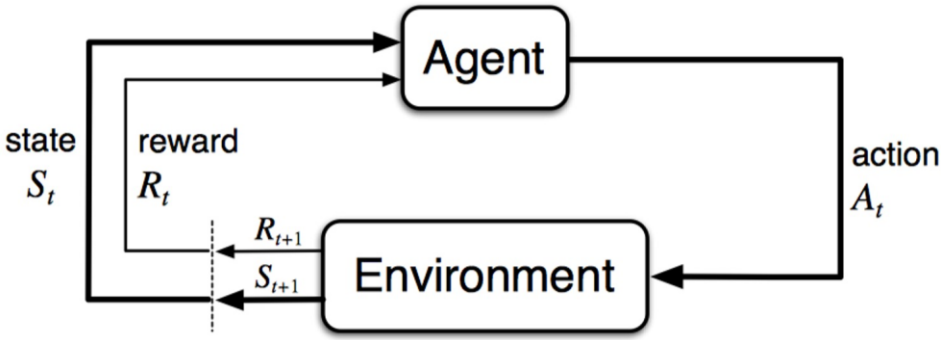
20) “Fixed Q-targets” in DQN refers to:

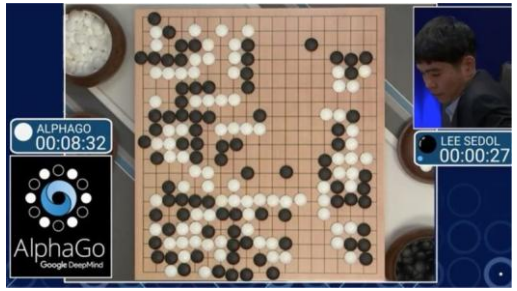
A. Fixing $\gamma$to 1

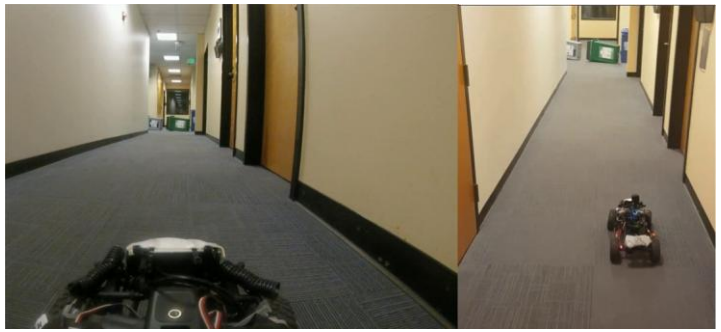
B. Using a lagged/target network to compute TD targets and updating it periodically

C. Freezing the replay buffer

D. Fixing the learning rate







Exercises

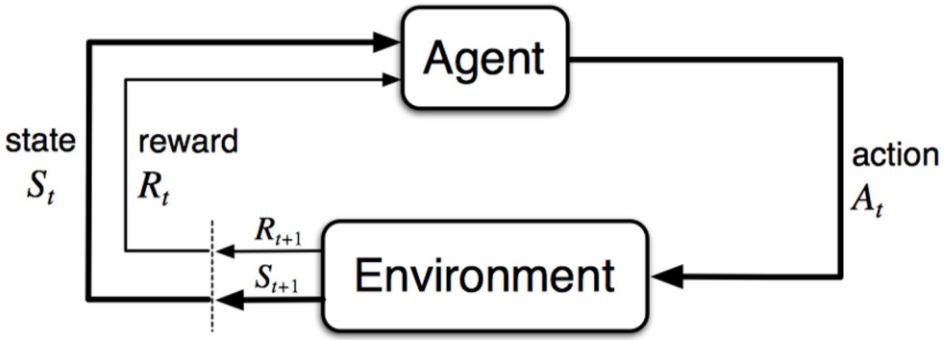
21) The policy gradient objective ("expected returns") can be optimized via:

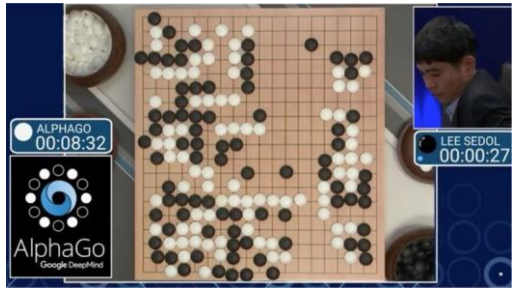
A. Value iteration

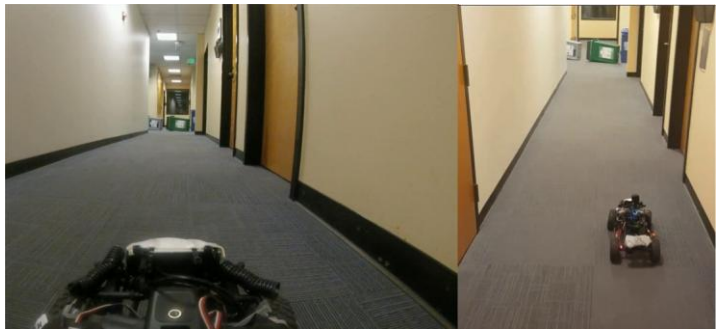
B. Gradient ascent using $\mathbb{E}[\nabla_\theta \log \pi_\theta (a \mid s)\, G]$

C. Principal component analysis

D. Policy evaluation with known transitions only



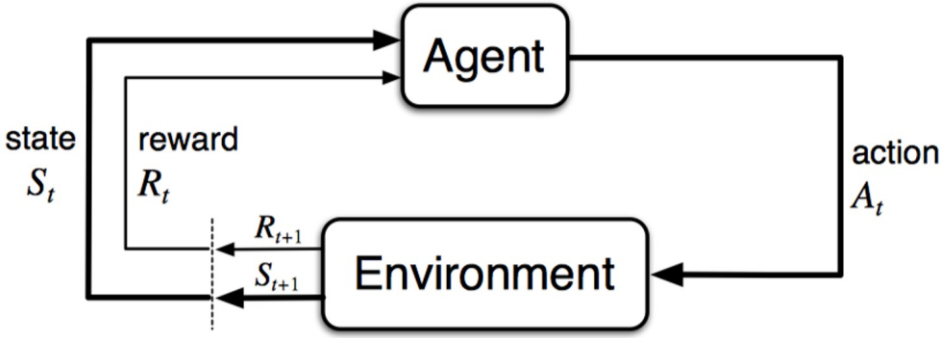


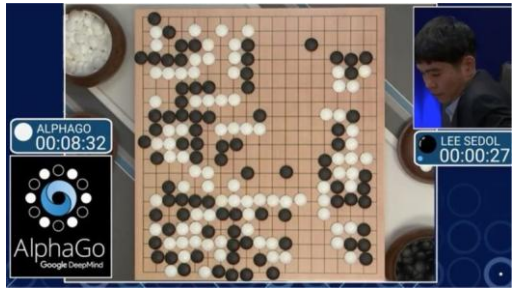


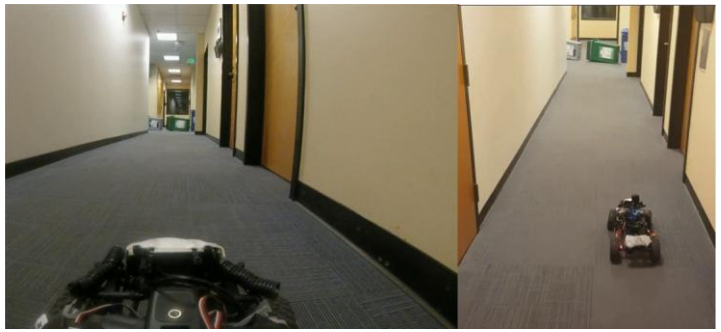
Exercises

22) A key property of the REINFORCE estimator shown is that it:

A. Requires exact transition probabilities
B. Requires a learned model of dynamics
C. Does not depend on knowing the transition probabilities
D. Uses bootstrapped targets







Exercises

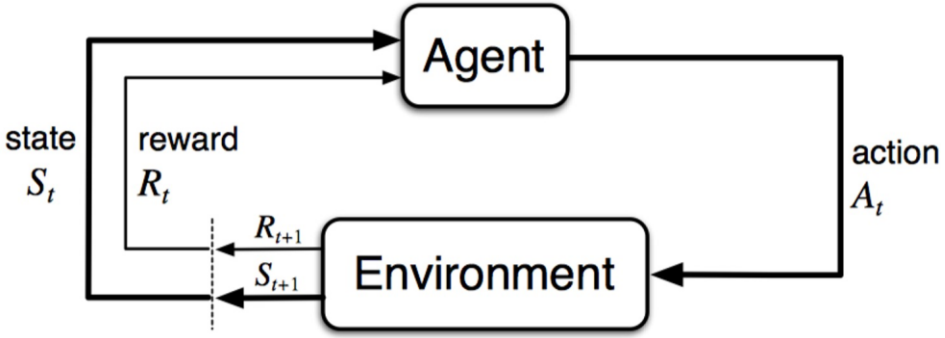
23) To reduce variance in policy gradients, the slides propose:

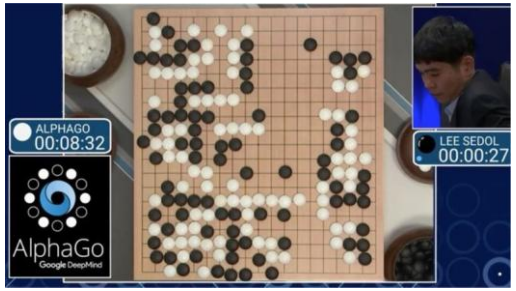
A. Increasing ϵforever

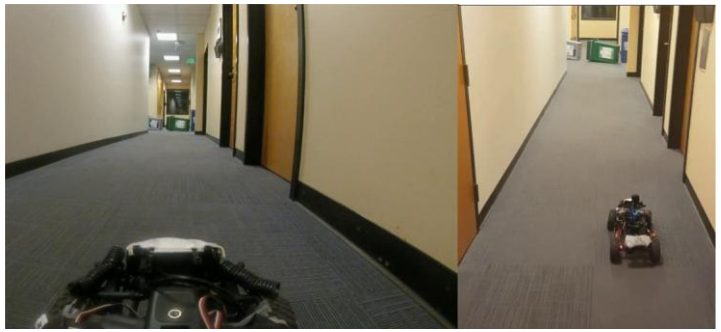
B. Subtracting a baseline $b(s)$)e.g., moving-average or value function)

C. Decreasing the reward scale to zero

D. Replacing returns with immediate rewards



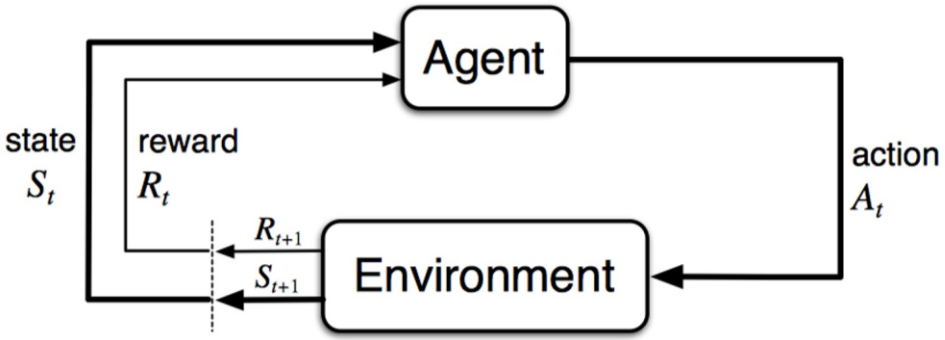


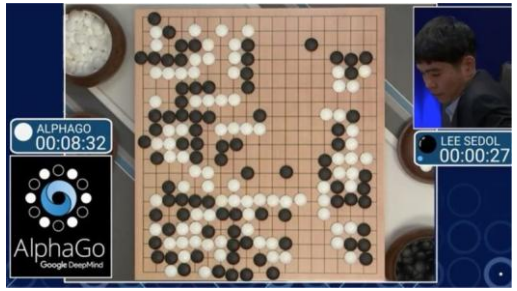


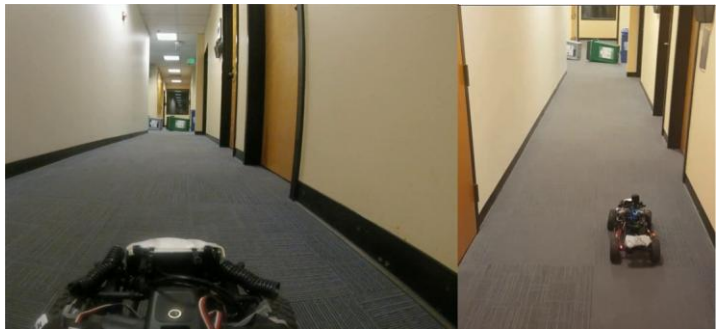
Exercises

24) A valid baseline in policy gradients may depend on:

A. Action only
B. Reward only
C. State (but not on the action), to avoid bias
D. The next state only







Exercises

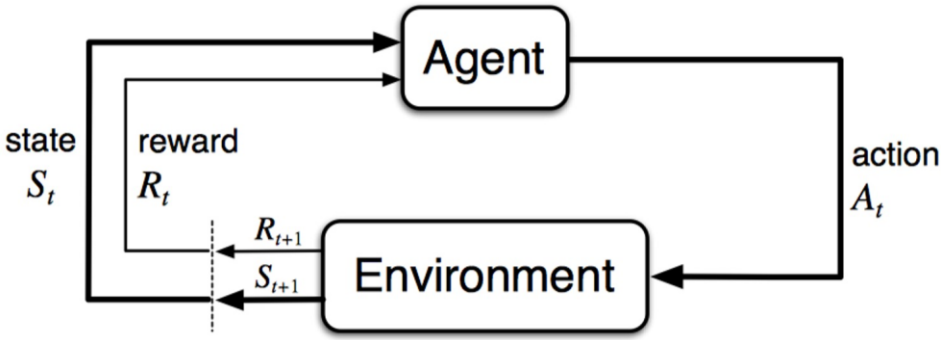
25) In actor–critic, the advantage used in the policy update can be written as:

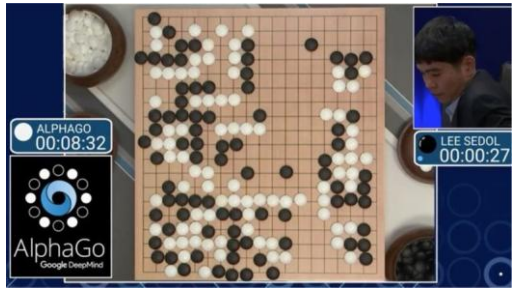
A. $A(s, a) = Q(s, a) + V(s)$

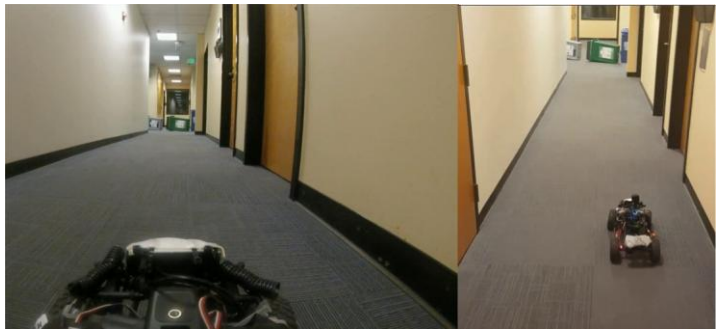
B. $A(s, a) = Q(s, a) - V(s)$,often approximated by $r + \gamma V(s') - V(s)$

C. $A(s, a) = r$

D. $A(s, a) = \gamma^{-1} V(s)$







Exercises

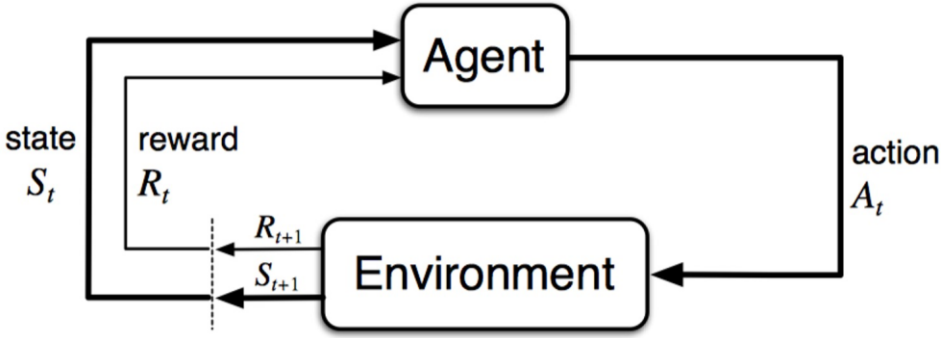
26) The “actor” and “critic” roles are, respectively:

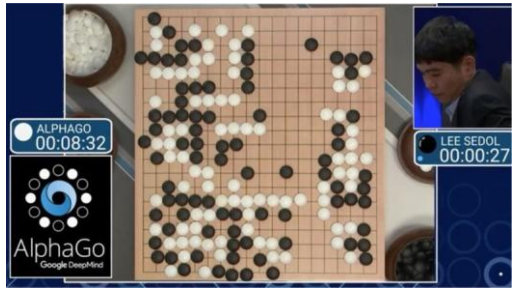
A. Evaluate actions; choose actions

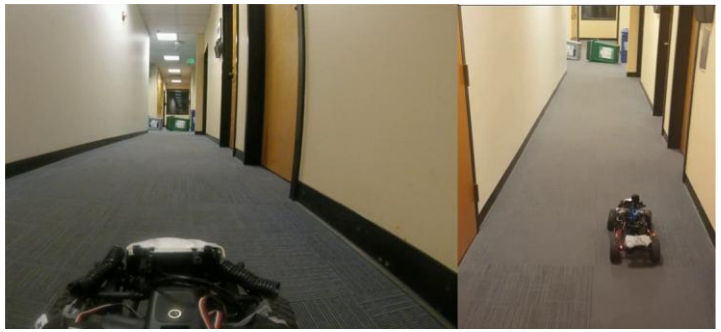
B. Choose actions (policy); evaluate them (value/Q) for learning

C. Update replay; update target network

D. Compute rewards; compute transitions







Exercises

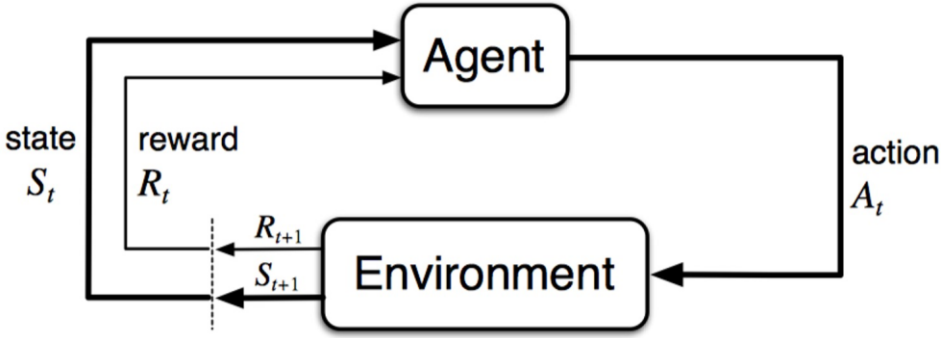
27) Which combination (from the summary slide) stabilizes deep value learning?

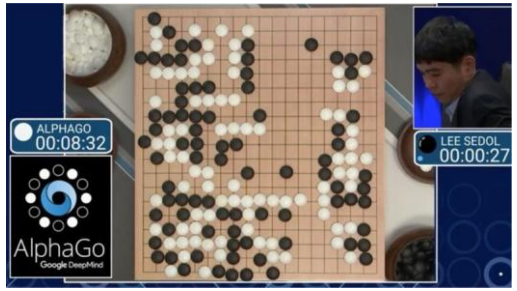
A. Entropy regularization + KL penalty

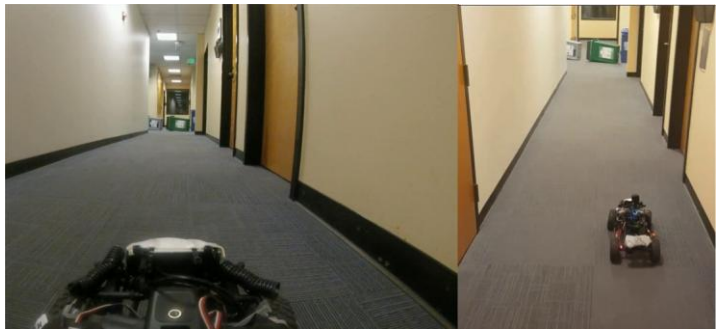
B. ϵ-greedy exploration + experience replay + fixed targets

C. Early stopping + cross-validation

D. Reward clipping + label smoothing only







Exercises

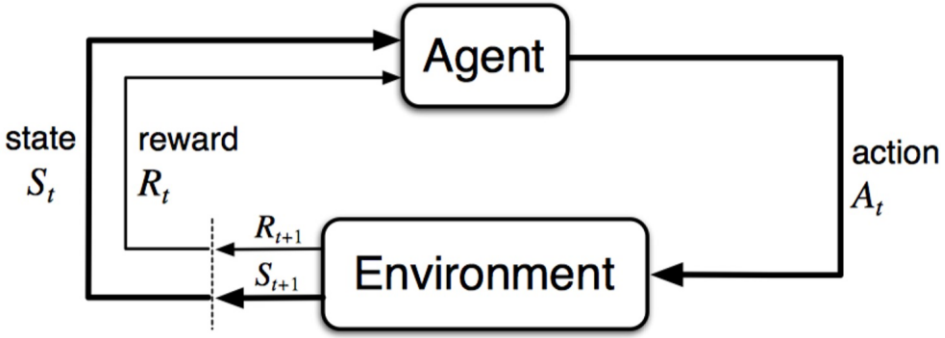
28) Which of the following best describes the limitation that motivates function approximation in RL?

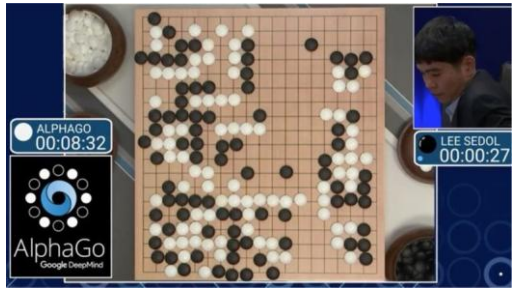
A. Rewards are always dense

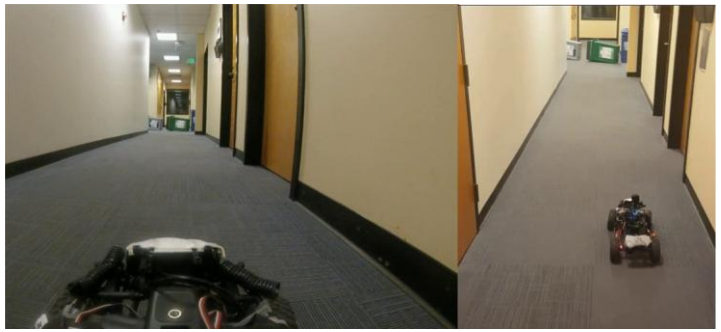
B. Tabular methods scale poorly because they must store/update $Q(s, a)$for every discrete pair

C. Policies cannot be stochastic

D. Value functions cannot be learned from







Exercises

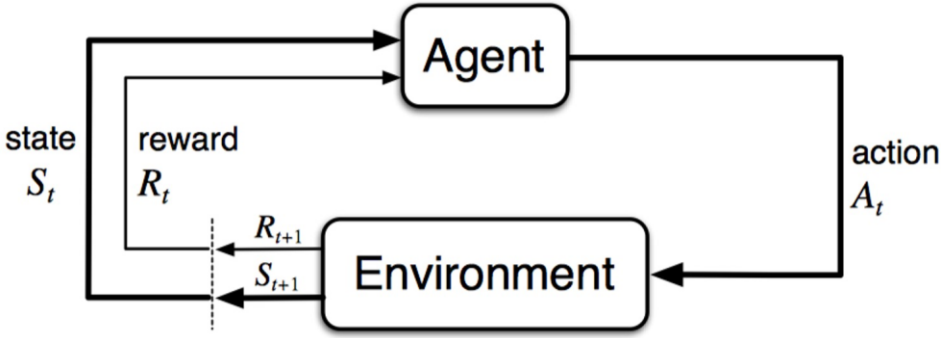
29) Policy iteration is:

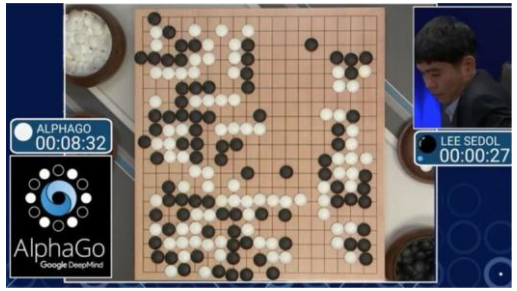
A. A bandit-only algorithm

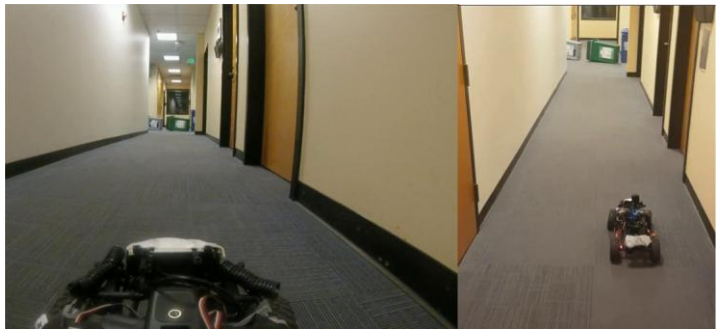
B. An exact method alternating policy evaluation and policy improvement, guaranteed to converge (under assumptions)

C. A model-free method

D. Equivalent to REINFORCE







Exercises

30) The “exploration vs exploitation” slide warns that:

A. Poor initial Q estimates don’t matter

B. Bad early estimates can trap the policy in sub-optimal regions without sufficient exploration

C. Exploration is unnecessary if rewards are sparse

D. Always set $\epsilon = 0$to converge faster