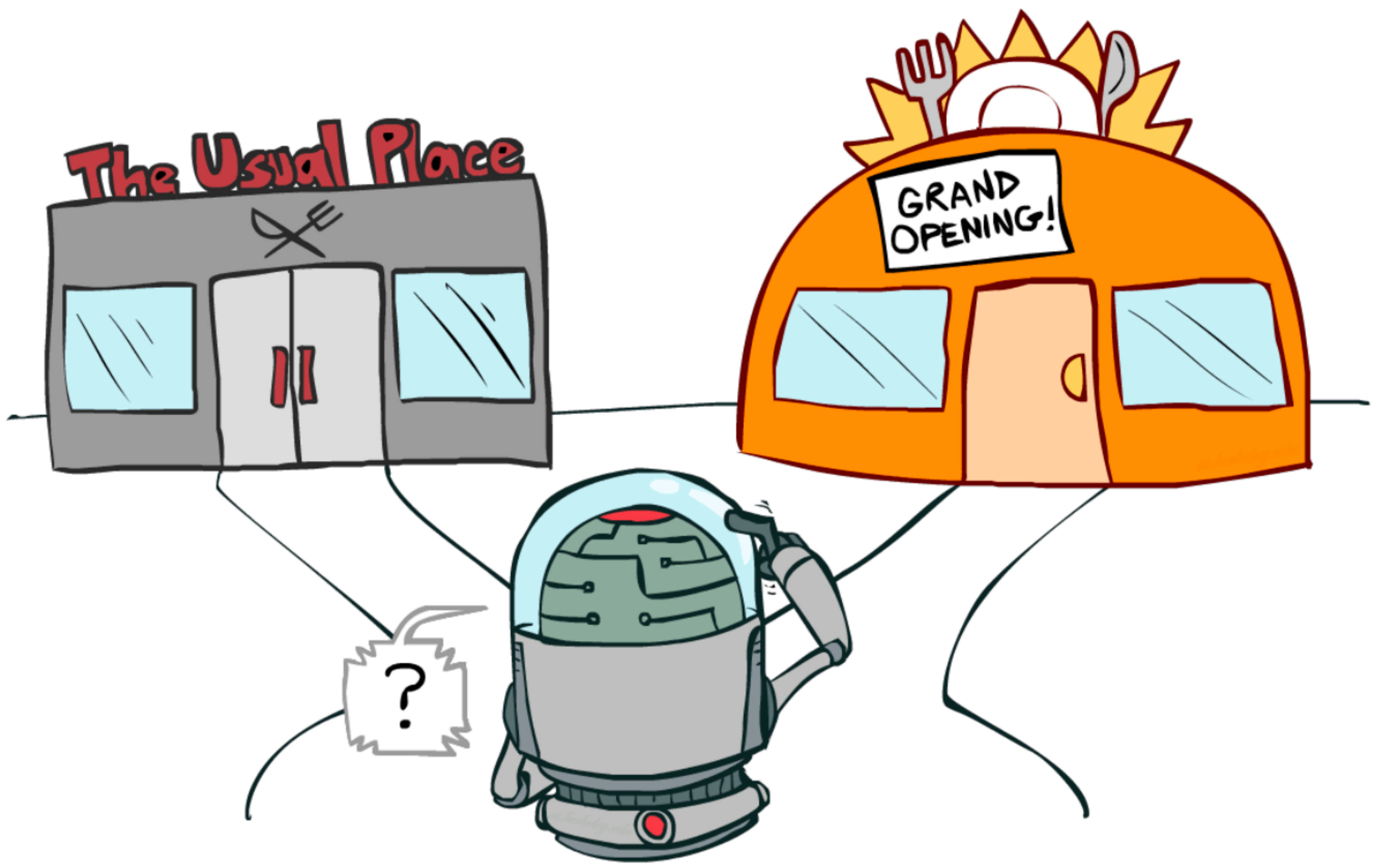
Approximate Dynamic Programming

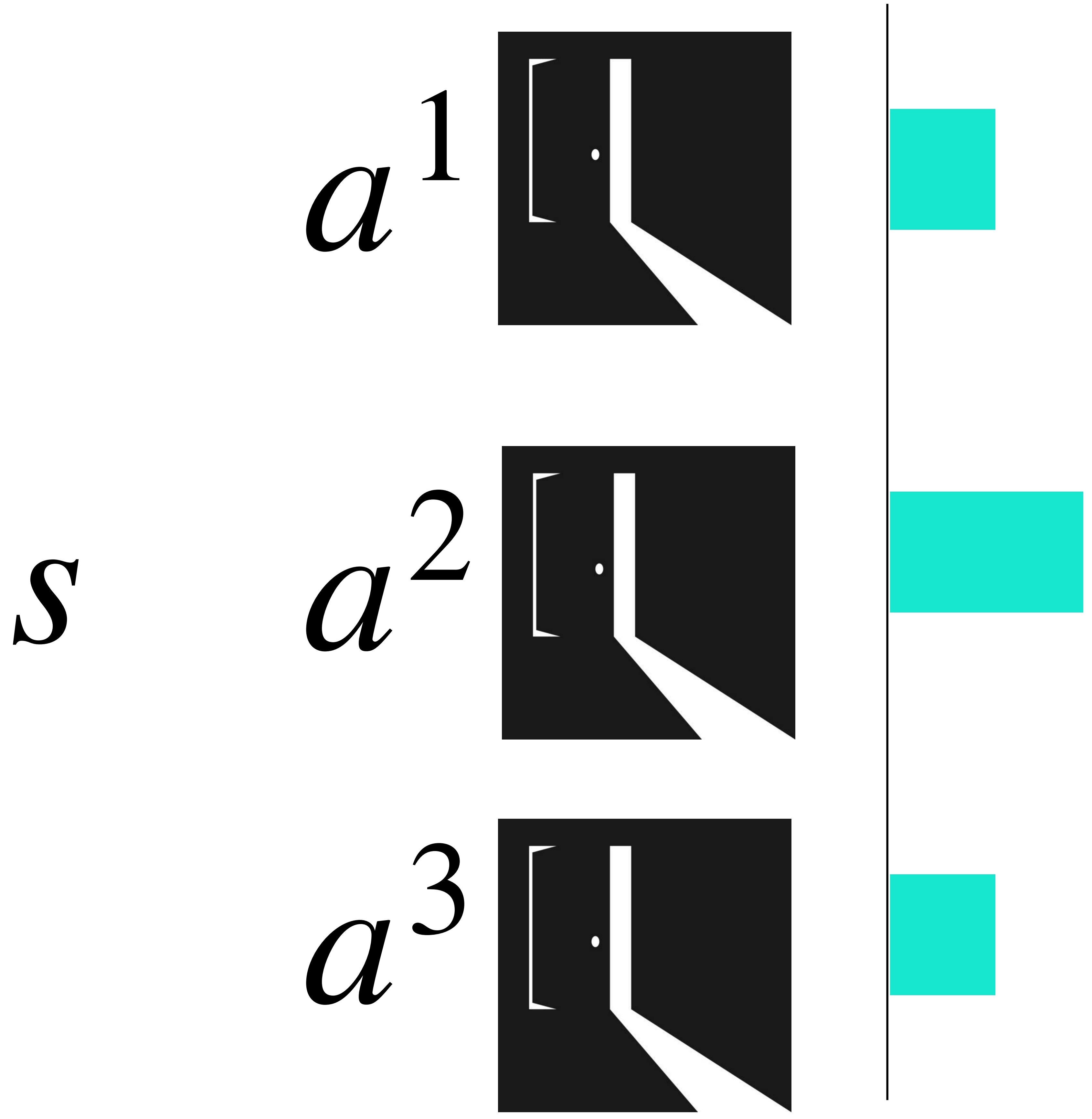
Sanjiban Choudhury

Cornell Bowers CIS
Computer Science

Recap: Two Ingredients of RL

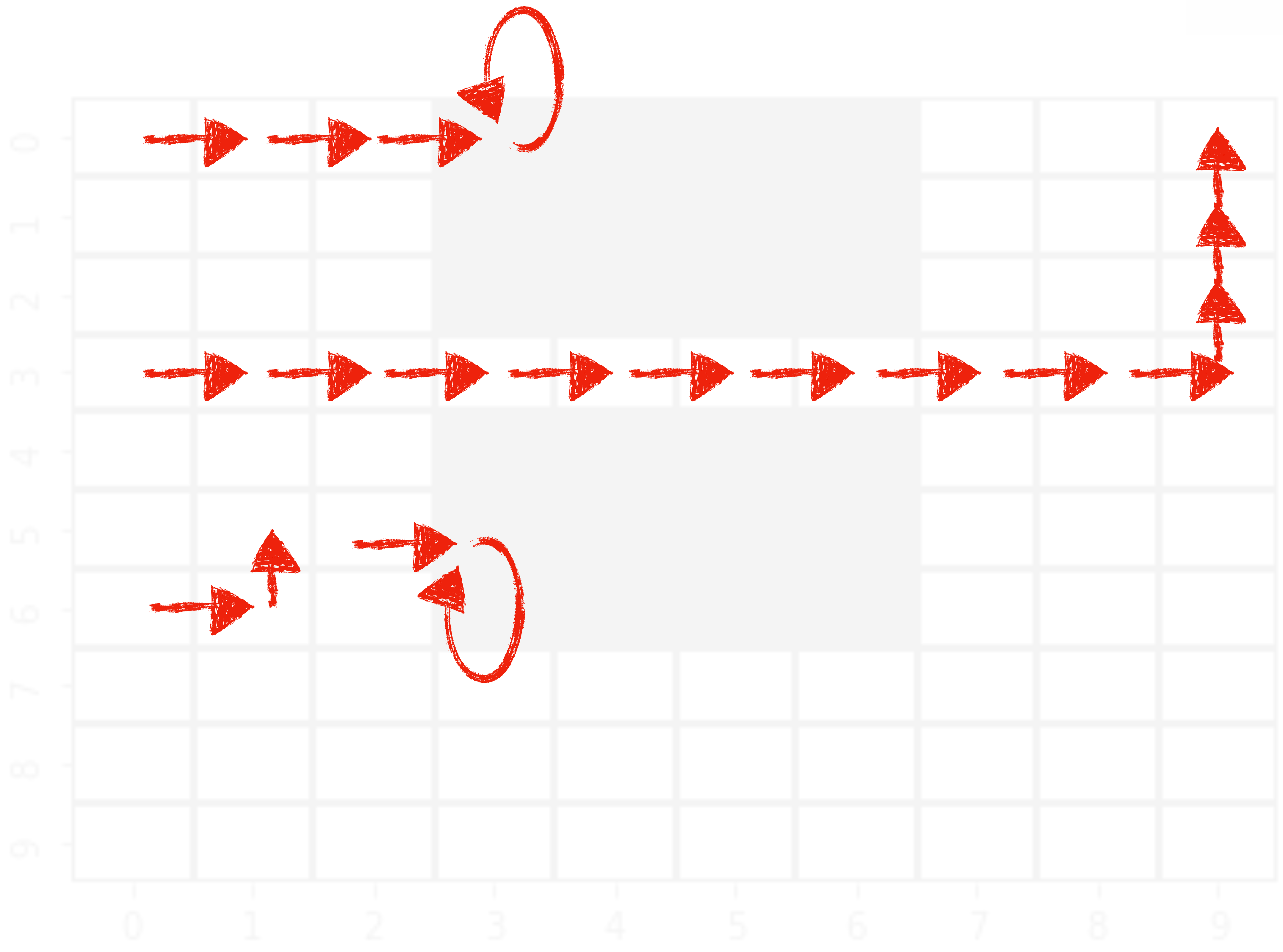


Exploration Exploitation

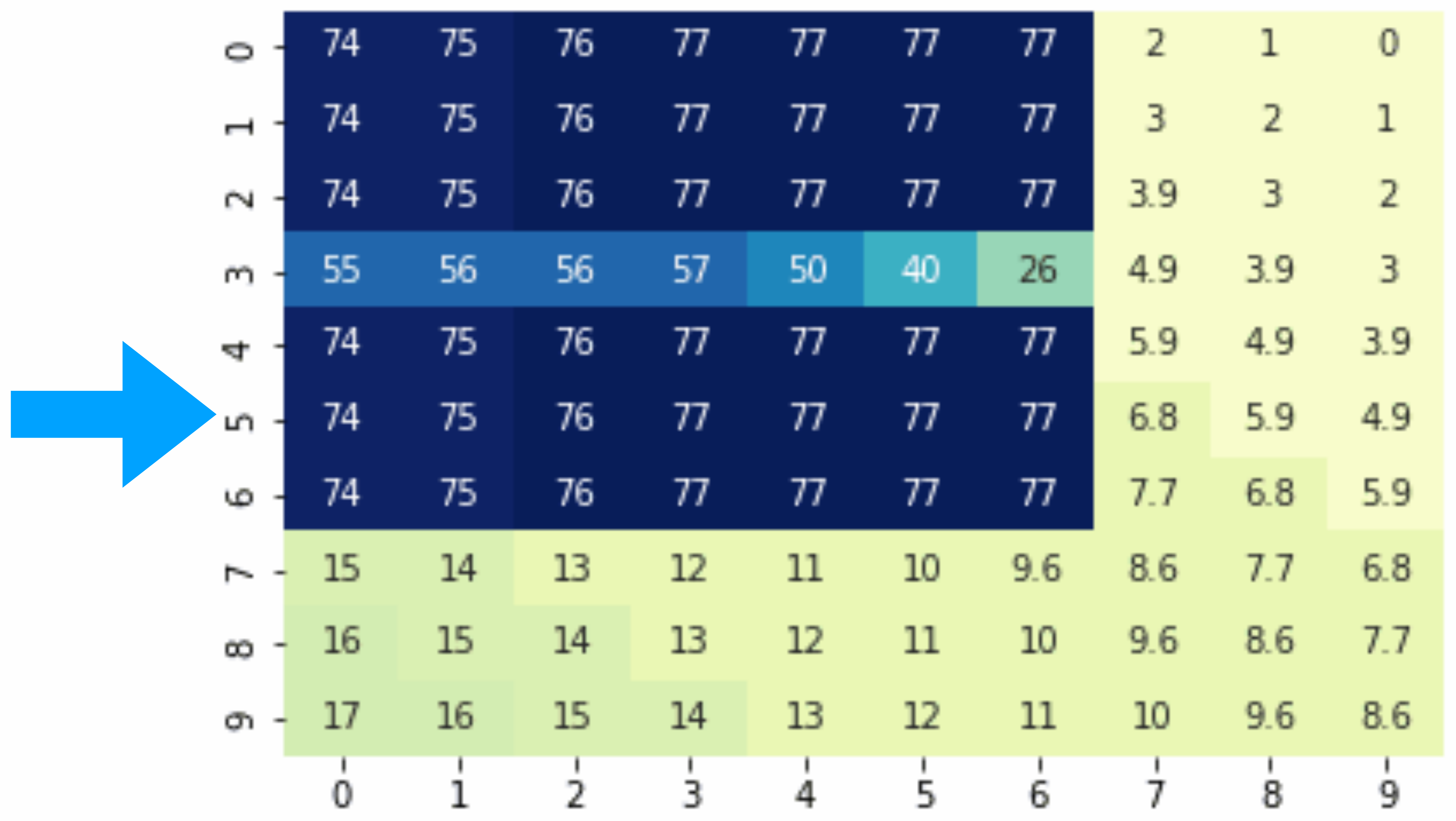


Estimate Values $Q(s, a)$

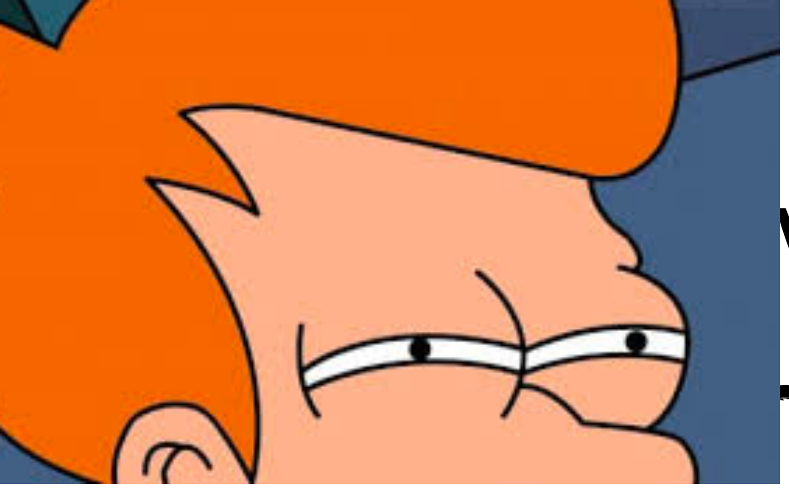
Estimate the value of policy from sample rollouts



Roll outs



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



Monte-Carlo

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

Zero Bias

High Variance

Always convergence

(Just have to wait till heat death of the universe)

Temporal Difference

$$V(s) \leftarrow V(s) + \alpha(c + \gamma V(s') - V(s))$$

Can have bias

Low Variance

May *not* converge if using function approximation

Last lecture, we looked at tabular values …

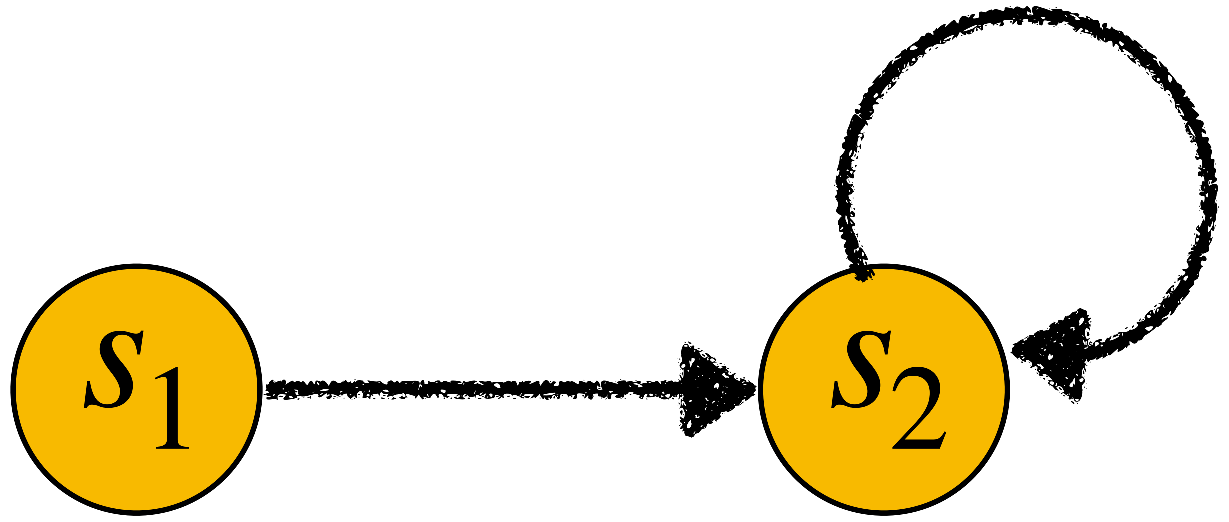
Is it trivial to upgrade to a Neural Network?

Activity!

A *tiny* MDP

Reward for being at any state is 0.0

Discount factor $\gamma = 0.9$



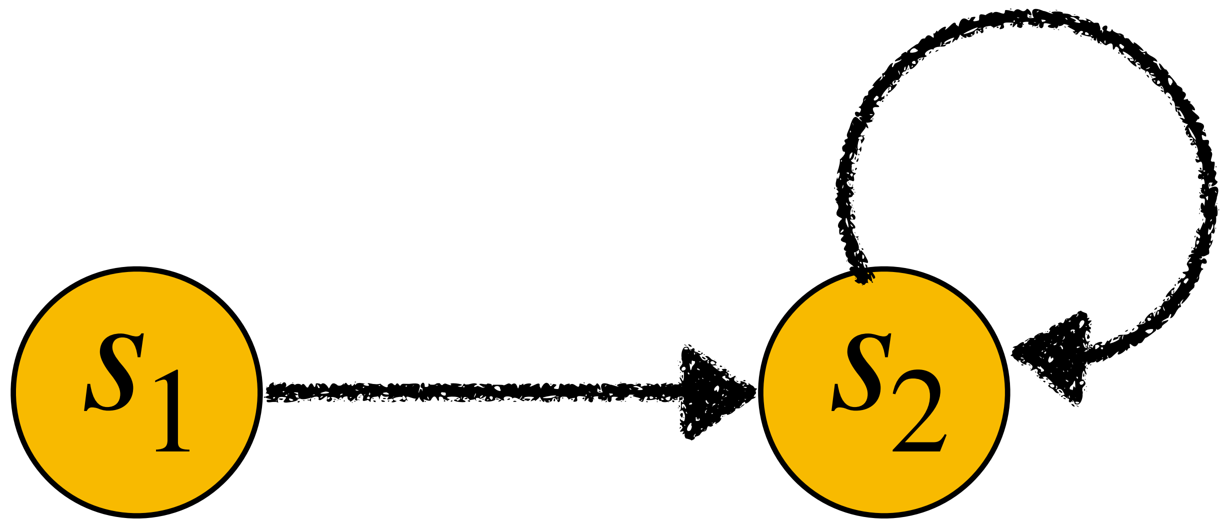
What happens when you run value iteration?

(Initialize with random values, say $V(s_1) = 1$ and 2)

A *tiny* MDP

Reward for being at any state is 0.0

Discount factor $\gamma = 0.9$



$f(s_1) = 1$ $f(s_1) = 2$

Let's say we want to use a *linear value function approximator*

$$V(s) = wf(s) = w * \begin{cases} 1 & if\ s = s_1 \\ 2 & if\ s = s_2 \end{cases}$$

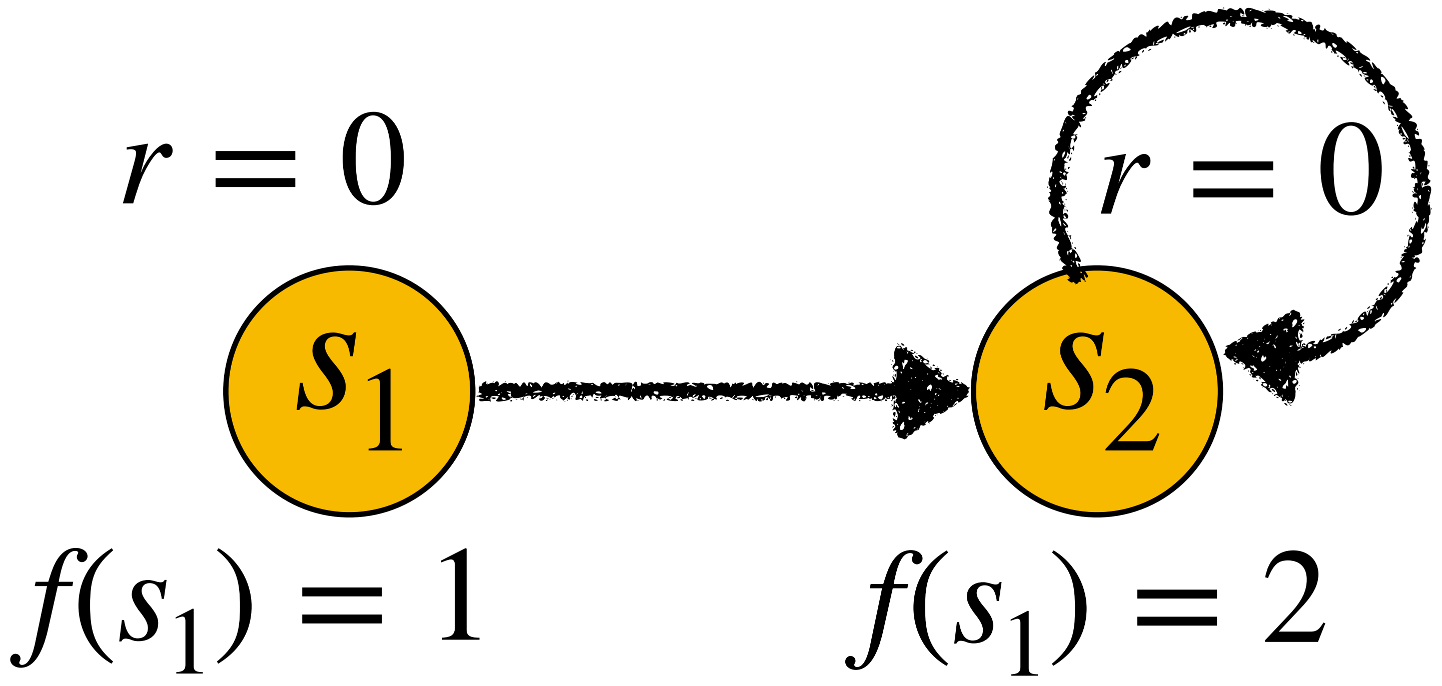
What happens if you run value iteration? (Initialize with w=1)

Think-Pair-Share

Think (30 sec): Initialize value iteration with w=1. What happens? What's the explanation?

Pair: Find a partner

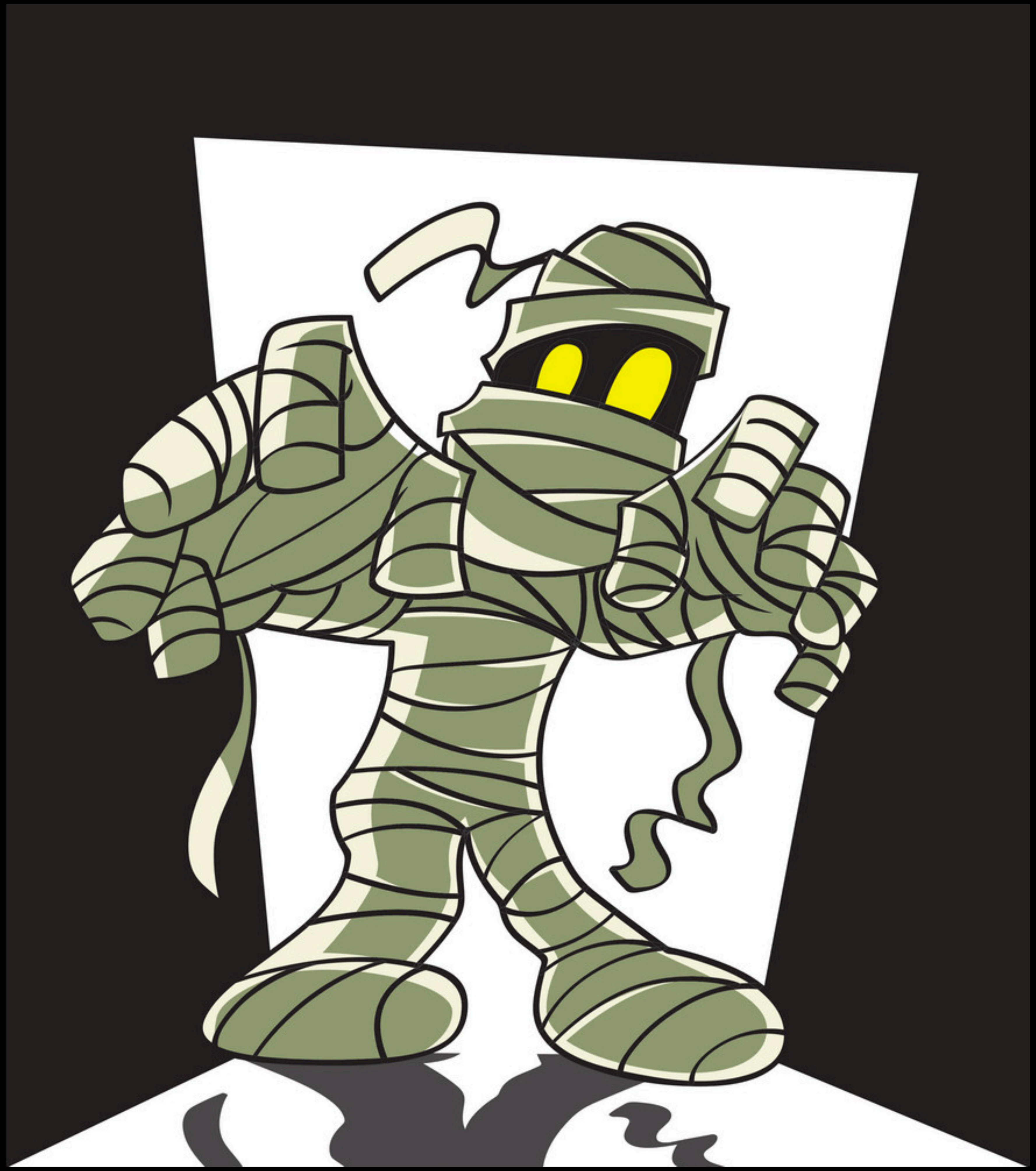
Share (45 sec): Partners exchange ideas



$$V(s) = wf(s)$$

Init with $w = 1$

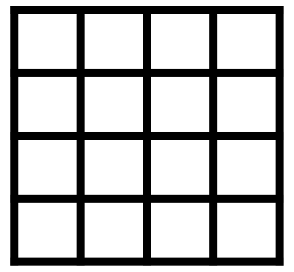
CURSE OF APPROXIMATION!



From dynamic programming to *Fitted* dynamic programming

Approximate (Fitted) Value Iteration

Q-iteration



$Q(s, a) \leftarrow 0$

while *not converged* **do**

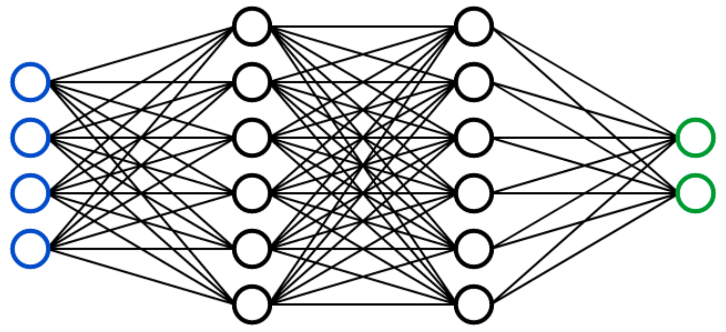
  for $s \in S, a \in A$

    $Q^{new}(s, a) = c(s, a) + \gamma \mathbb{E}_{s'} \min_{a'} Q(s', a')$

  $Q \leftarrow Q^{new}$

return Q

Fitted Q-iteration



Given $\{s_i, a_i, c_i, s_i'\}_{i=1}^N$

Init $Q_\theta(s, a) \leftarrow 0$

while *not converged* **do**

  $D \leftarrow \varnothing$

  for $i \in 1, \ldots, n$

    input $\leftarrow \{s_i, a_i\}$,

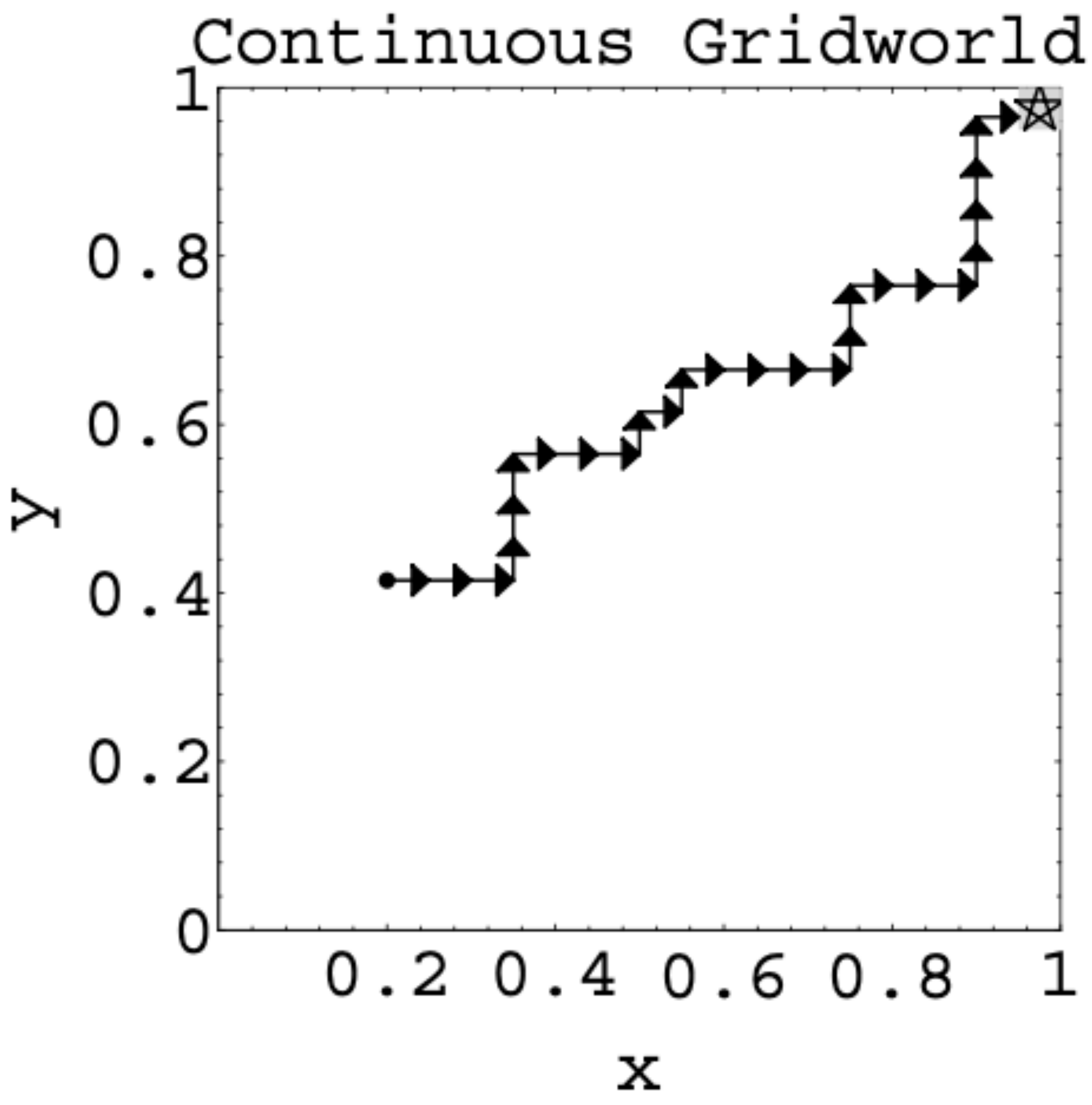
    target $\leftarrow c_i + \gamma \min_a Q_\theta(s_i', a')$

    $D \leftarrow D \cup$ {input, output}

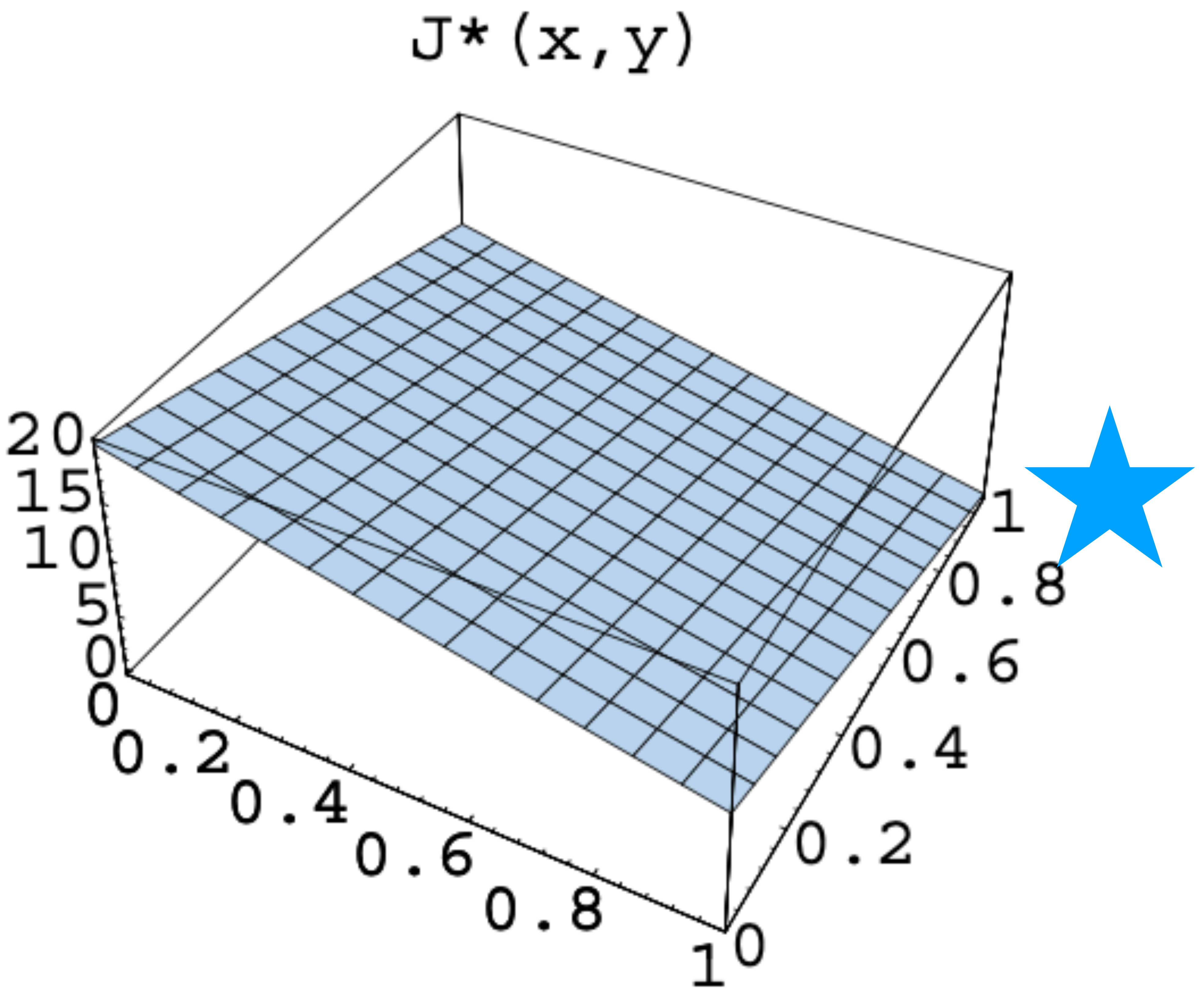
  $Q_\theta \leftarrow$ **Train**(D)

return Q_θ

A simple example: Gridworld



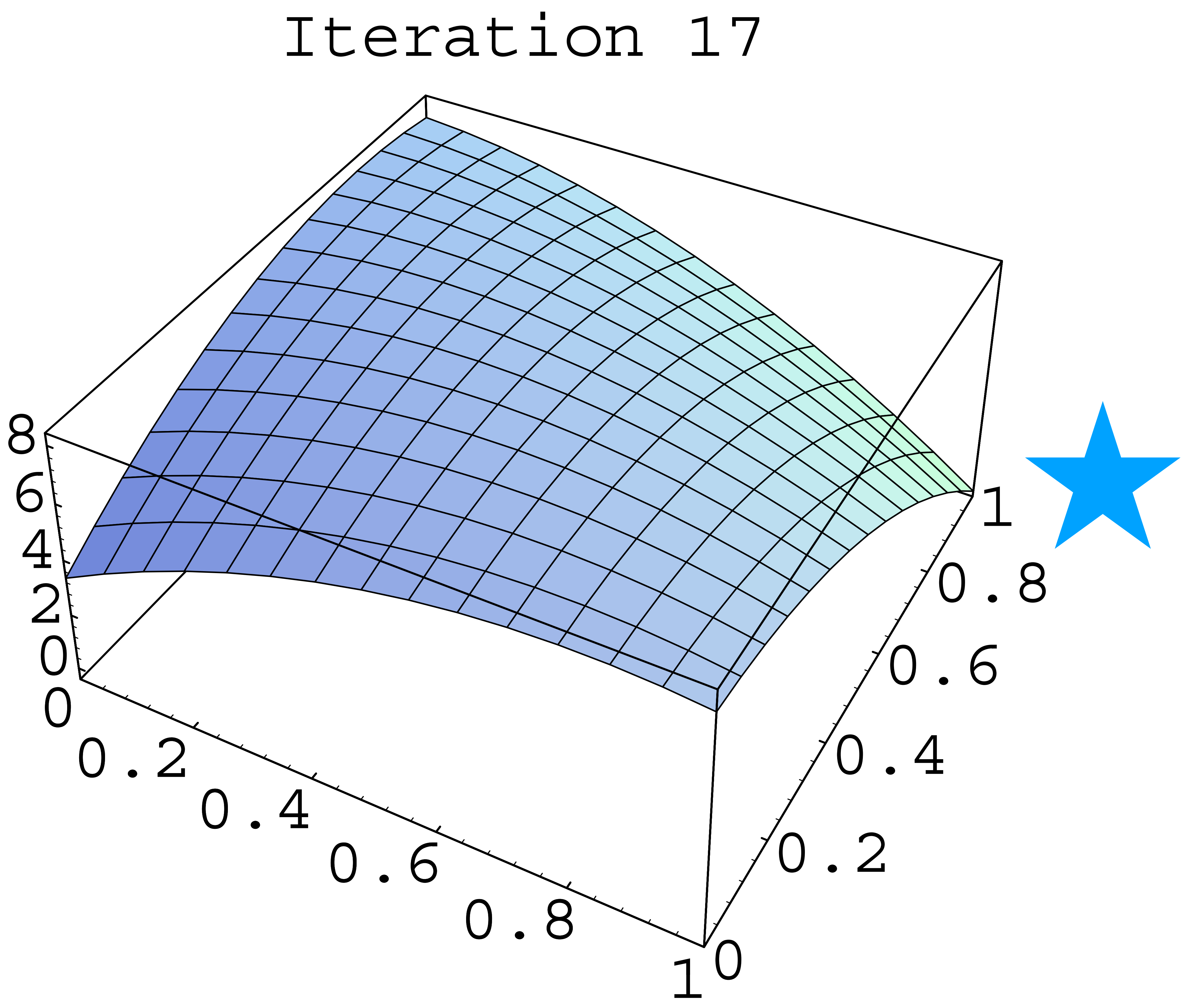
Optimal path



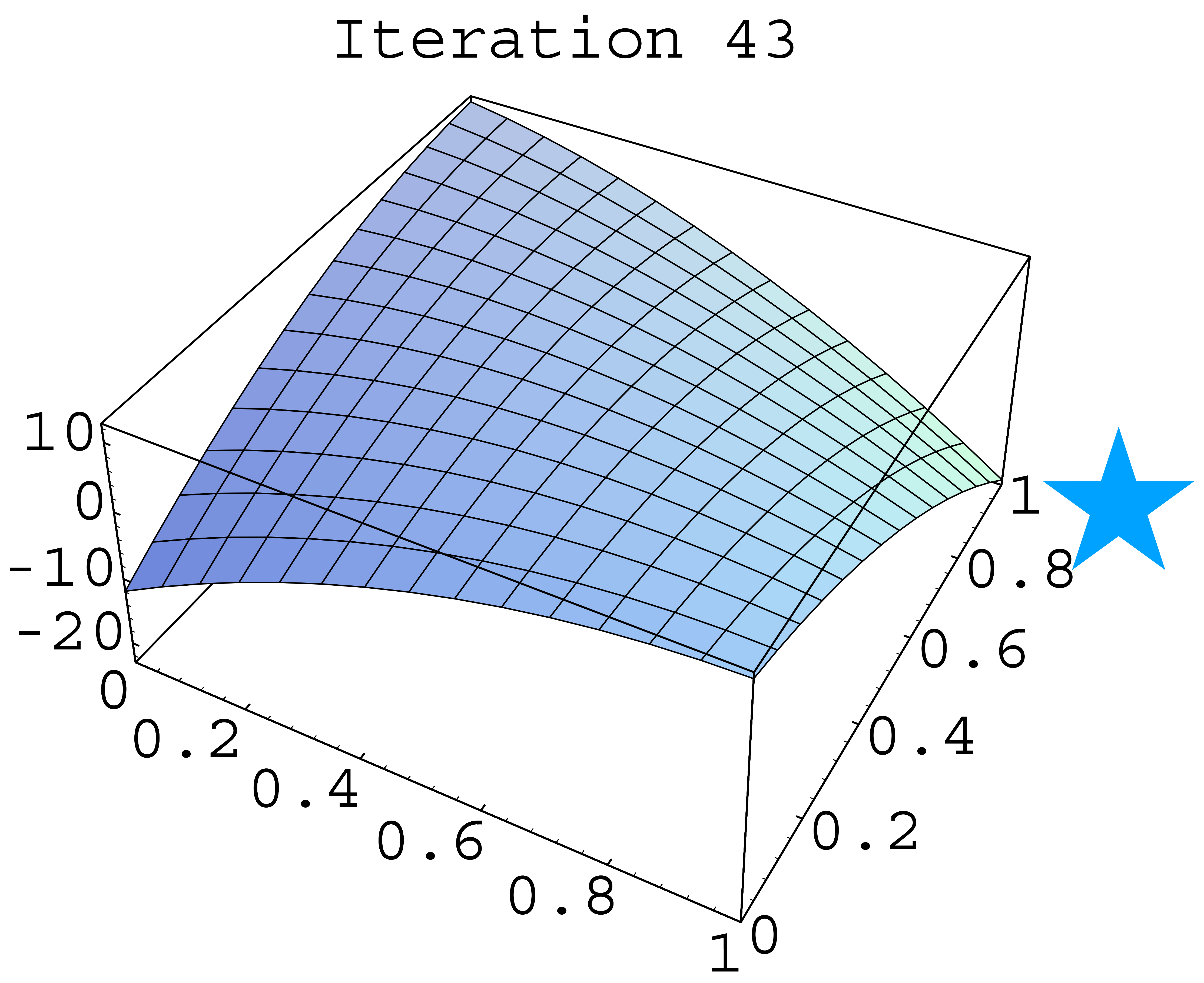
True value function

Boyan,Justin A and Moore, Andrew W, Generalization in Reinforcement Learning: Safely Approximating the Value Function. NeurIPS 1994

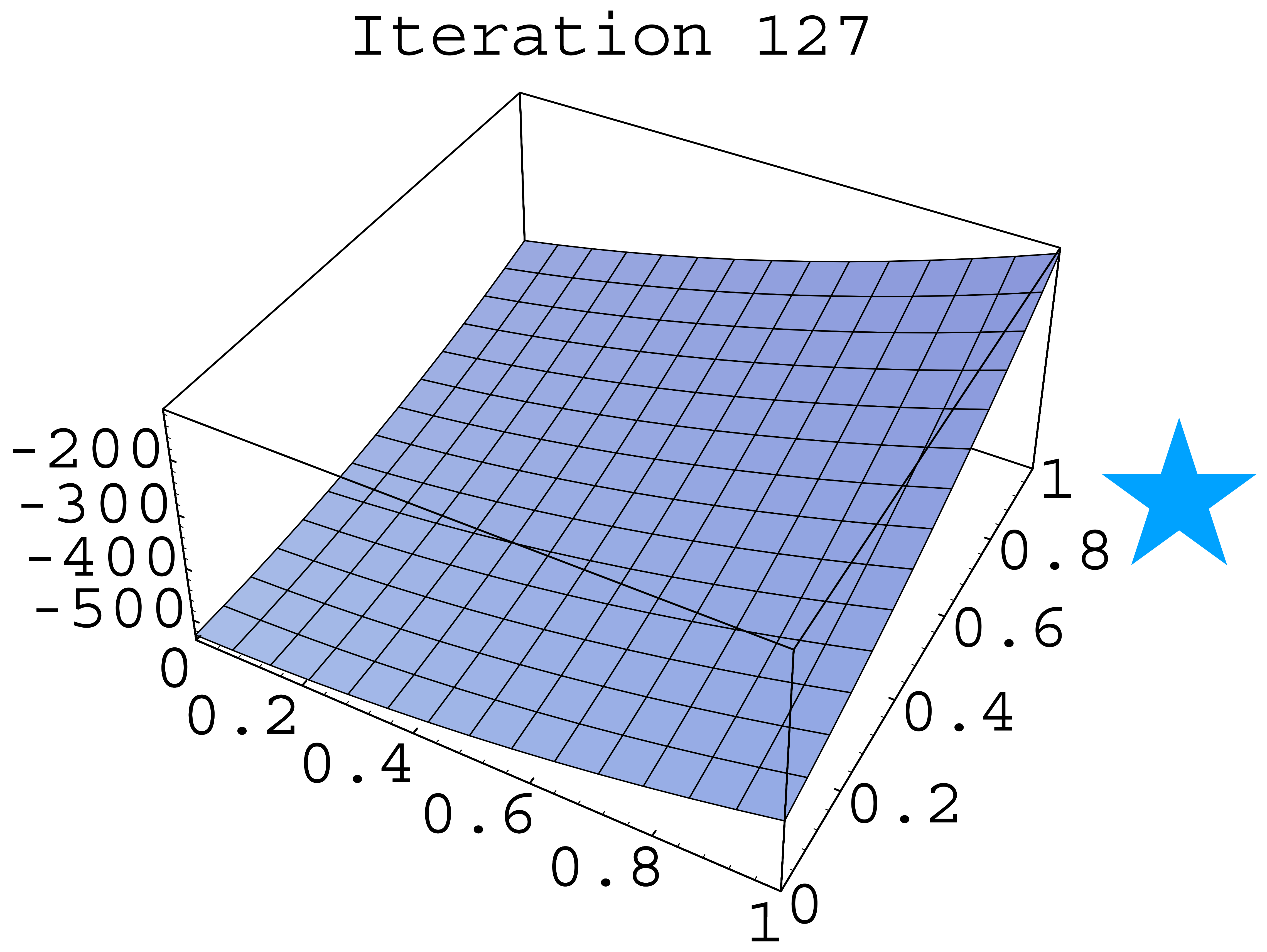
What happens when we run value iteration with a *quadratic*?



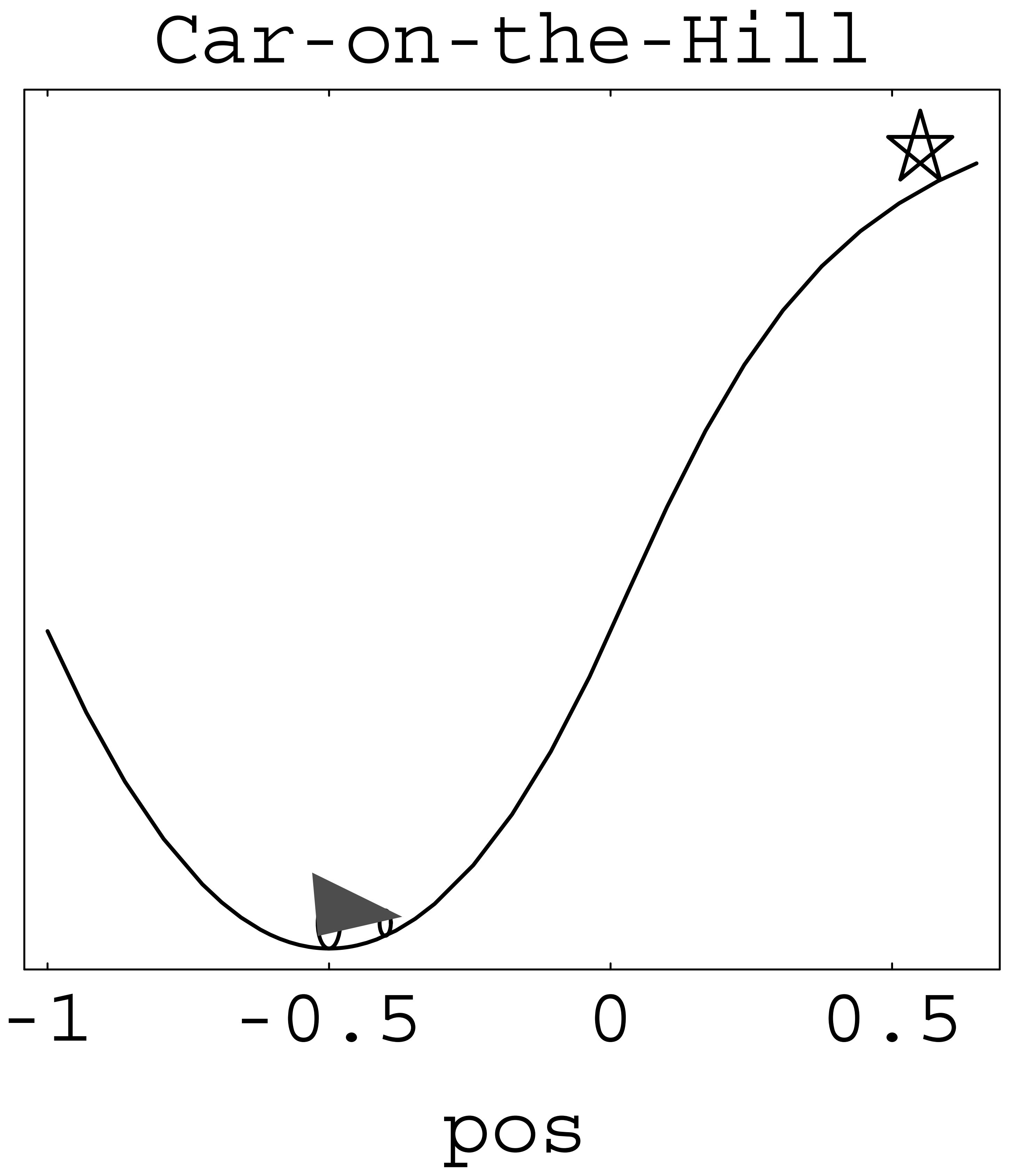
What happens when we run value iteration with a *quadratic*?

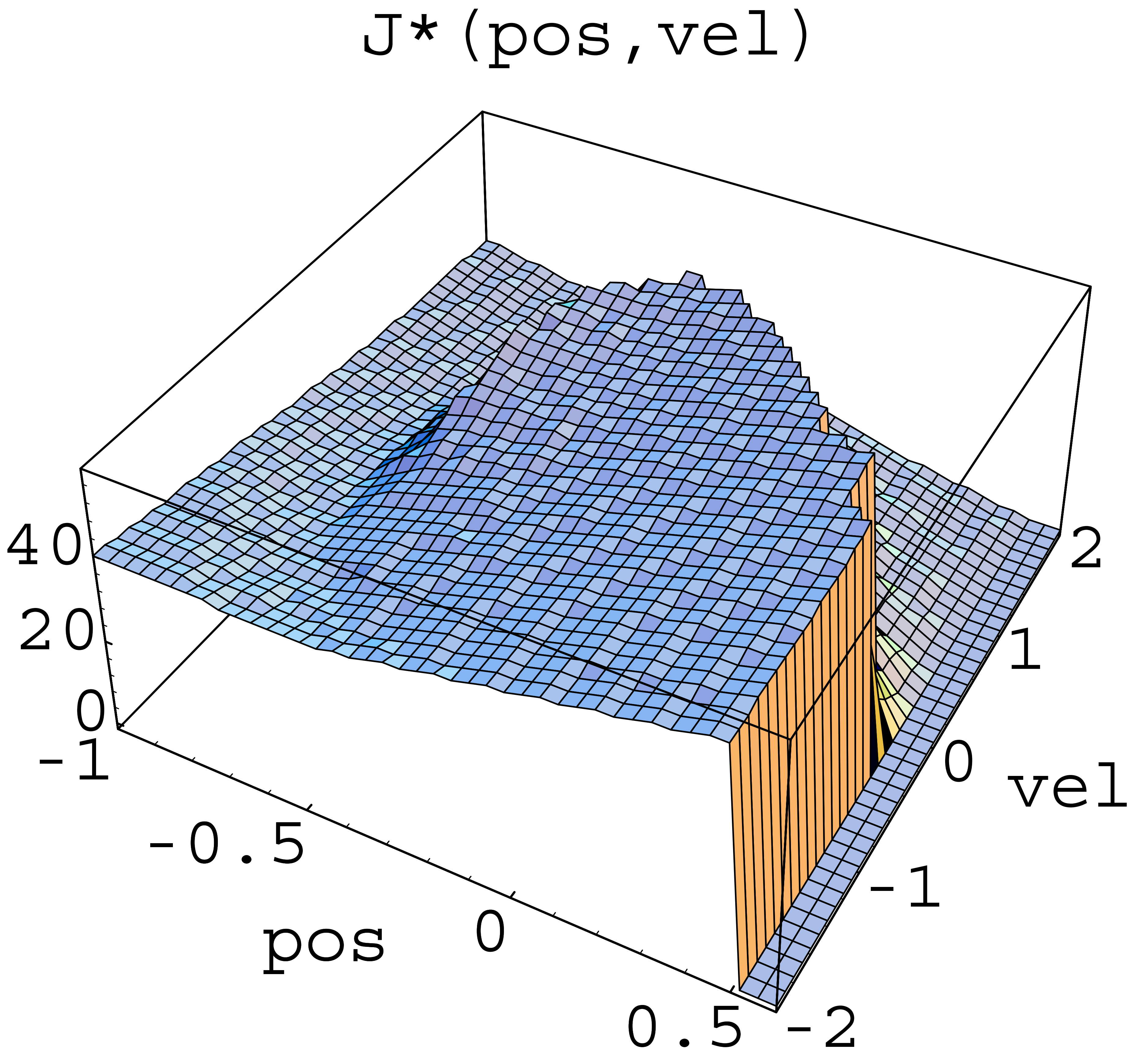


What happens when we run value iteration with a *quadratic*?

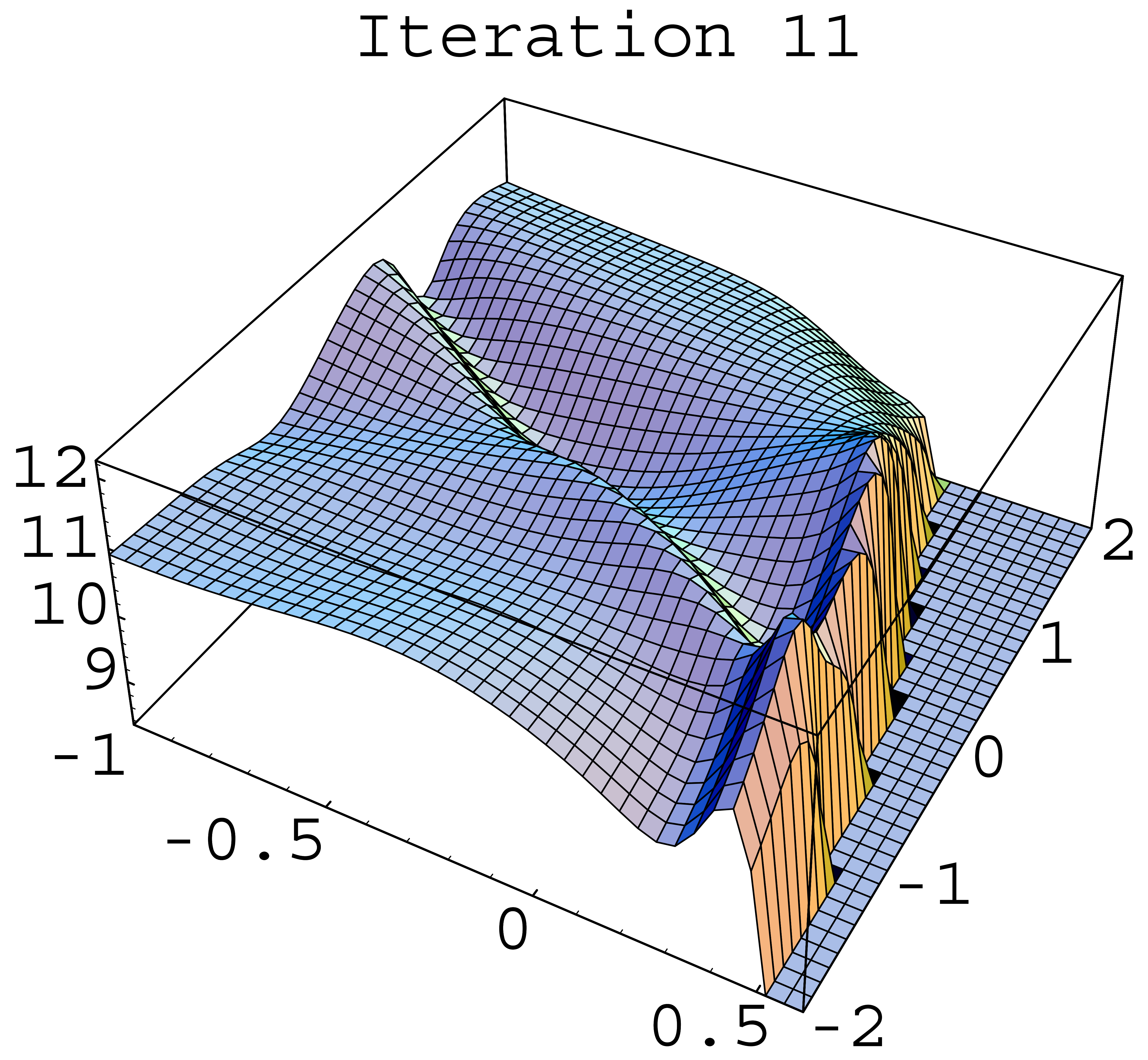


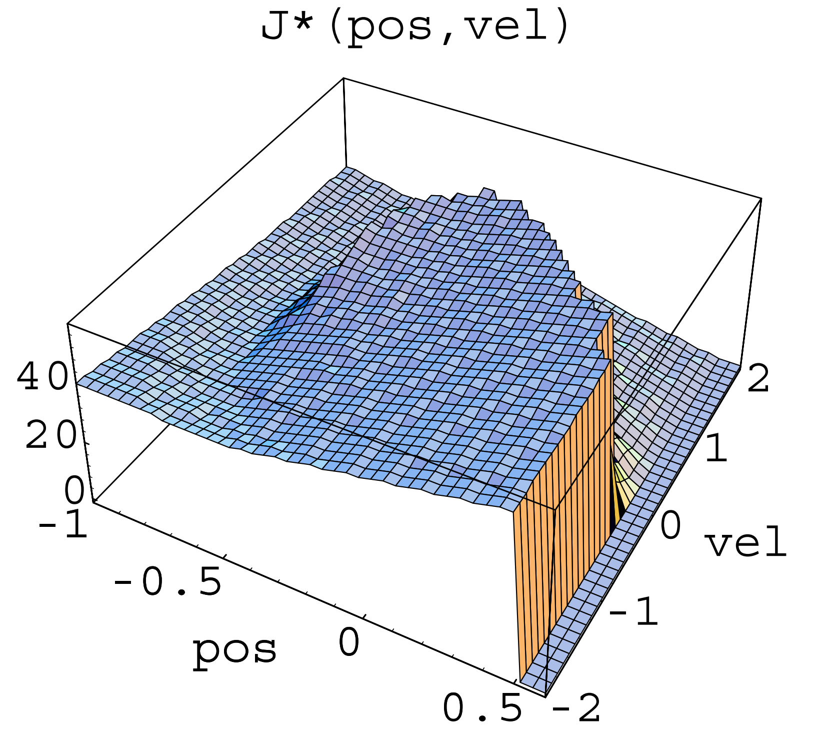
Another Example: Mountain Car!



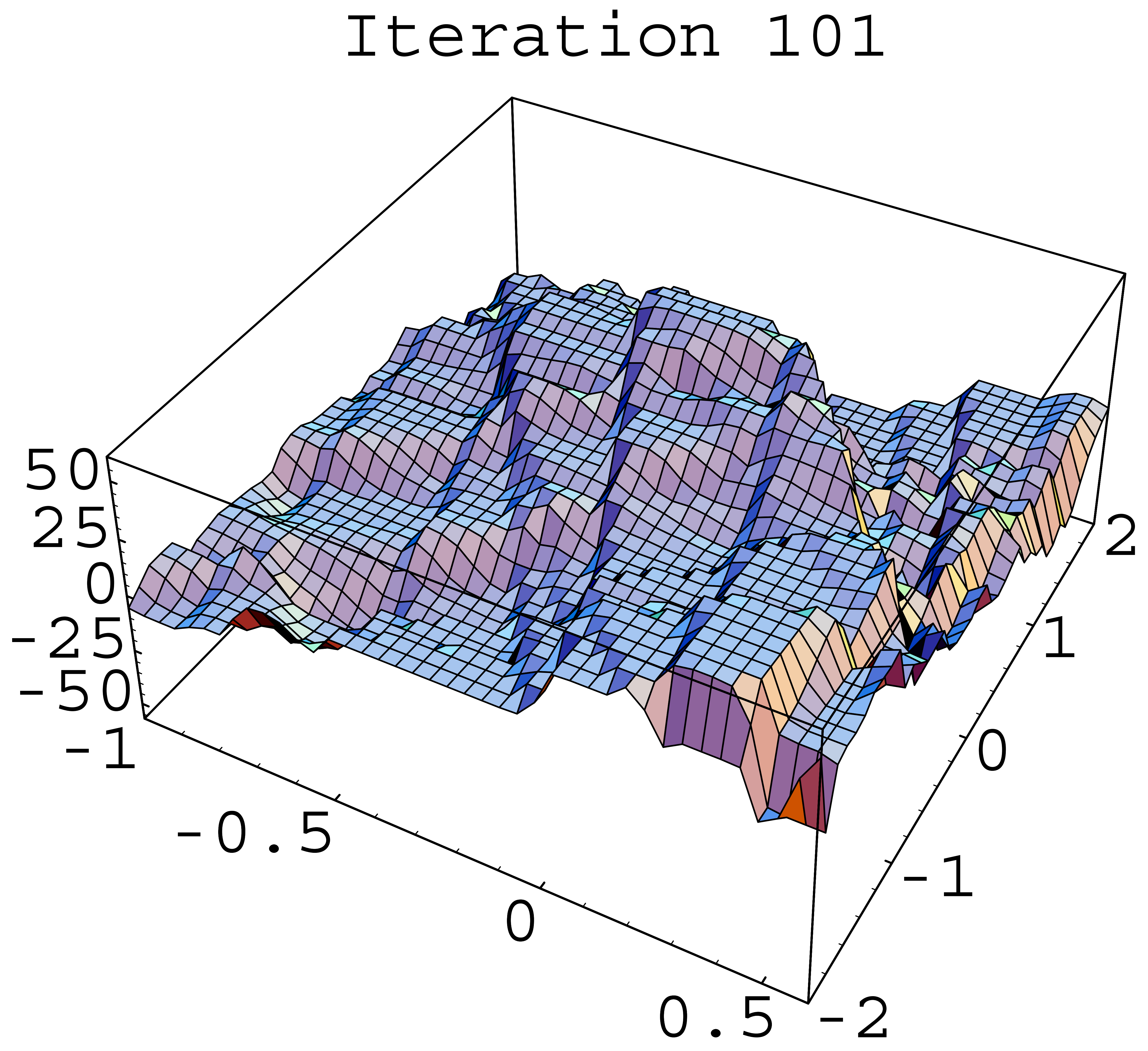


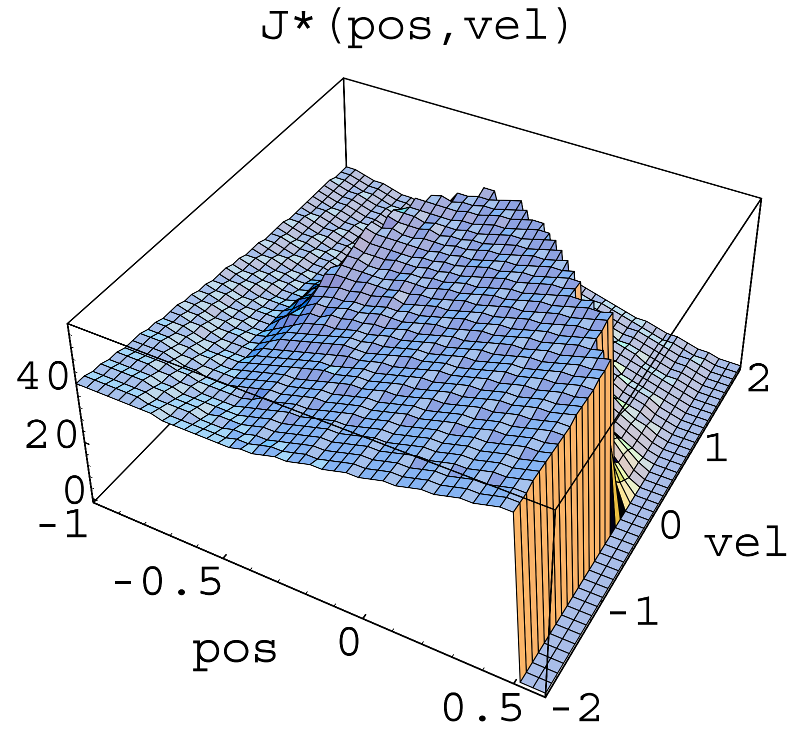
What happens when we run value iteration with a *2 Layer MLP?*



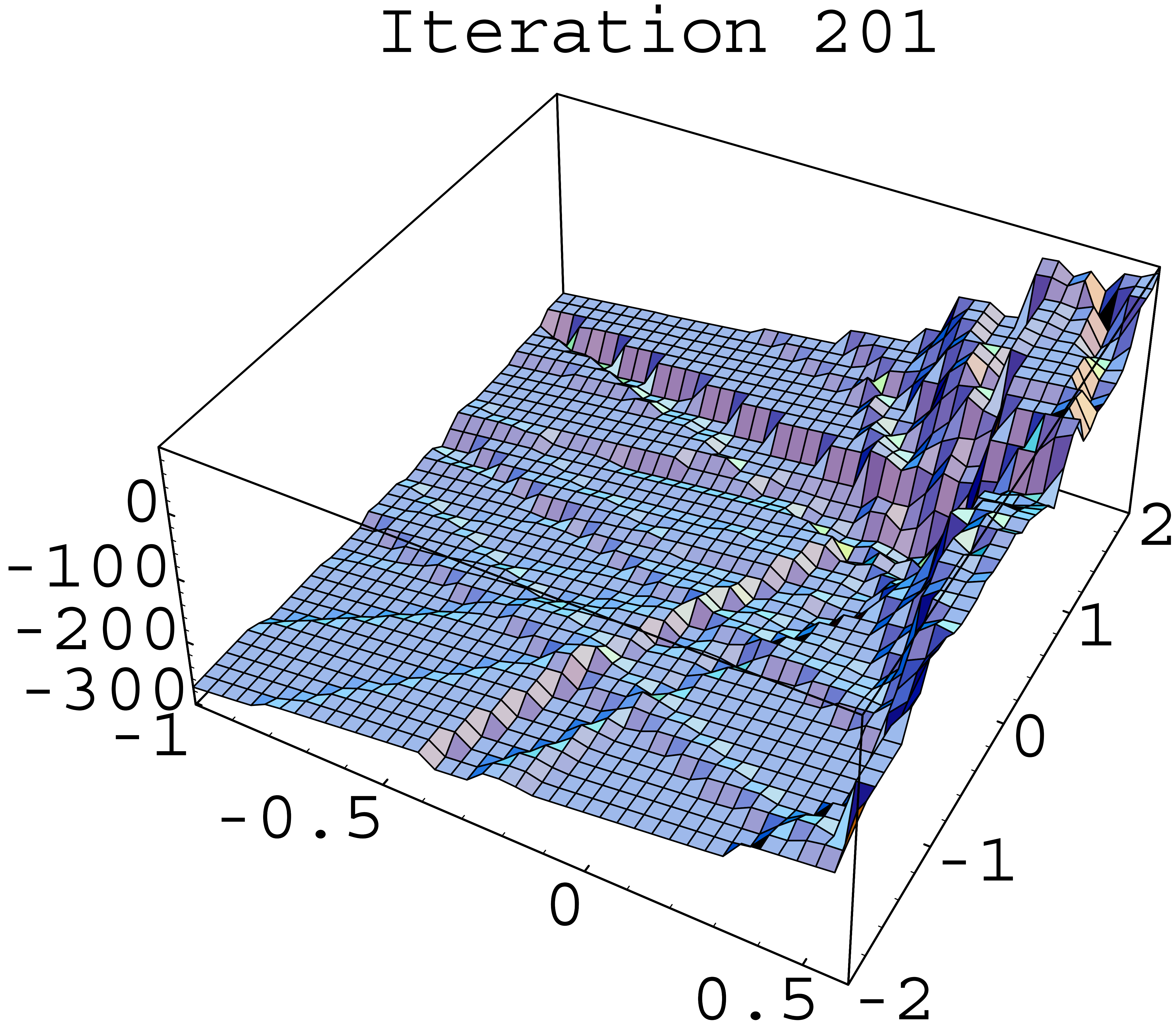


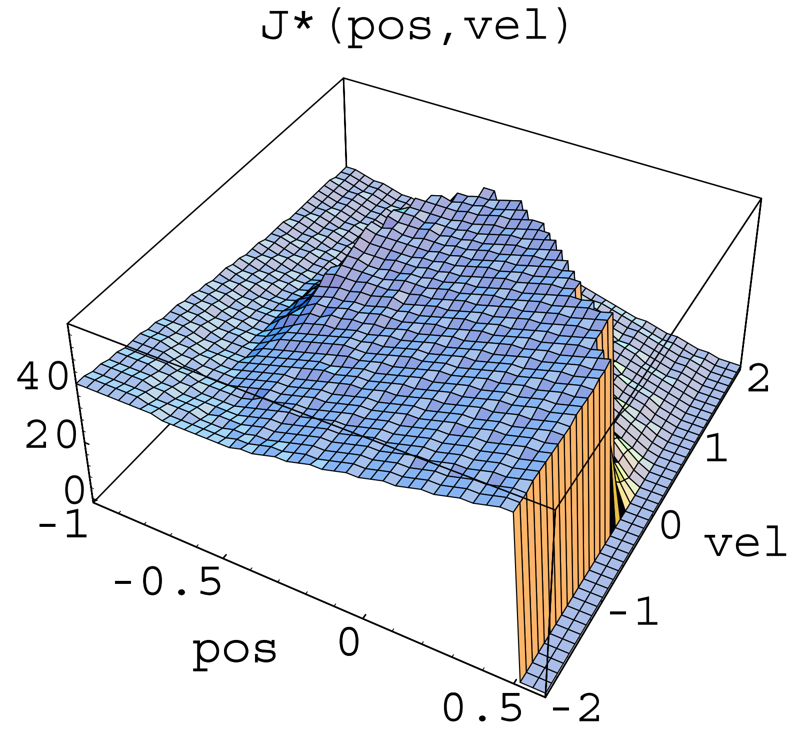
What happens when we run value iteration with a *2 Layer MLP?*





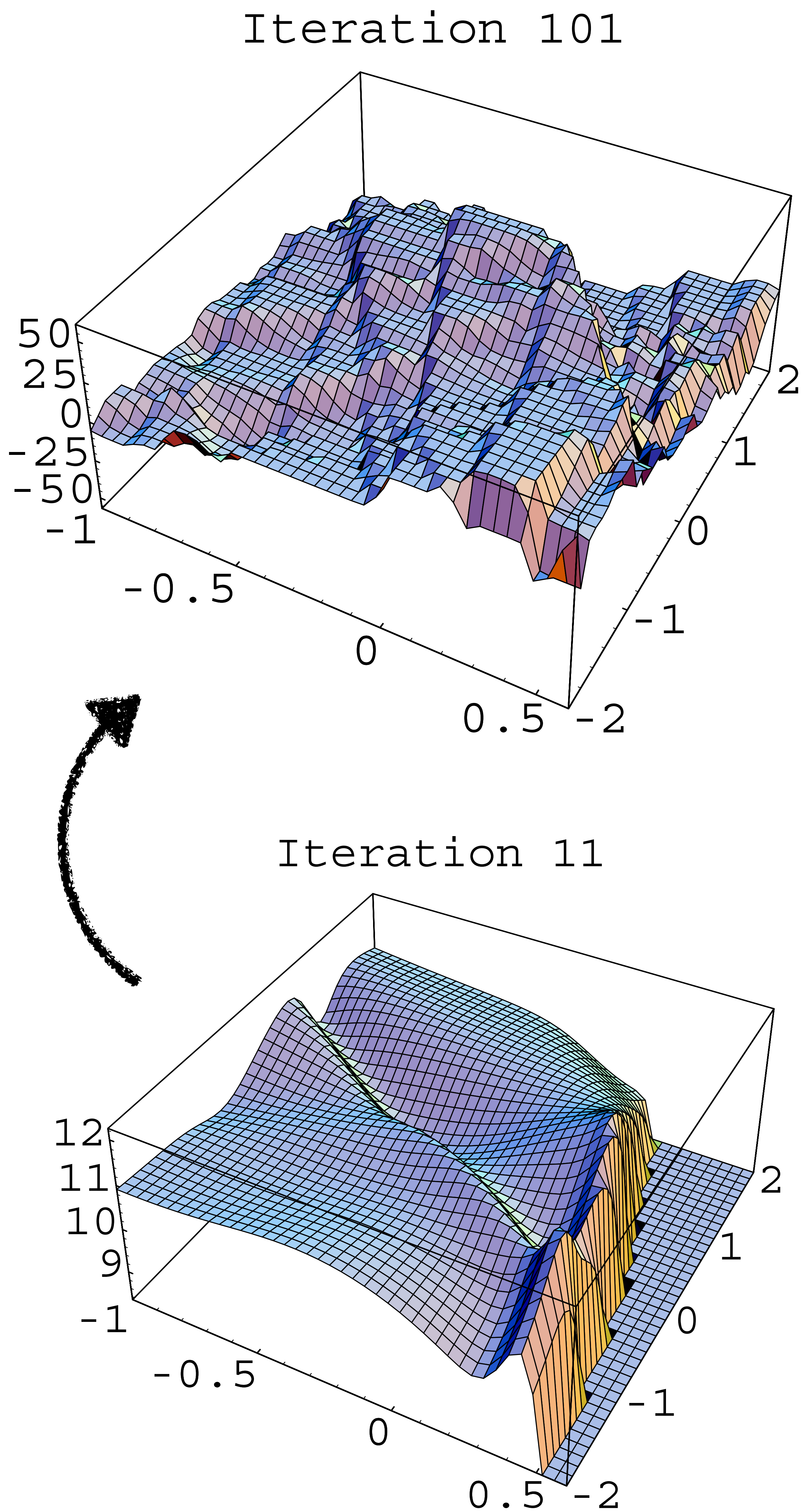
What happens when we run value iteration with a *2 Layer MLP?*





The problem of Bootstrapping!

max()



The problem of Bootstrapping!

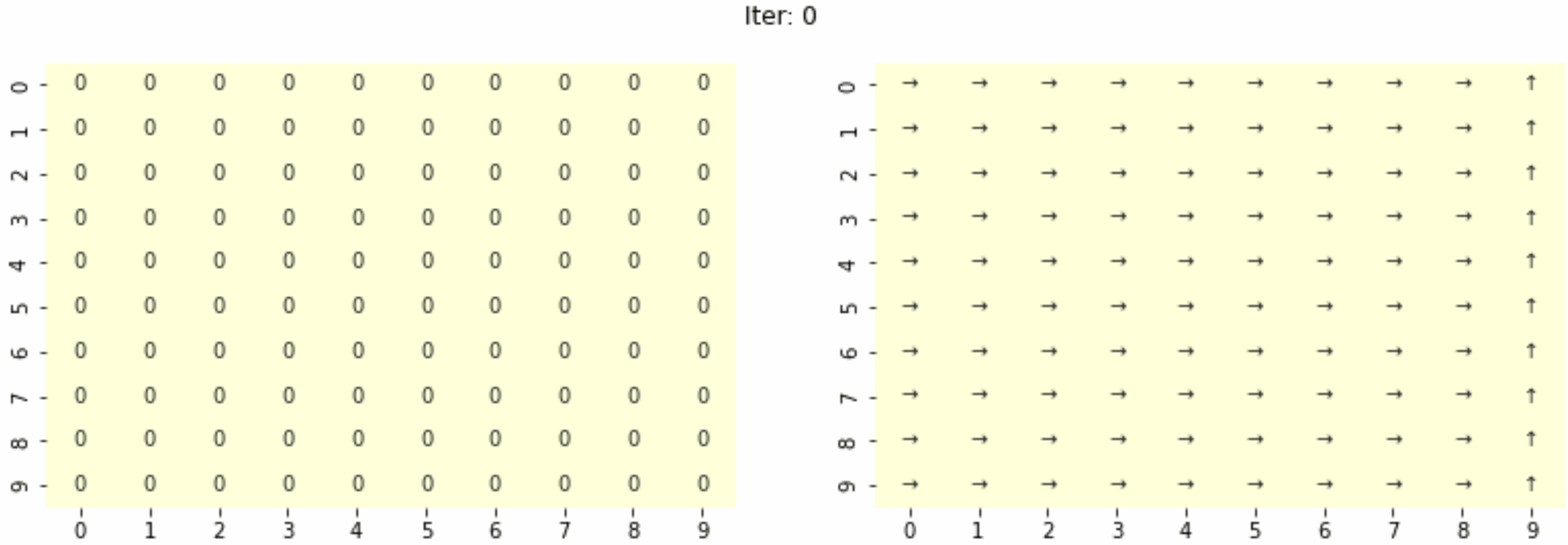
- Since these methods rely on approximating the value function inductively, errors in approximation are propagated, and, even worse, amplified as the algorithm encourages actions that lead to states with sub-optimal values.
- The key reason behind this is the minimization operation performed when generating the target value used for the action value function. The minimization operation pushes the desired policy to visit states where the value function approximation is less than the true value of that state – that is to say, states that look more attractive than they should. This typically happens in areas of state spaces which are very few training samples and could, in fact, be quite bad places to arrive.

What about policy iteration?

Policy Iteration

Policy Evaluation

Policy Improvement



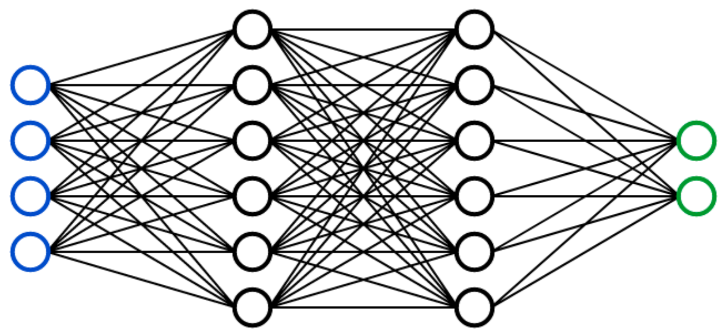
$$Q^{\pi}(s,a) = c(s,a)) + \gamma \mathbb{E}_{s' \sim \mathcal{T}(s,a)} Q^{\pi}(s', \pi(s')))]$$

$$\pi^{+}(s) = \arg\min_{a} Q^{\pi}(s,a)$$

Approximate (Fitted) Policy Iteration

Fitted policy evaluation

Given $\{s_i, a_i, c_i, s_i'\}_{i=1}^N$



Init $Q_\theta(s,a) \leftarrow 0$

while *not converged* **do**

$D \leftarrow \emptyset$

for $i \in 1, \dots, n$

input $\leftarrow \{s_i, a_i\}$

target $\leftarrow c_i + \gamma Q_\theta(s_i', \pi(s_i'))$

$D \leftarrow D \cup$ {input, output}

$Q_\theta \leftarrow$ **Train**(D)

return Q_θ

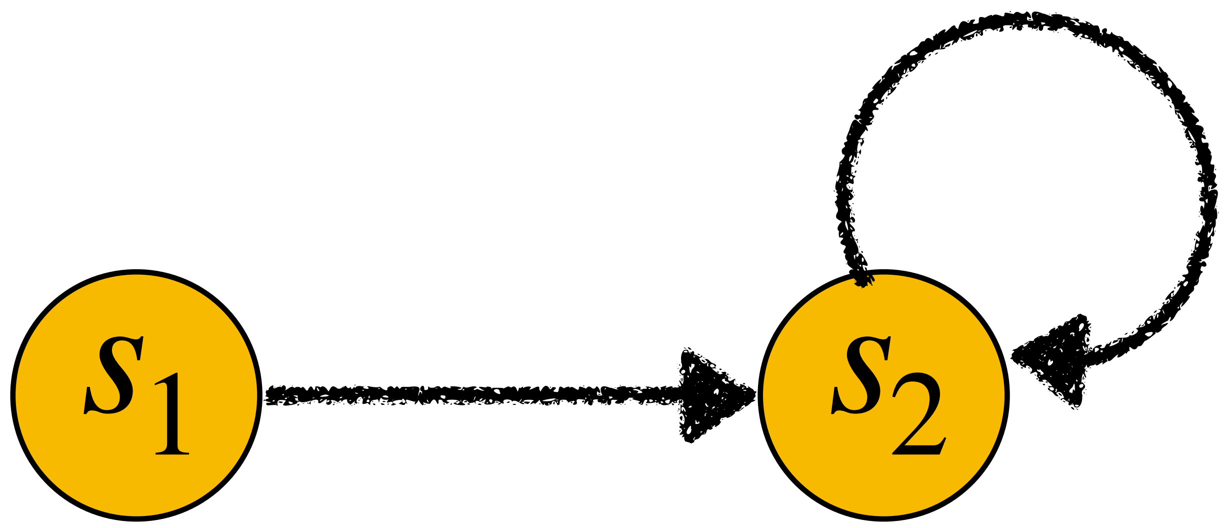
Policy Improvement

This remains the same!

$$\pi^+(s) = \arg\min_a Q^\pi(s,a)$$

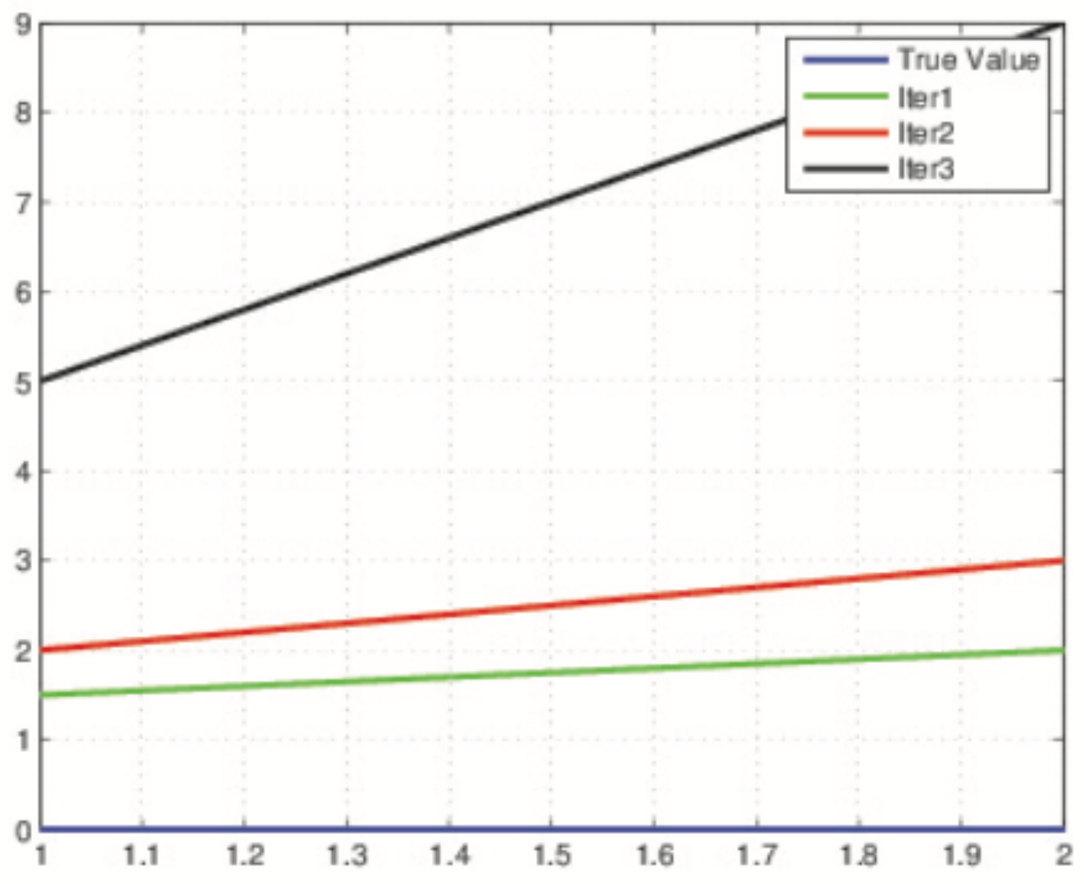
Surely approximate value evaluation is more stable than approximate value iteration? (There is no min()!)

Well … not quite

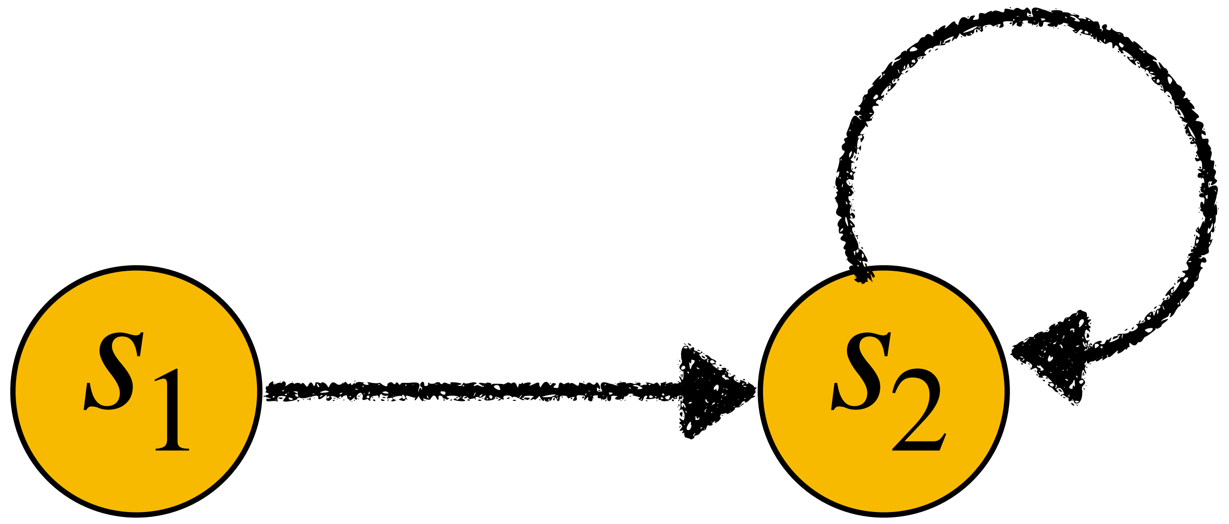


$f(s_1) = 1$ $f(s_1) = 2$

w blows up!



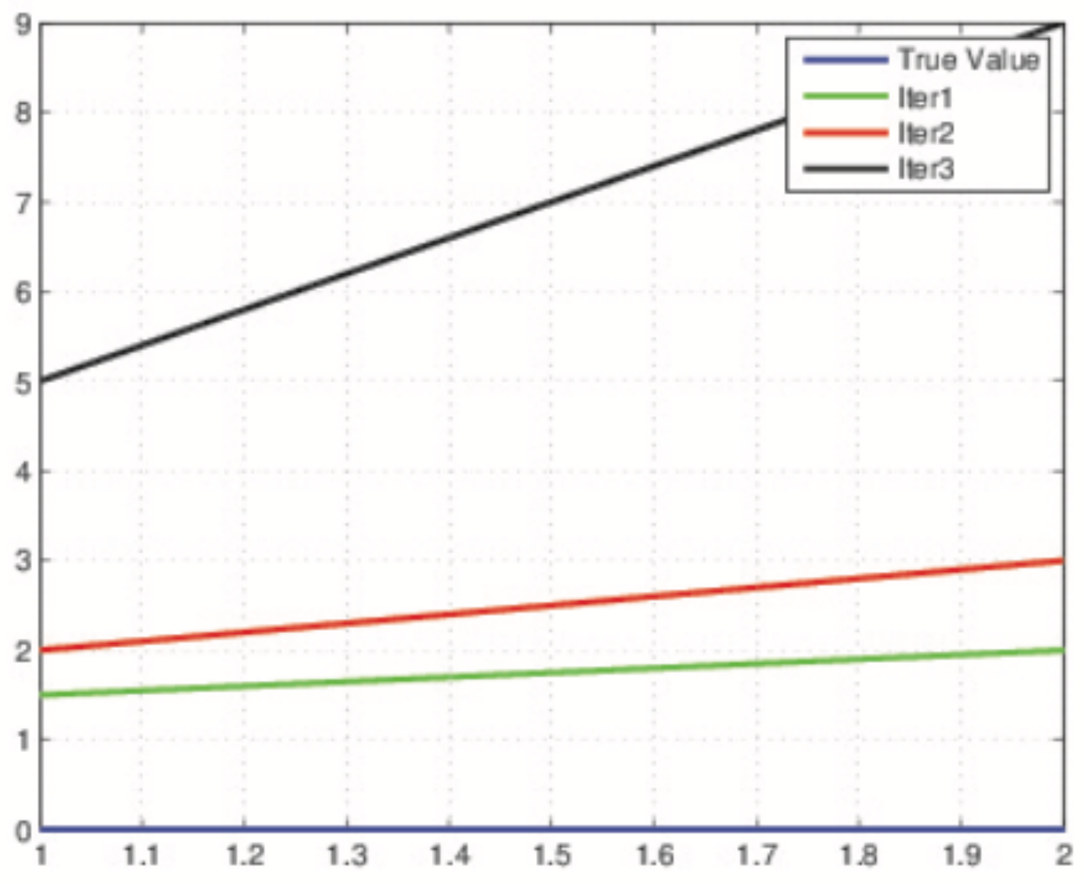
Well … not quite



$f(s_1) = 1$ $f(s_1) = 2$

But we can fix this by *on-policy weighting*

w blows up!



Weight each datapoint by how often the policy visits it.

But what about policy improvement?

Fitted policy evaluation

Given $\{s_i, a_i, c_i, s_i'\}_{i=1}^N$

Init $Q_\theta(s,a) \leftarrow 0$

while *not converged* **do**

$D \leftarrow \emptyset$

for $i \in 1, \ldots, n$

input $\leftarrow \{s_i, a_i\}$

target $\leftarrow c_i + \gamma Q_\theta(s_i', \pi(s_i'))$

$D \leftarrow D \cup$ {input, output}

$Q_\theta \leftarrow$ **Train**(D)

return Q_θ

This is fine..

Policy Improvement

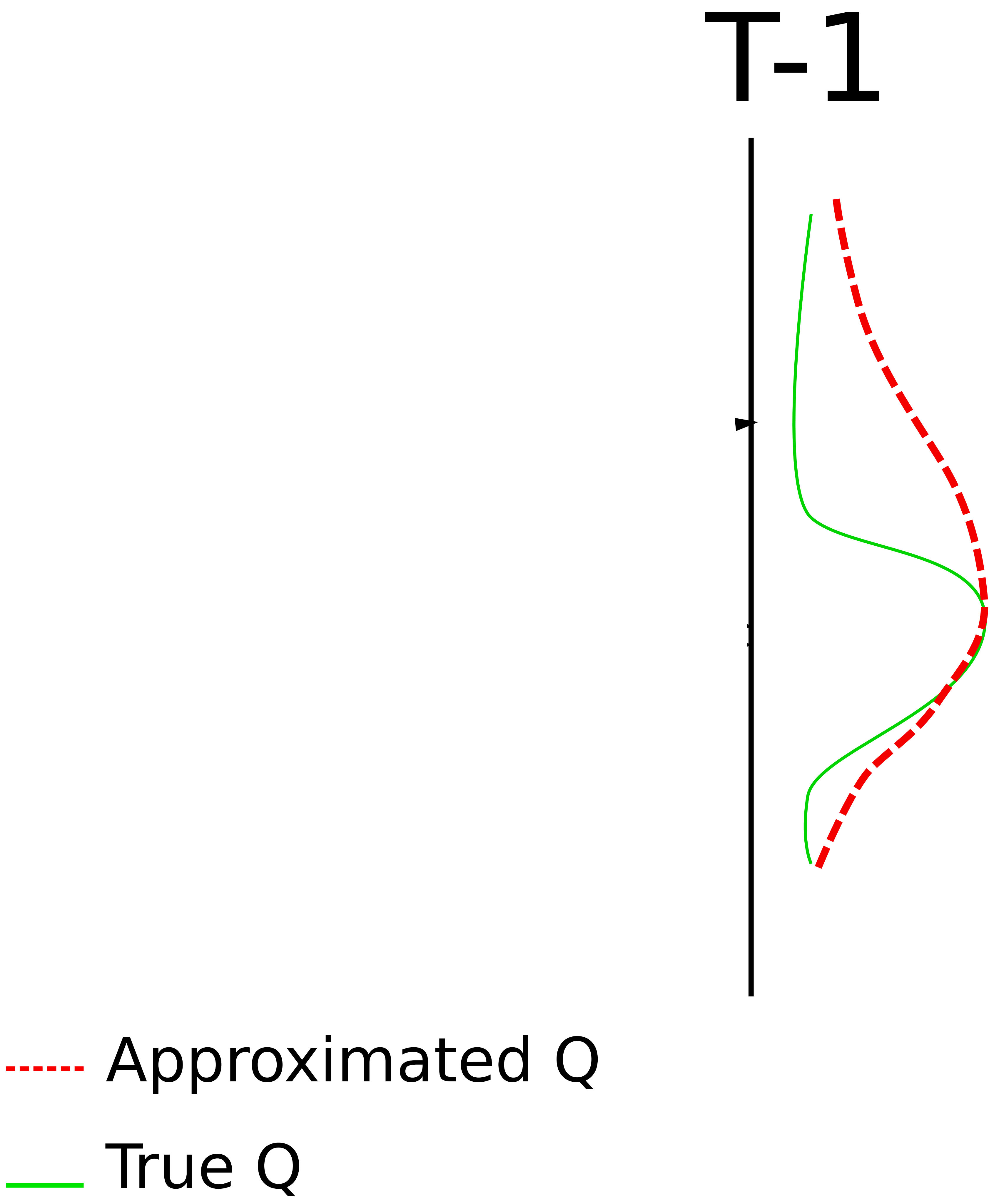
But this has the min() step!

$$\pi^+(s) = \arg\min_a Q^\pi(s,a)$$

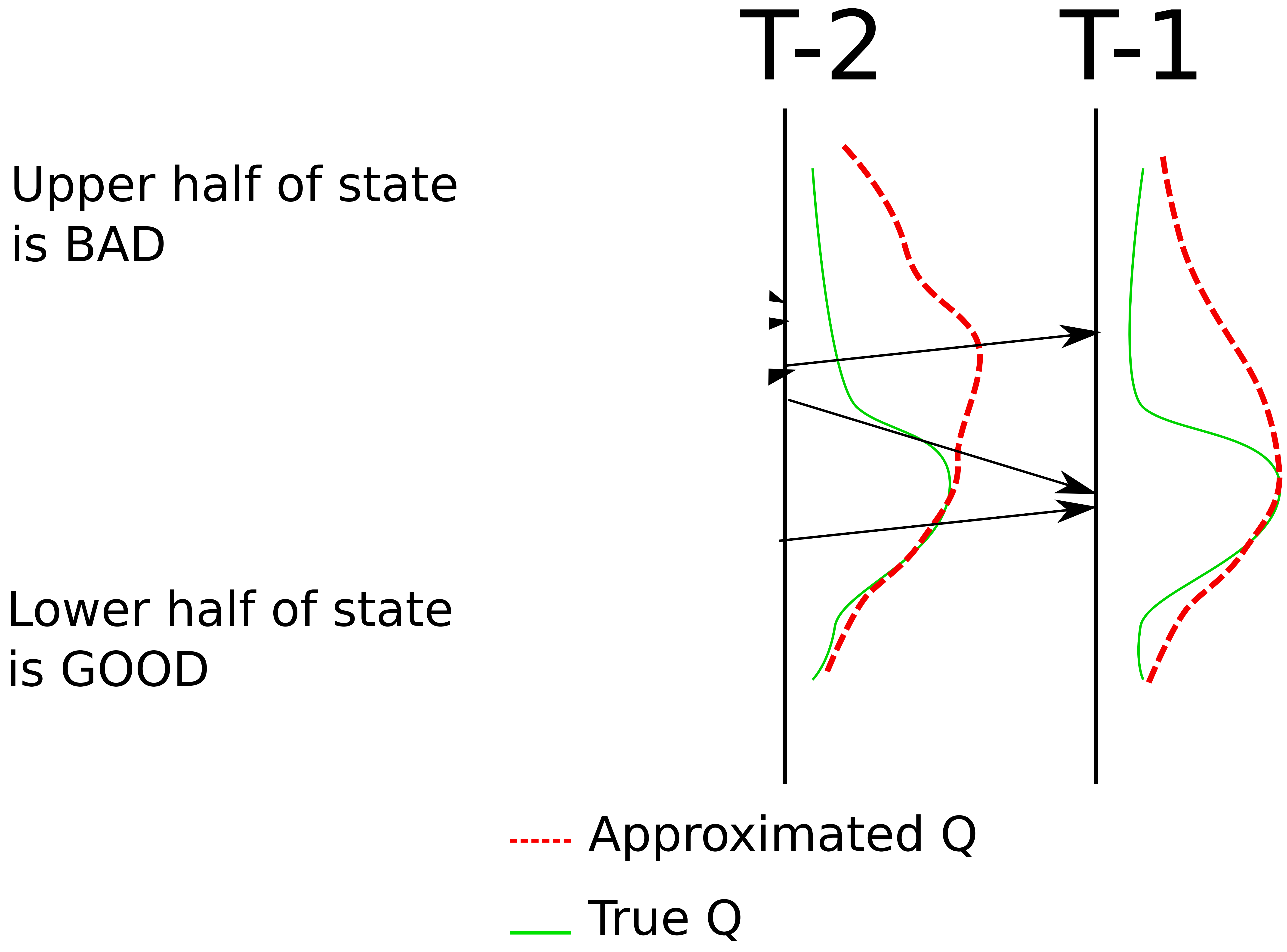
The problem of distribution shift

Upper half of state
is BAD

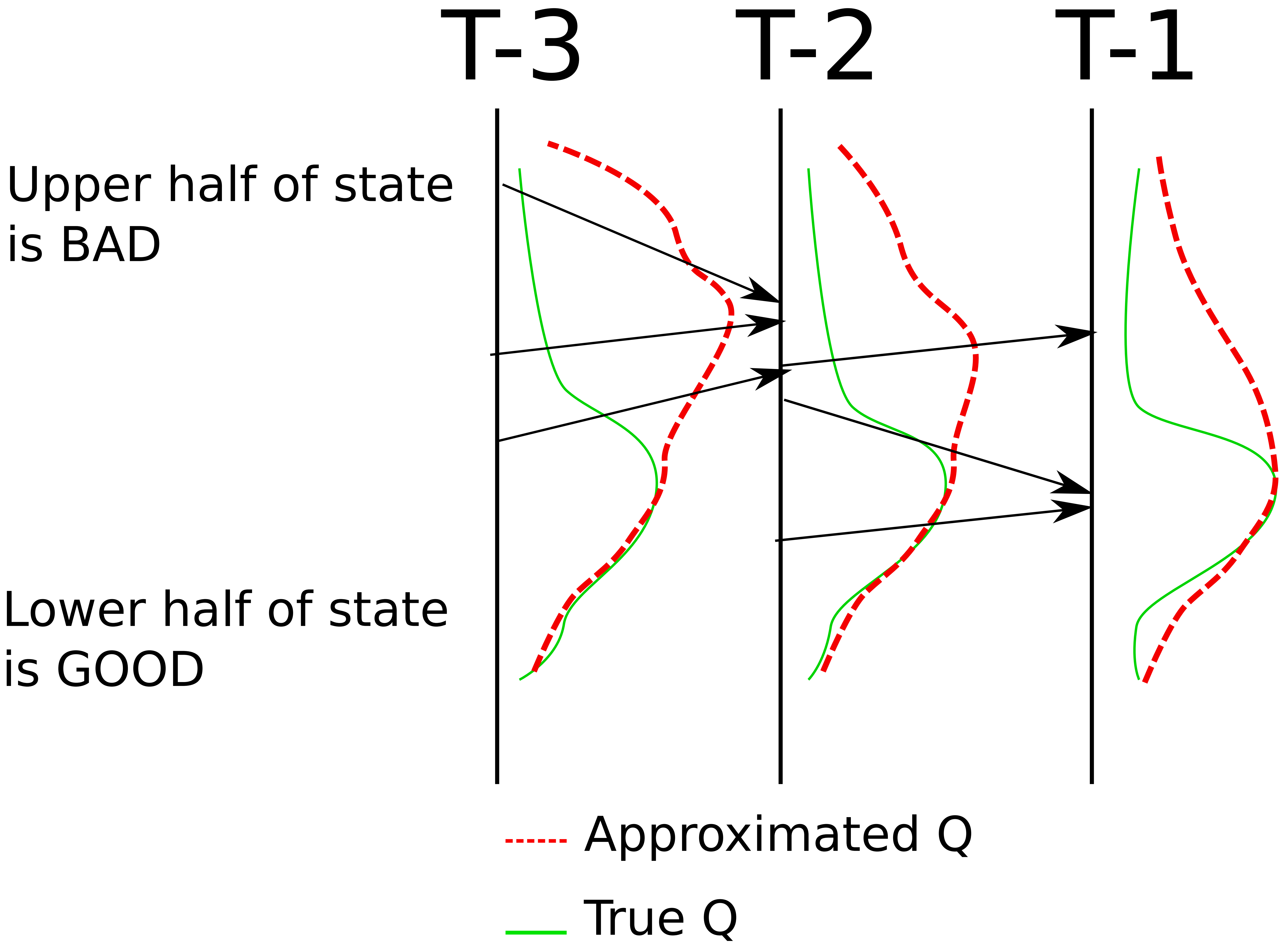
Lower half of state
is GOOD



The problem of distribution shift



The problem of distribution shift

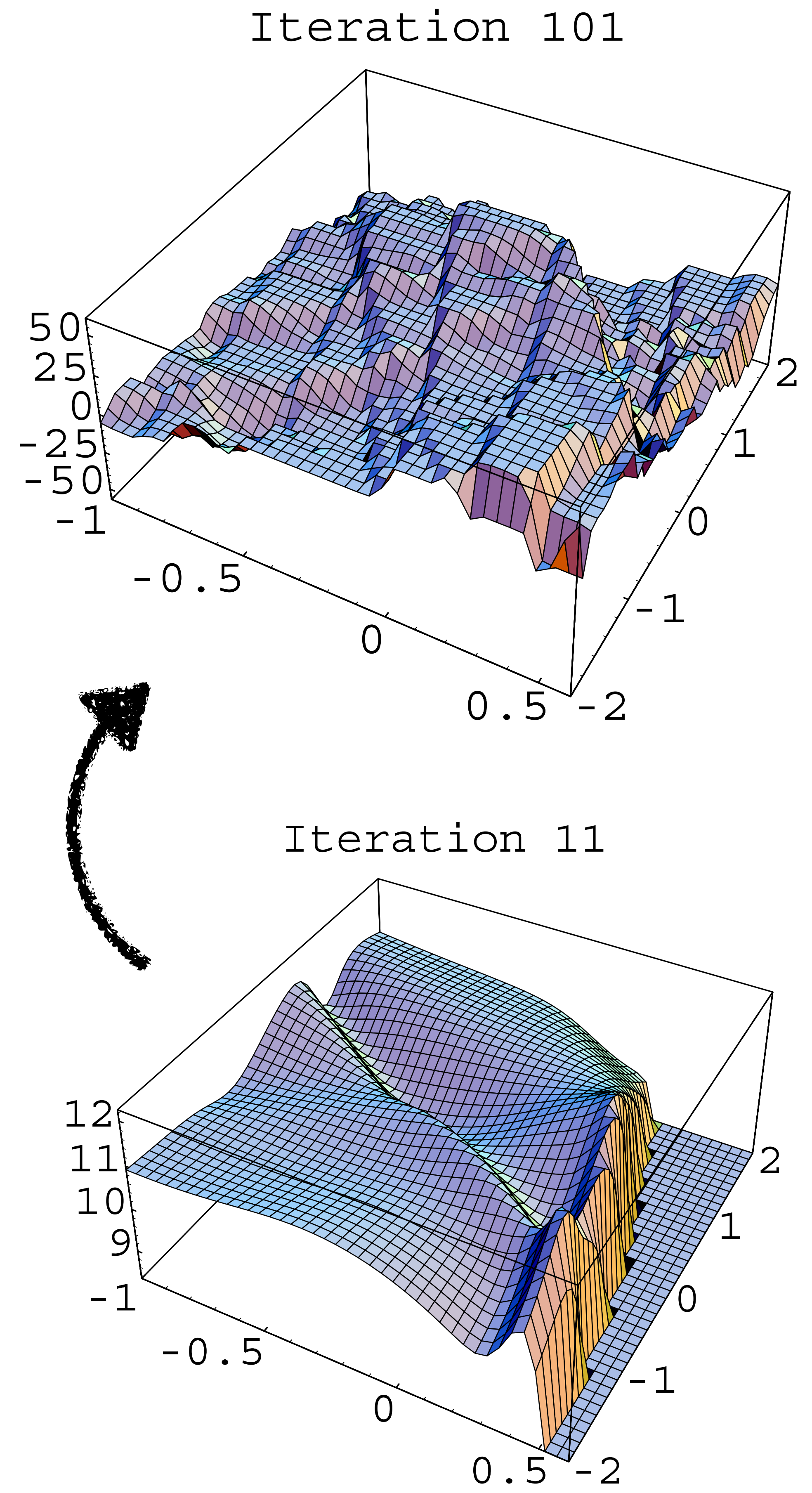



Boostrapping
Distribution Shift

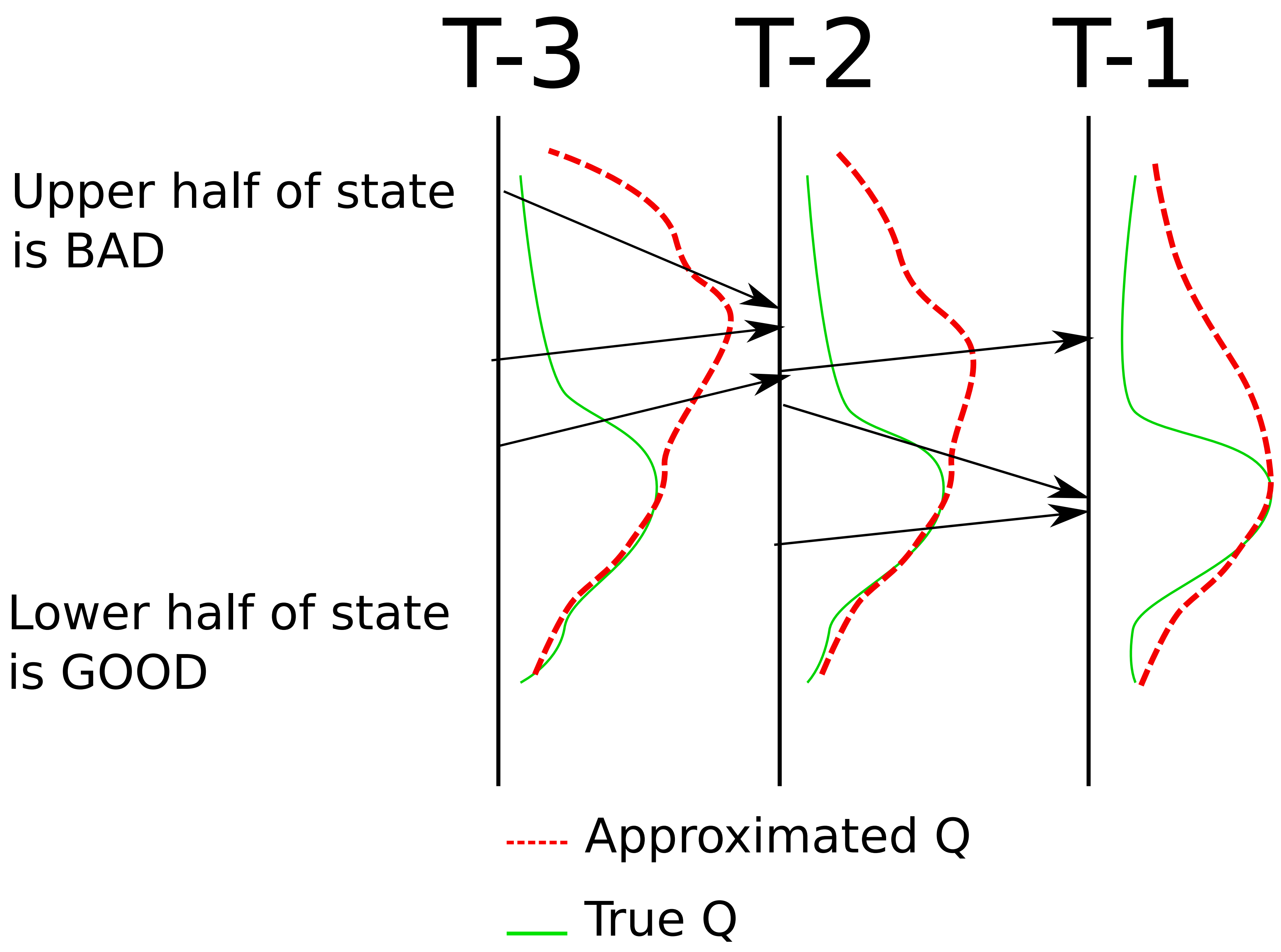
Two sides of the same coin

Bootstrapping

max

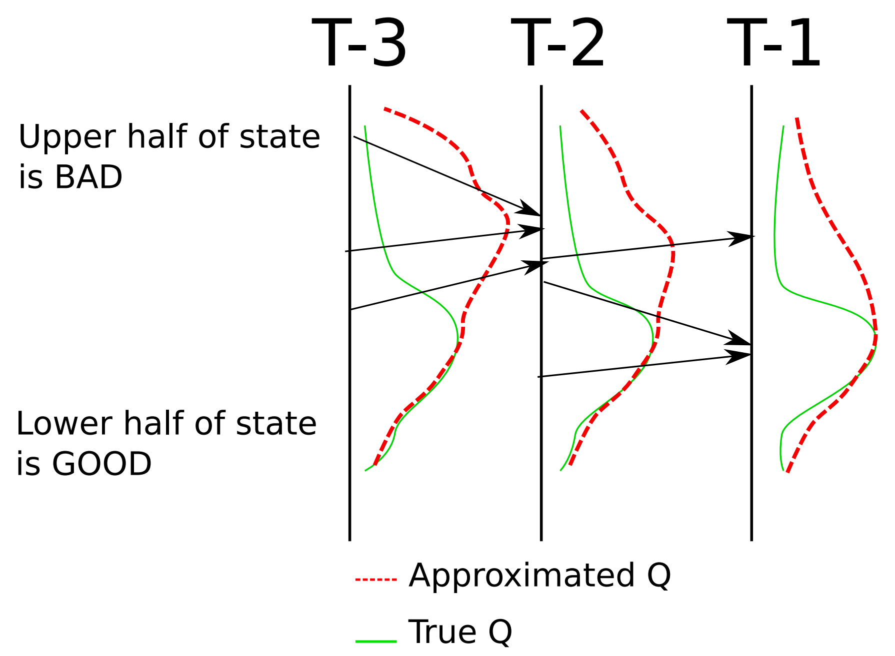


Distribution shift

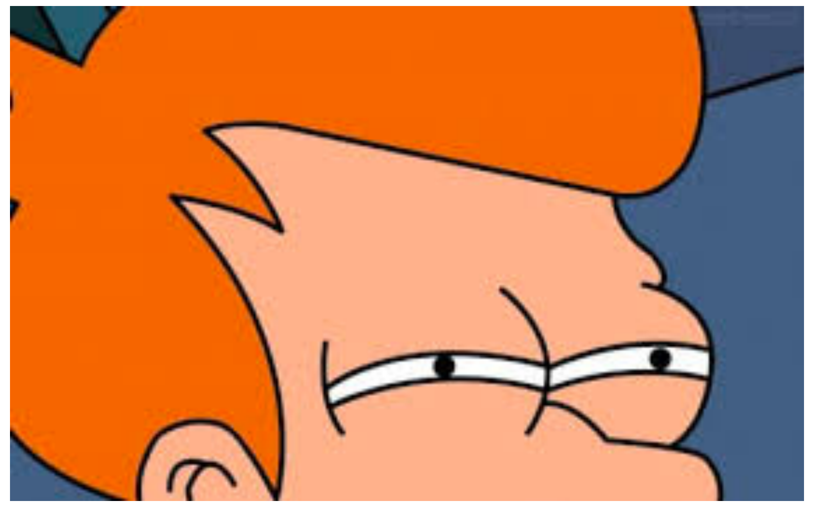


Ideas for fixing this?

Remedies



Bootstrapping



When doing min(),
don't trust value estimate

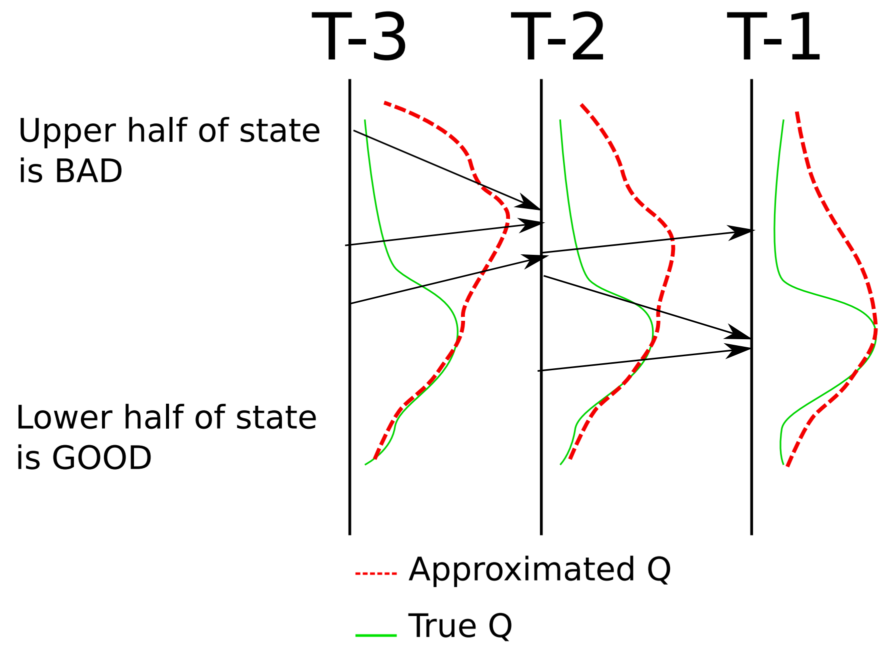
Execute policy
and trust actual returns

Distribution shift

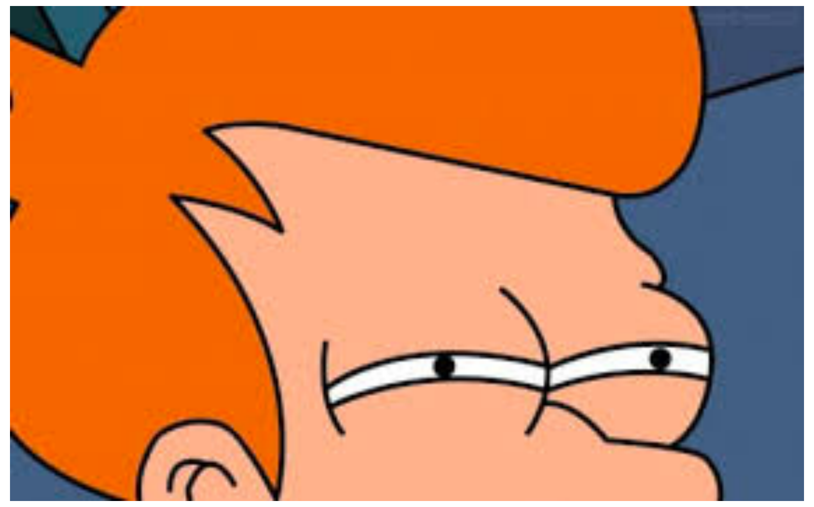
Minimize the
distribution shift

Be conservative,
change policy slowly

Remedies



Bootstrapping



When doing min(),
don't trust value estimate

Execute policy
and trust actual returns

Distribution shift

Minimize the
distribution shift

Be conservative,
change policy slowly

Fixing bootstrapping

Policy Search via Dynamic Programming

Design considerations with PSDP

Where do we get baseline distributions from?

From a human expert or from hand-crafted policies!

What do we do if T is very very large?

Just use a stationary policy

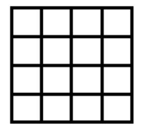
Performance Guarantees of PSDP

$$J(\pi_{PSDP}) - J(\pi^*) \le \sum_{t=1}^{T} \| \frac{d^t_{\pi^*}}{d^t_{\mu}} \|_\infty \epsilon_t$$

tl;dr

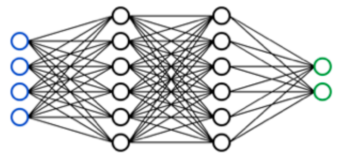
Approximate (Fitted) Value Iteration

Q-iteration



$Q(s,a) \leftarrow 0$
while *not converged* **do**
  for $s \in S, a \in A$
    $Q^{new}(s,a) = c(s,a) + \gamma \mathbb{E}_{s'} \min_{a'} Q(s',a')$
  $Q \leftarrow Q^{new}$
return Q

Fitted Q-iteration

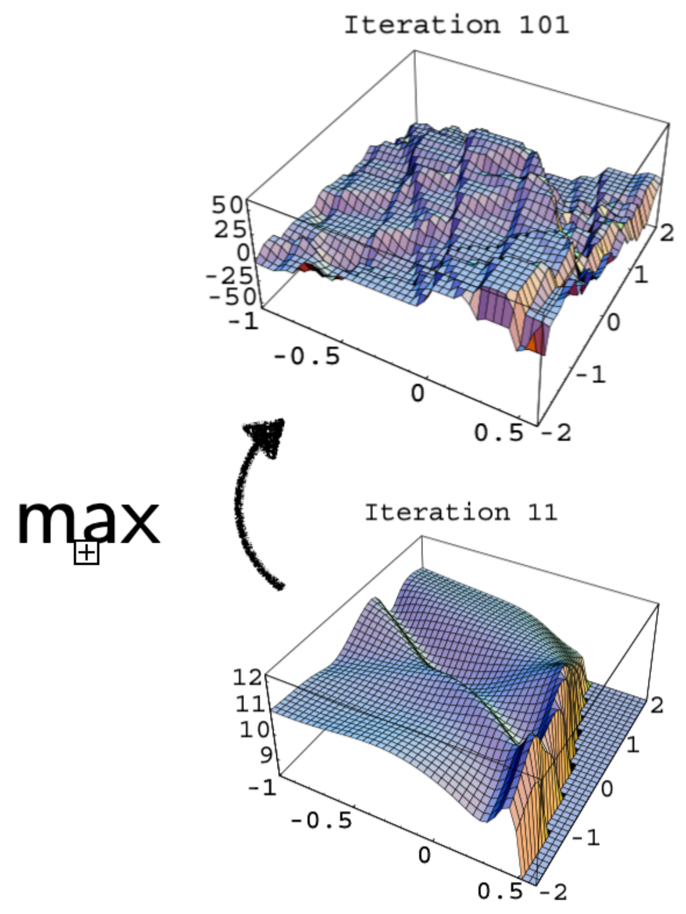


Given $\{s_i, a_i, c_i, s_i'\}_{i=1}^N$

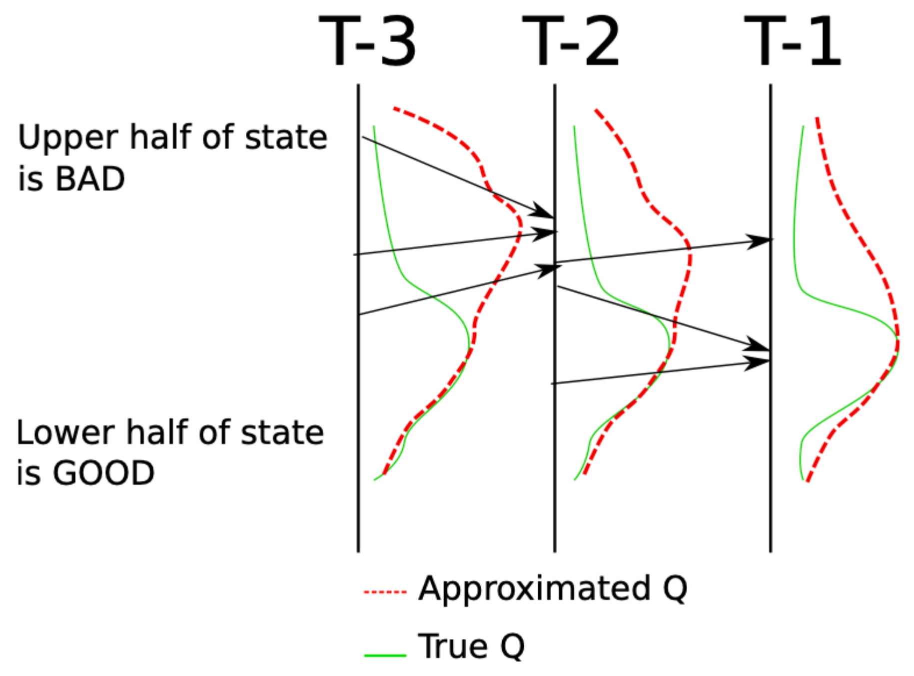
Init $Q_\theta(s,a) \leftarrow 0$
while *not converged* **do**
  $D \leftarrow \varnothing$
  for $i \in 1,\ldots,n$
    input $\leftarrow \{s_i, a_i\}$
    target $\leftarrow c_i + \gamma \min_a Q_\theta(s_i', a')$
    $D \leftarrow D \cup \{$input, output$\}$
  $Q_\theta \leftarrow$ **Train**(D)
return Q_θ

1

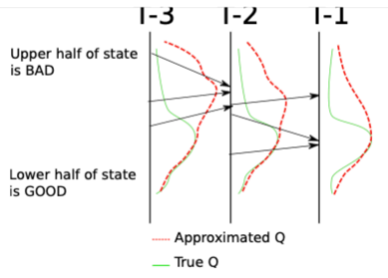
Bootstrapping



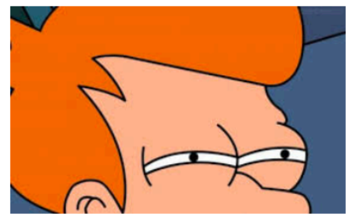
Distribution shift



Remedies



Bootstrapping



When doing min(),
don't trust value estimate

Execute policy
and trust actual returns

Distribution shift

Minimize the
distribution shift

Be conservative,
change policy slowly