# Approximate Value and Policy Iteration

Sanjiban Choudhury

Cornell Bowers CIS
Computer Science

# The story thus far ...

# The story thus far

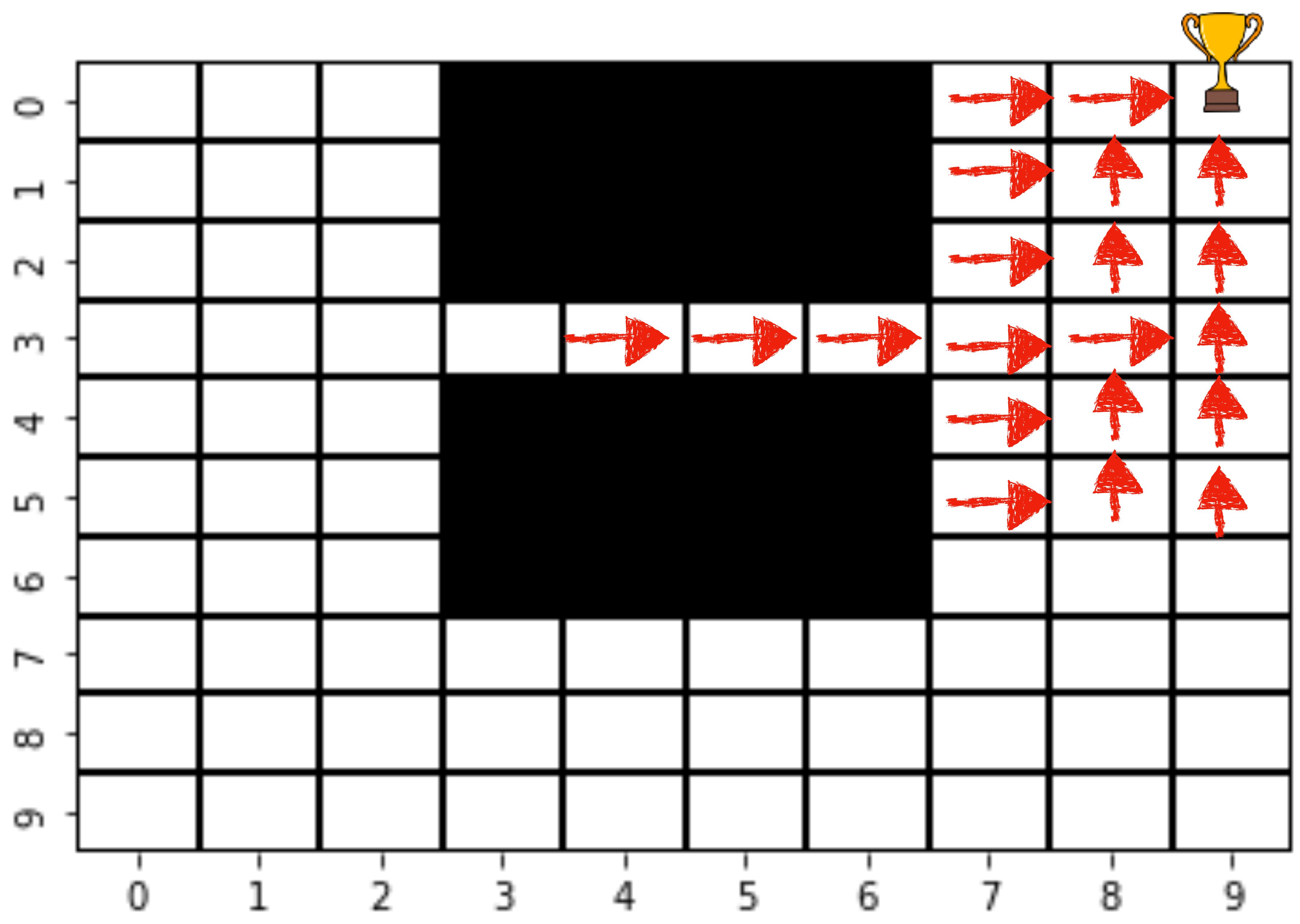
We know how to define an MDP

If the MDP is known (i.e. I know my costs and my transition)

We know how to solve a MDP

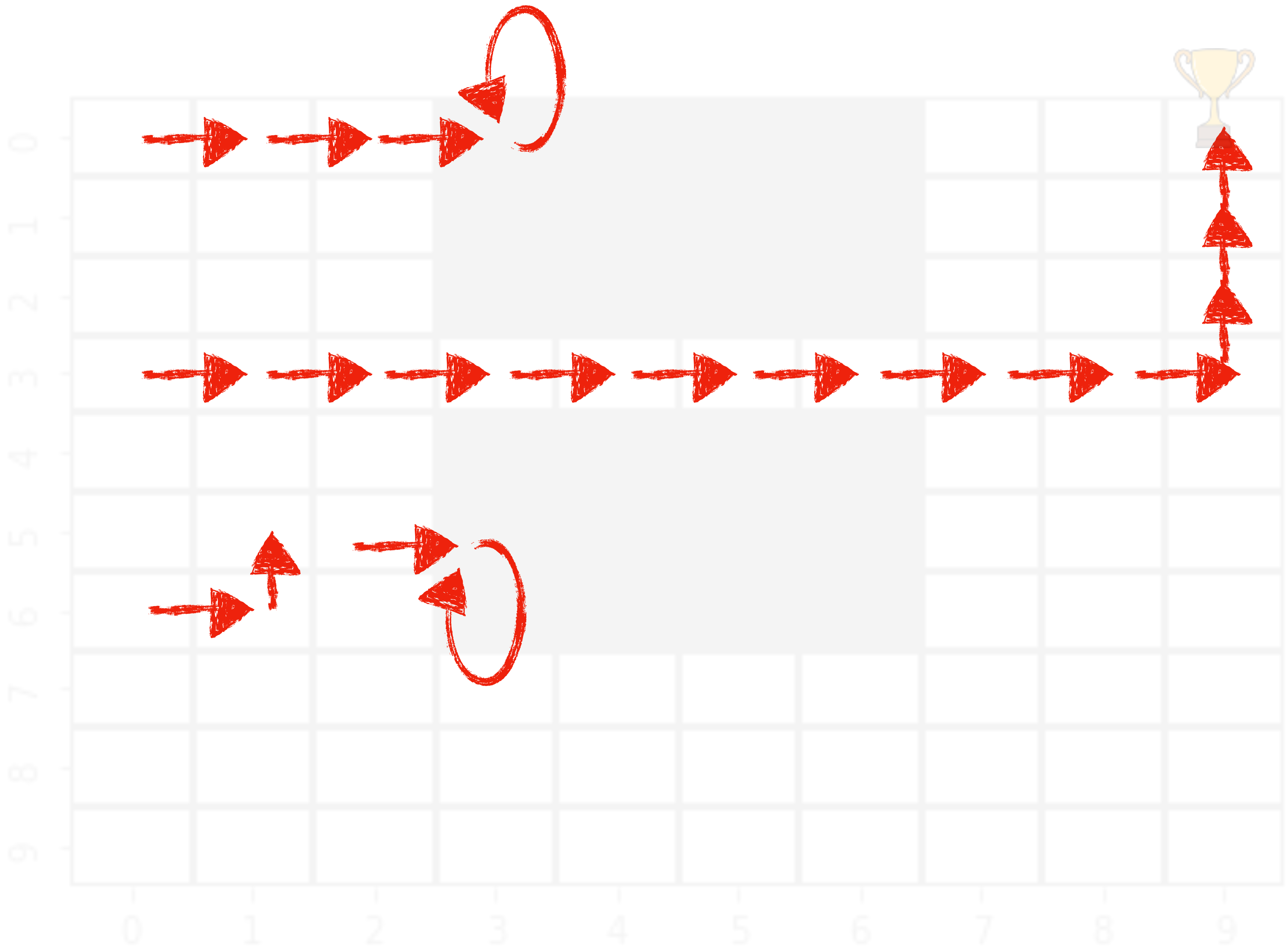
What happens if the MDP is unknown?

# Known MDP




*If I know the transition function, I could teleport to any state, try any action and know the next state*

# *Unknown* MDP



*I don't know the transition, I can only roll-out from start state, and see where I end up*

# Recall: How do we solve a known MDP?

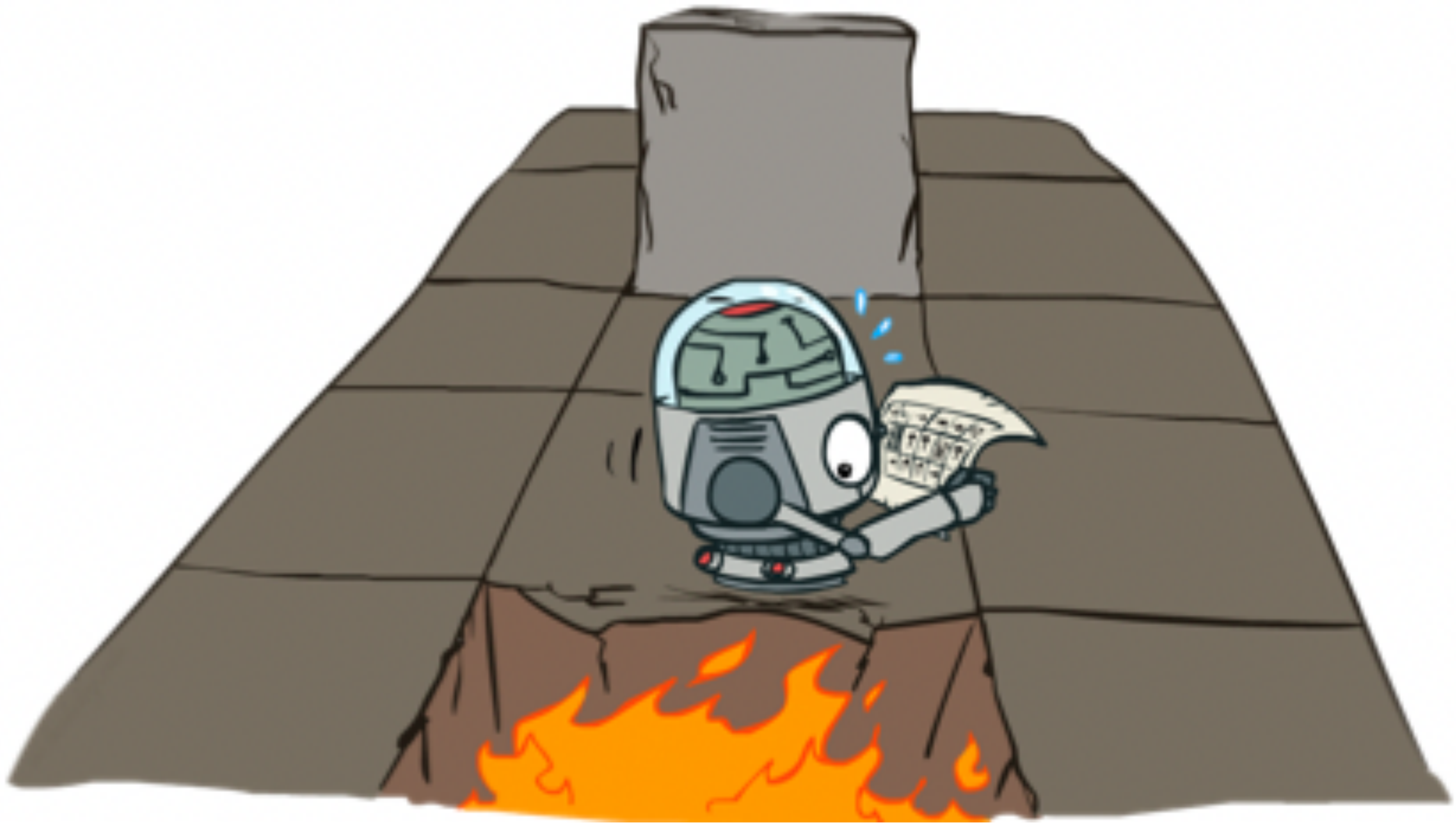
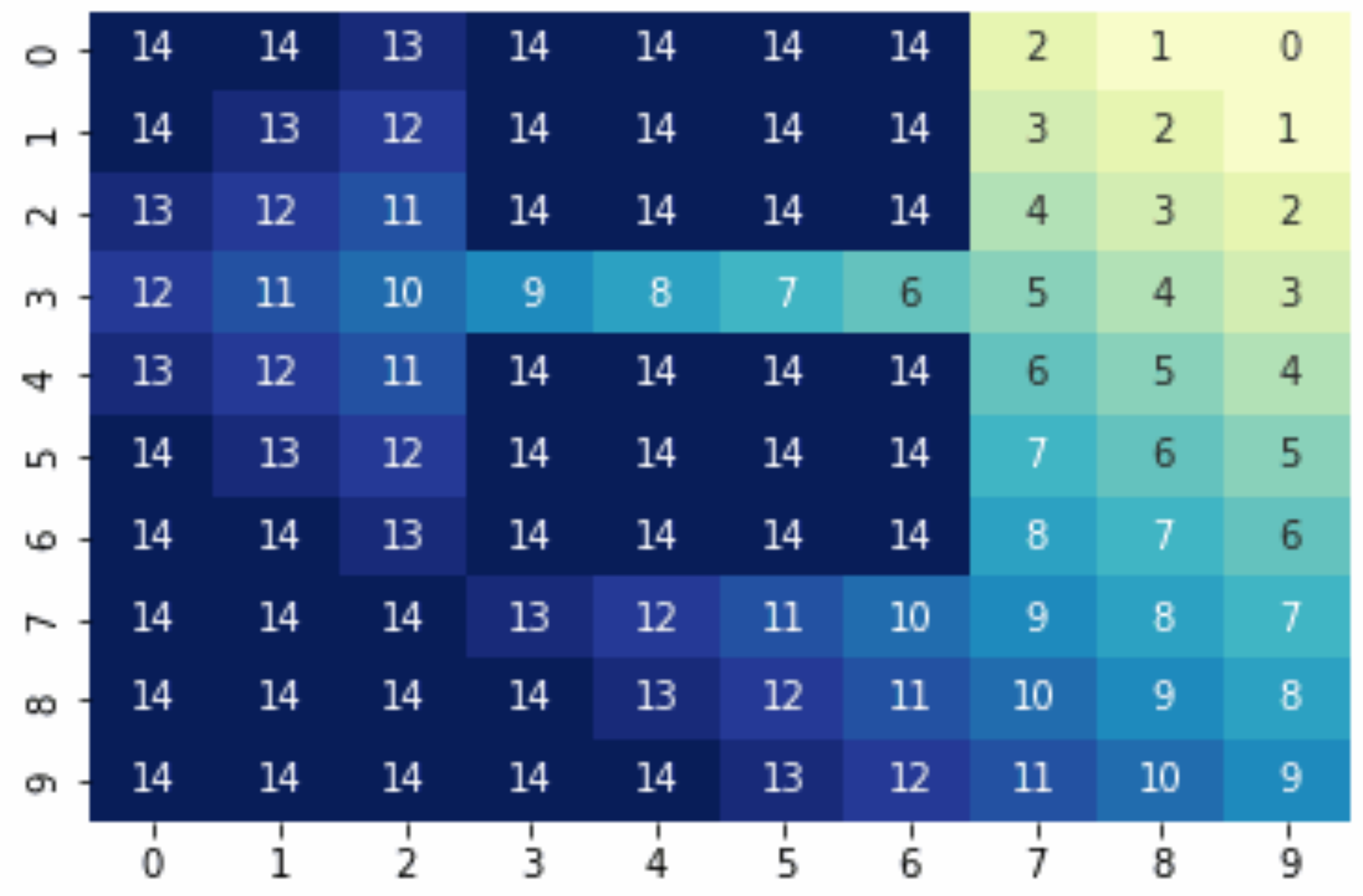


Image courtesy Dan Klein

# Value Iteration




Initialize value function at last time-step

$$V^*(s, T-1) = \min_a c(s,a) \quad \forall s$$

for $t = T-2, \ldots, 0$

Compute value function at time-step t

$$V^*(s,t) = \min_a \left[ c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right] \forall s$$

# Q-Value Iteration

Initialize value function at last time-step

$$Q^*(s, a, T-1) = c(s, a) \quad \forall (s, a)$$

for $t = T-2, \ldots, 0$

Compute value function at time-step t

$$Q^*(s, a, t) = c(s, a) + \gamma \sum_{s'} \mathcal{T}(s' | s, a) \min_{a'} Q^*(s', a', t+1) \quad \forall s, a$$

# Q-Value Iteration (Infinite horizon)

Initialize value function at last time-step

$$Q^*(s,a) = c(s,a) \quad \forall (s,a)$$

While not converged

Update value function

$$Q^*(s,a) = c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' | s,a) \min_{a'} Q^*(s',a') \quad \forall (s,a)$$

# Two Problems

Initialize value function at last time-step

$$Q^*(s,a) = c(s,a) \quad \forall (s,a)$$

While not converged

Update value function

$$Q^*(s,a) = c(s,a) + \gamma \sum_{s'} \mathcal{T}(s'|s,a) \min_{a'} Q^*(s',a') \quad \forall (s,a)$$

*Are these known?*

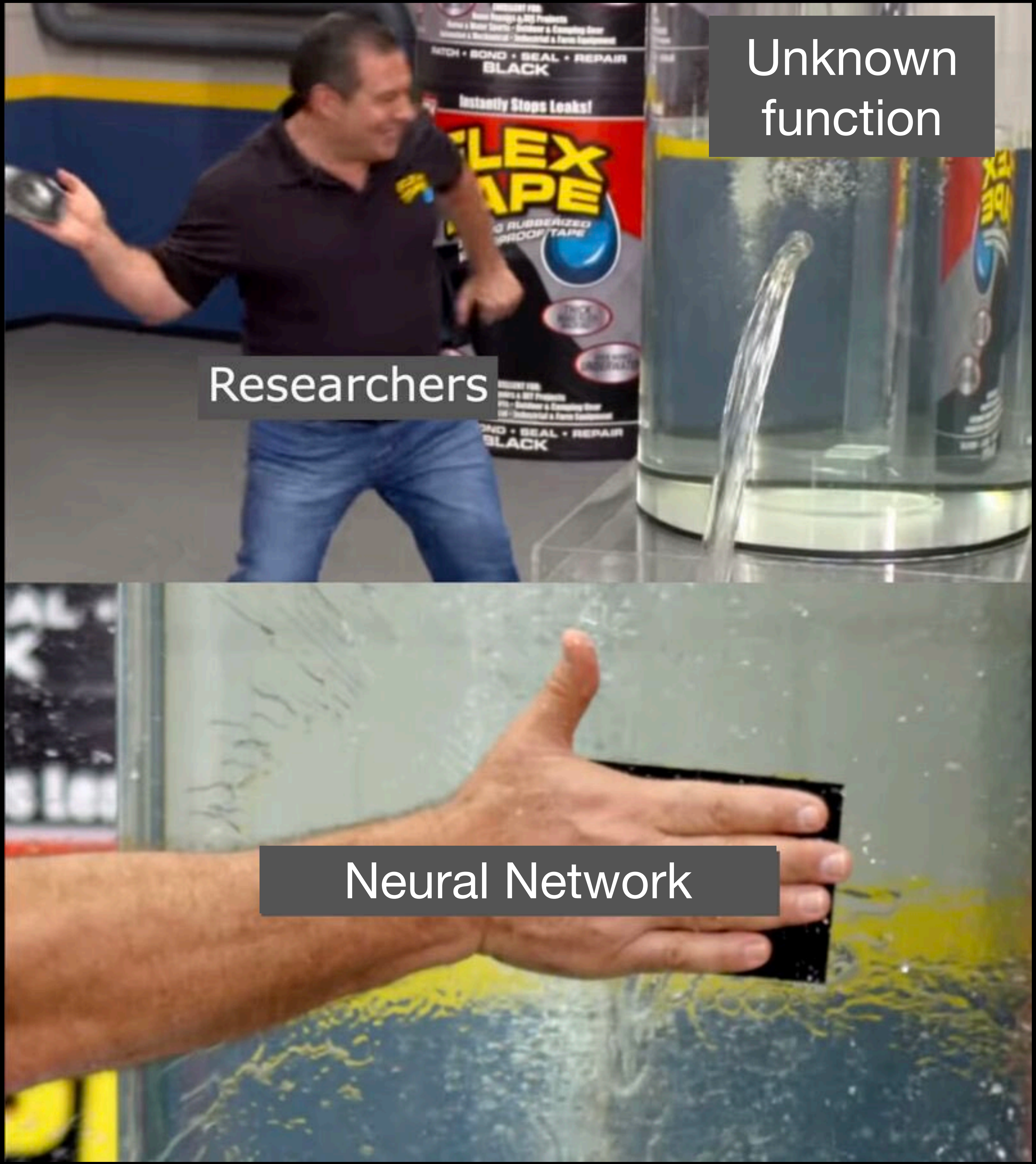
*Can I do this?*

*1) What happens when states are continuous?*

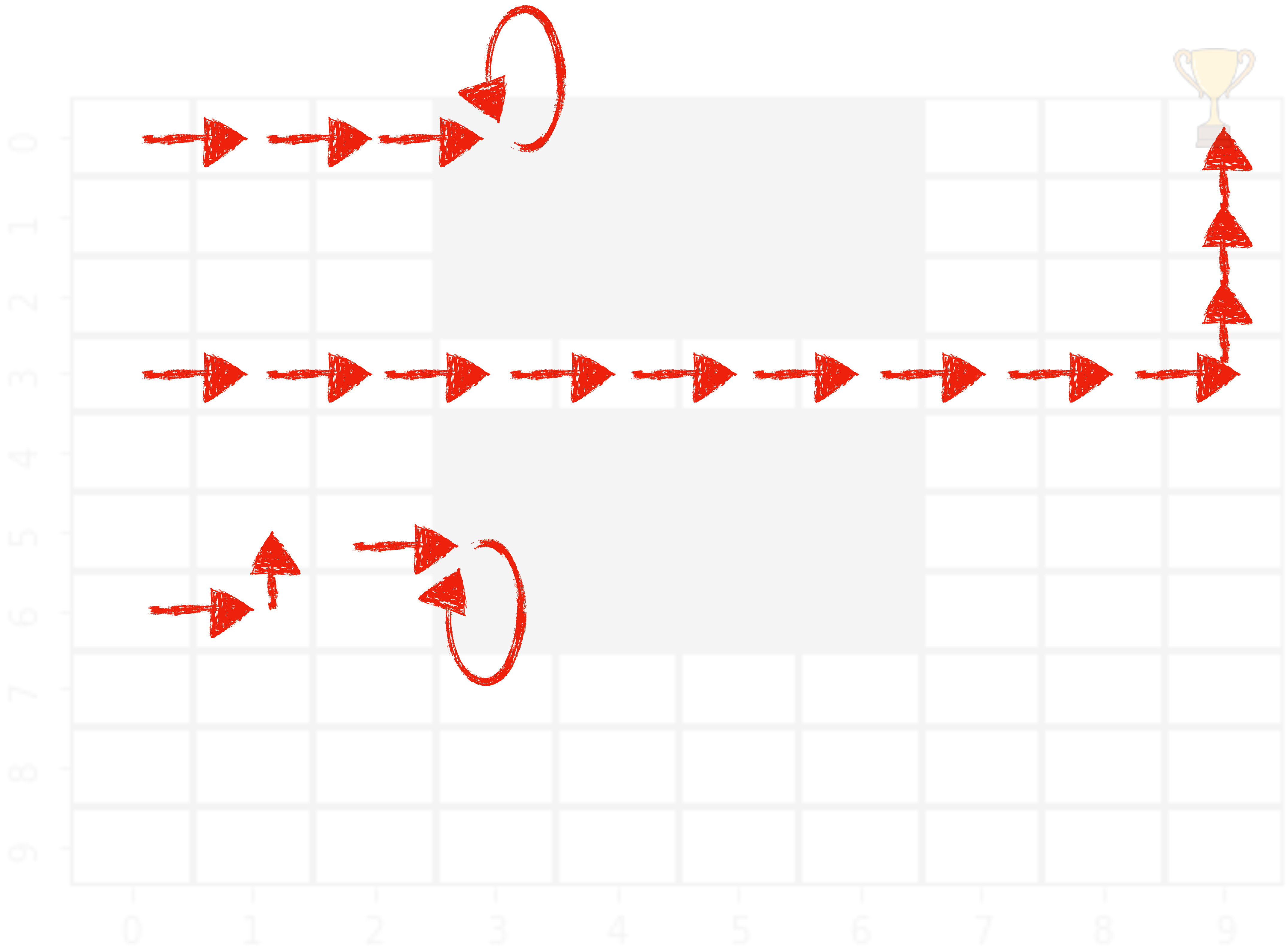
*2) What happens when I don't know the MDP?*

# Simple Idea

Can I collect roll-out data from the real world and just fit a Q function?


Unknown function
Researchers
Neural Network

# Step 1: First collect roll-out data

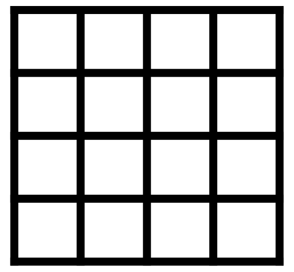


*Data is a tuple of state, action, cost, next state*

$$\mathscr{D} = \{(s_i, a_i, c_i, s_{i+1})\}_{i=1}^{n}$$

# Step 2: Fitted Q-Iteration

## Regular Q-iteration



$Q(s, a) \leftarrow c(s, a)$

**while** *not converged* **do**

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

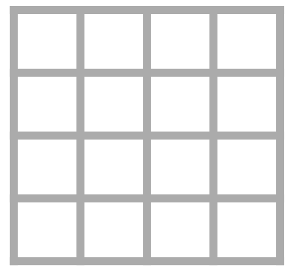
    $Q^{new}(s, a) = c(s, a) + \gamma \mathbb{E}_{s'} \min_{a'} Q(s', a')$

  $Q \leftarrow Q^{new}$

**return** $Q$

# Step 2: Fitted Q-Iteration

## Regular Q-iteration



$Q(s,a) \leftarrow c(s,a)$

**while** *not converged* **do**

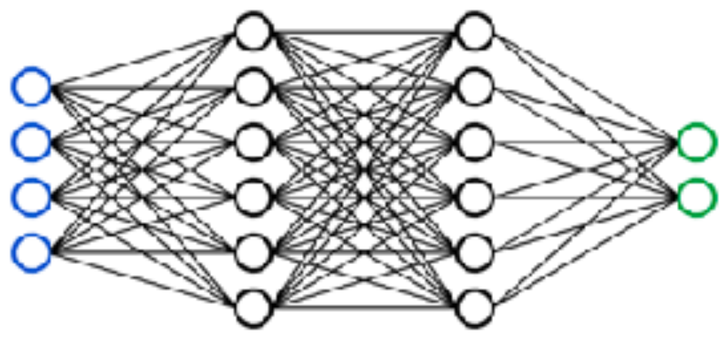
  **for** $s \in S, a \in A$

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

  $Q \leftarrow Q^{new}$

**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 $\leftarrow c_i + \gamma \min_{a} Q_\theta(s_i', a')$

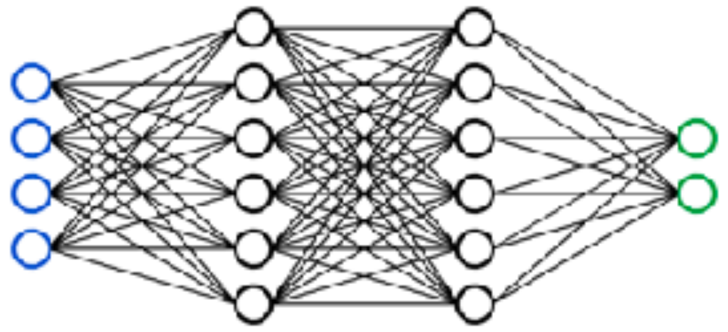
    $D \leftarrow D \cup \{$input, output$\}$

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

**return** $Q_\theta$

# Step 2: Fitted Q-Iteration

*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** $Q_\theta$

*Use old copy of Q to set target*

*Training is a regression problem*

$$\ell(\theta) = \sum_{i=1}^{n} (Q_\theta(s_i, a_i) - target)^2$$

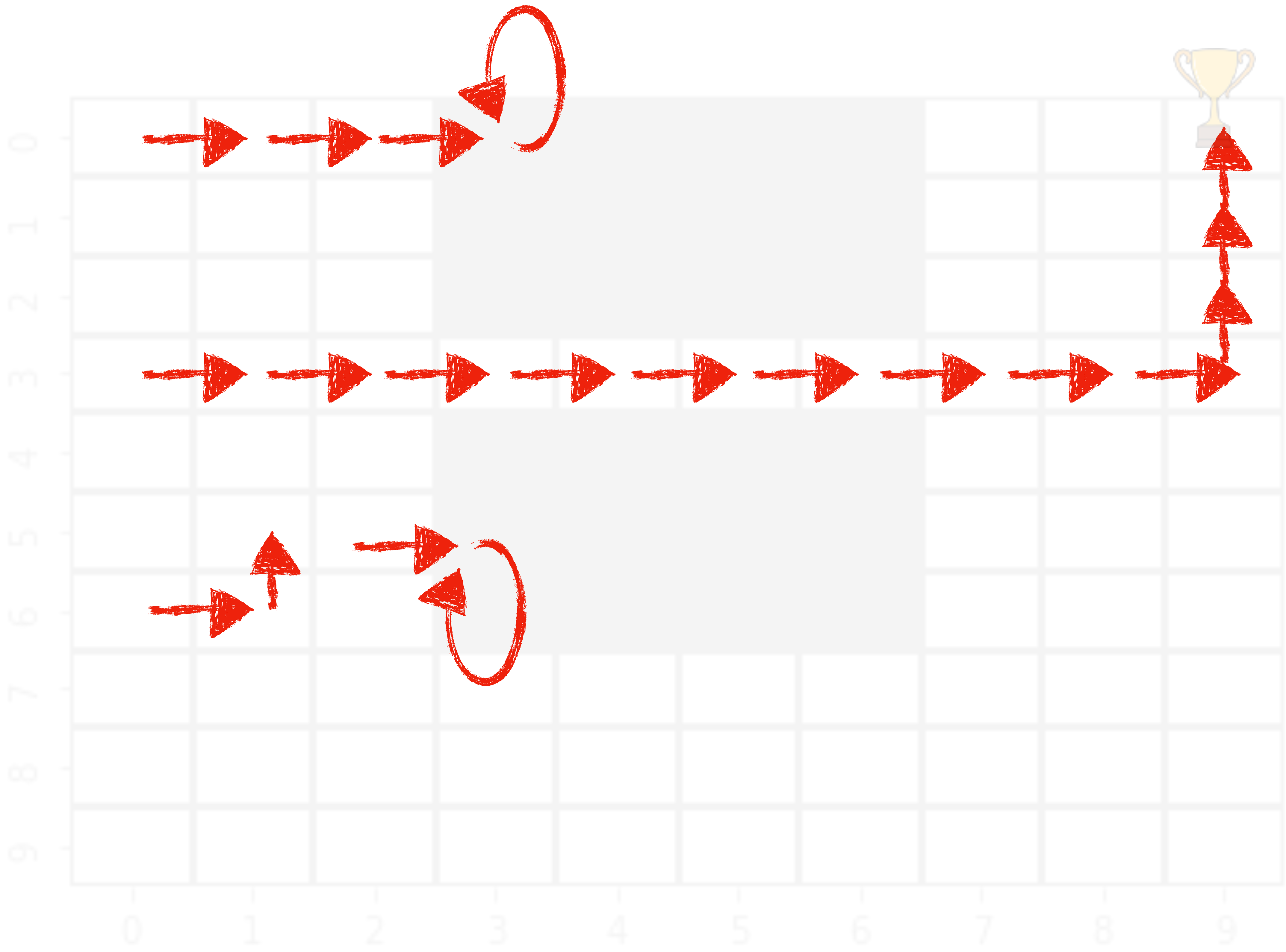
# Temporal Difference Error (TD Error)

Penalize violation of Bellman Equation

$$\ell(\theta) = \left( c(s,a) + \gamma \min_{a'} Q_{\theta_{old}}(s', a') - Q_{\theta}(s_t, a_t) \right)^2$$

$$\theta = \theta_{\text{old}} - \alpha \nabla_{\theta} l(\theta)$$

# What policy do I use to collect data?



Do I explore randomly? Do I use my learnt Q function?

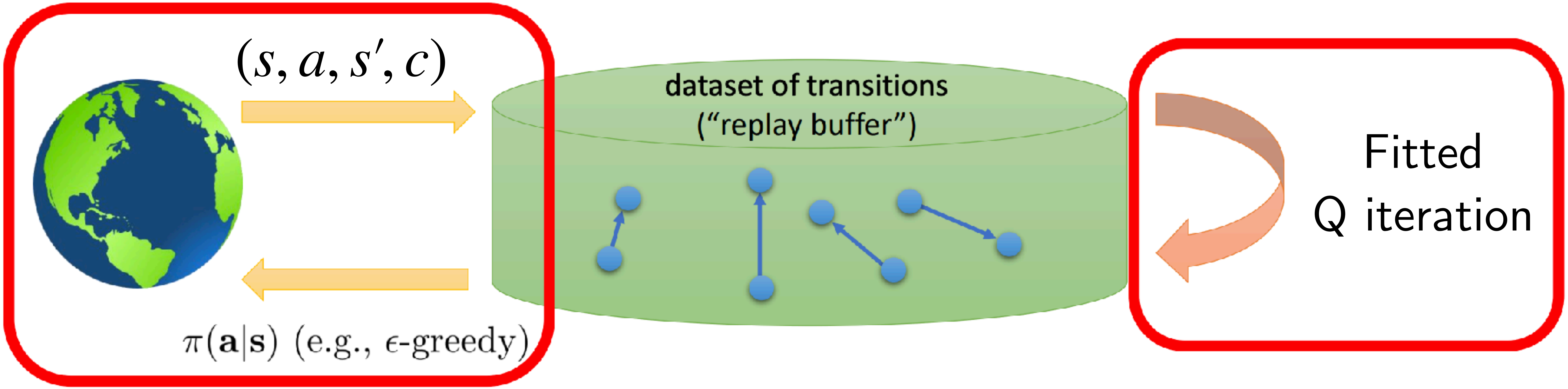
# What policy do I use to collect data?

When poll is active respond at **PollEv.com/sc2582**

Send **sc2582** to **22333**

Do I explore randomly? Do I use my learnt Q function?

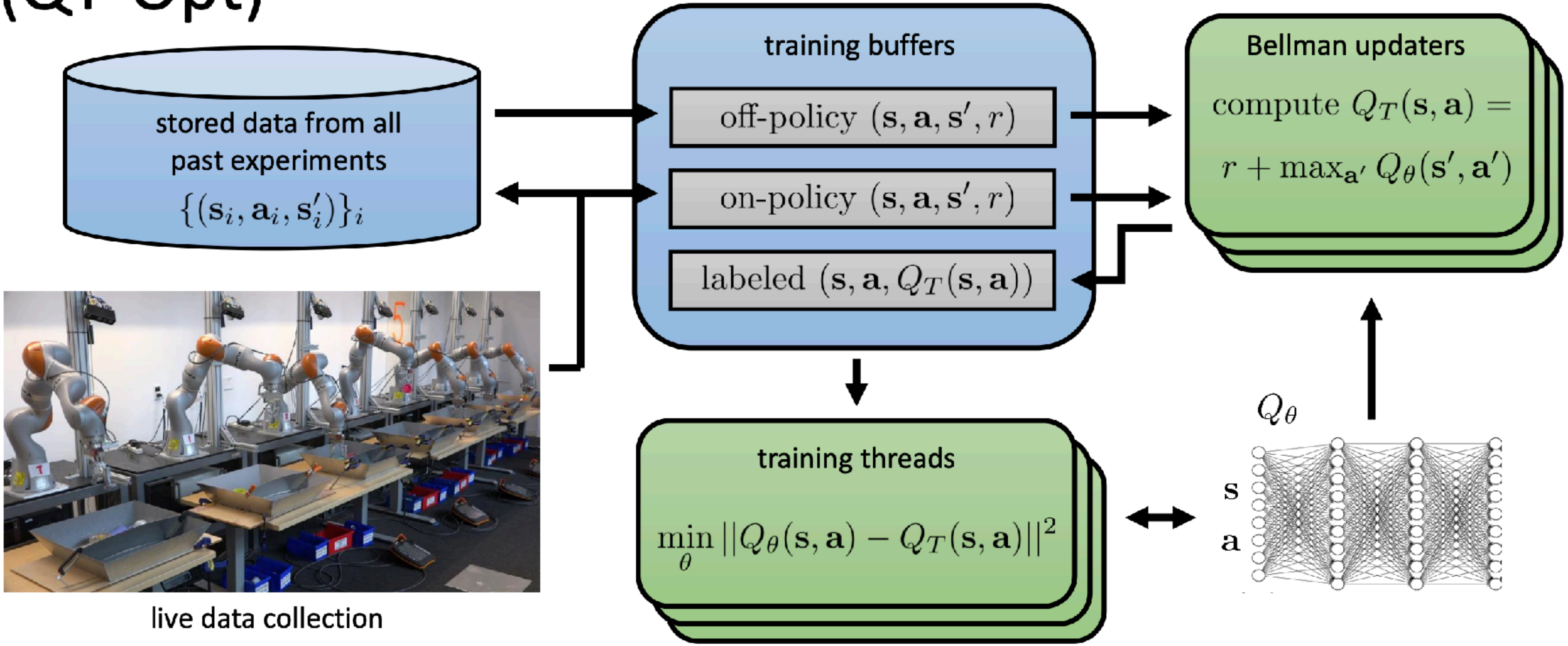
# Q-learning: Learning off-policy



# QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

Training time
Distributed RL
-
=
Reward: Grasp success determined by subtracting pre and post-drop images
State, Action, Reward
Learned weights
Inference time
State: 472x472 Image and gripper aperture
Camera
Robot
Critic Function
$Q(\text{State}, \text{Action})$
Action proposals
Q-Values
Action: Gripper displacement and aperture
Cross-Entropy Method
$\arg\max_{\text{Action}} Q(\text{State}, \text{Action})$

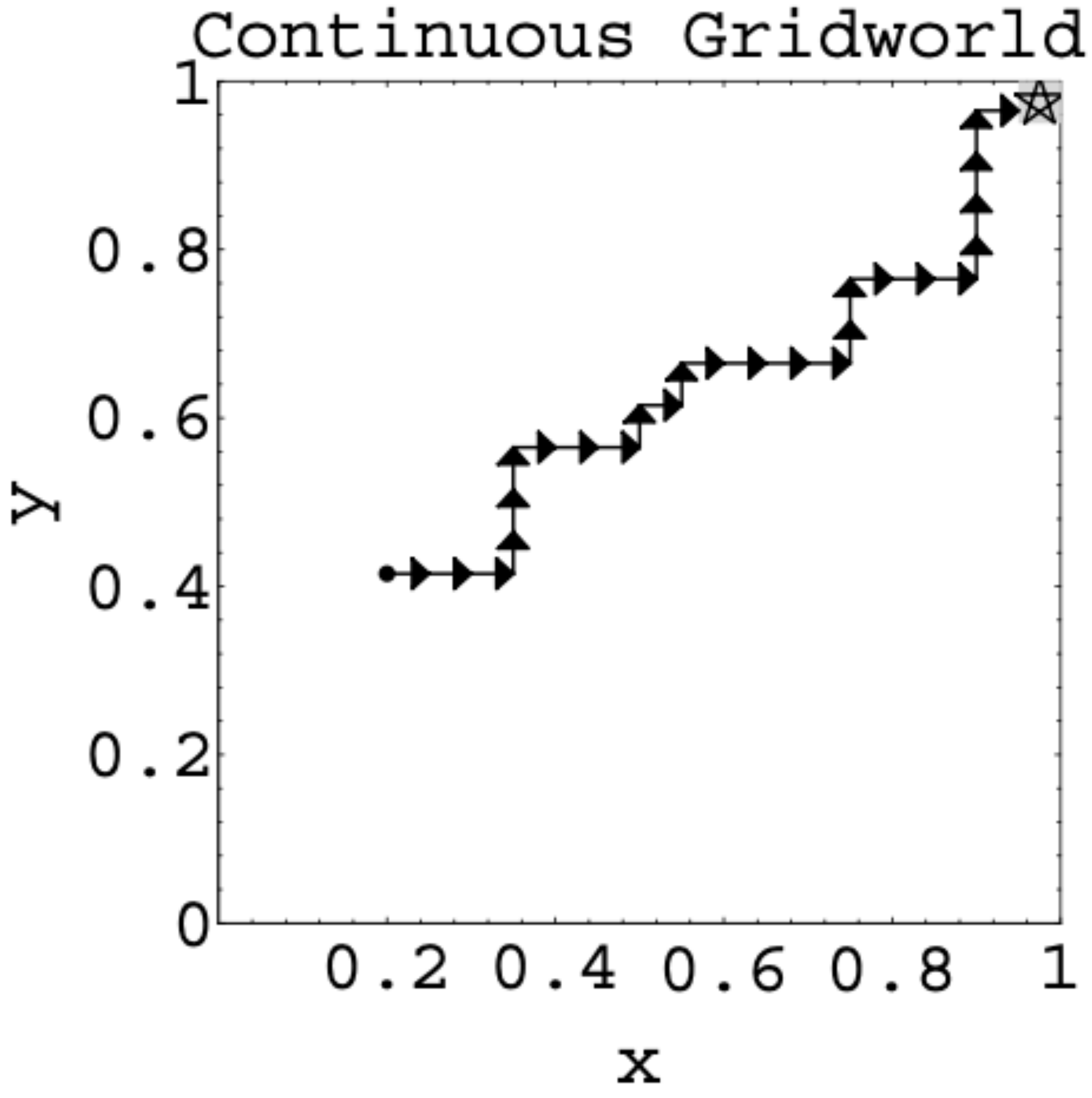
# Large-scale Q-learning with continuous actions (QT-Opt)

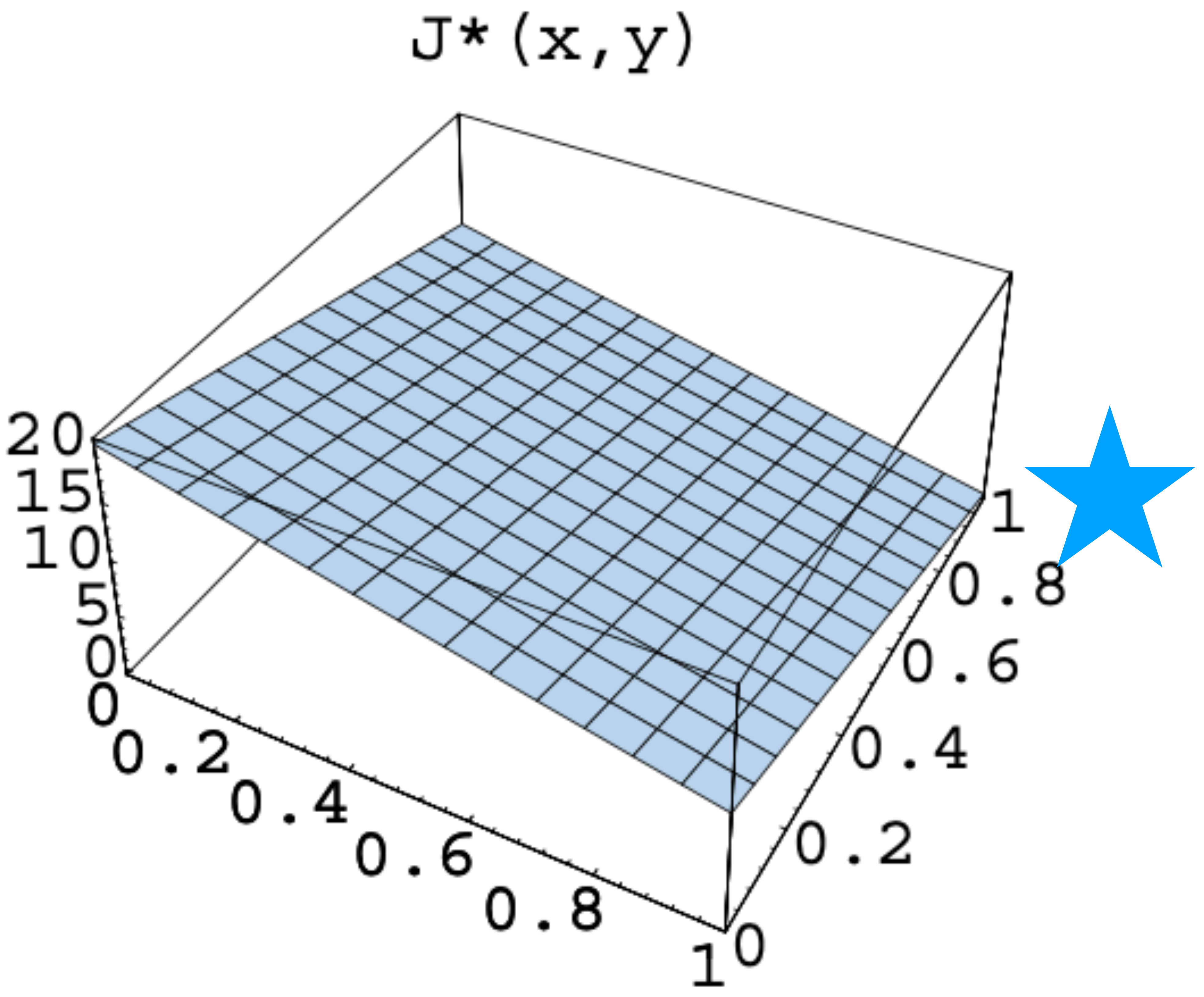



Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

So does approximate value iteration work?

# A simple example: Gridworld



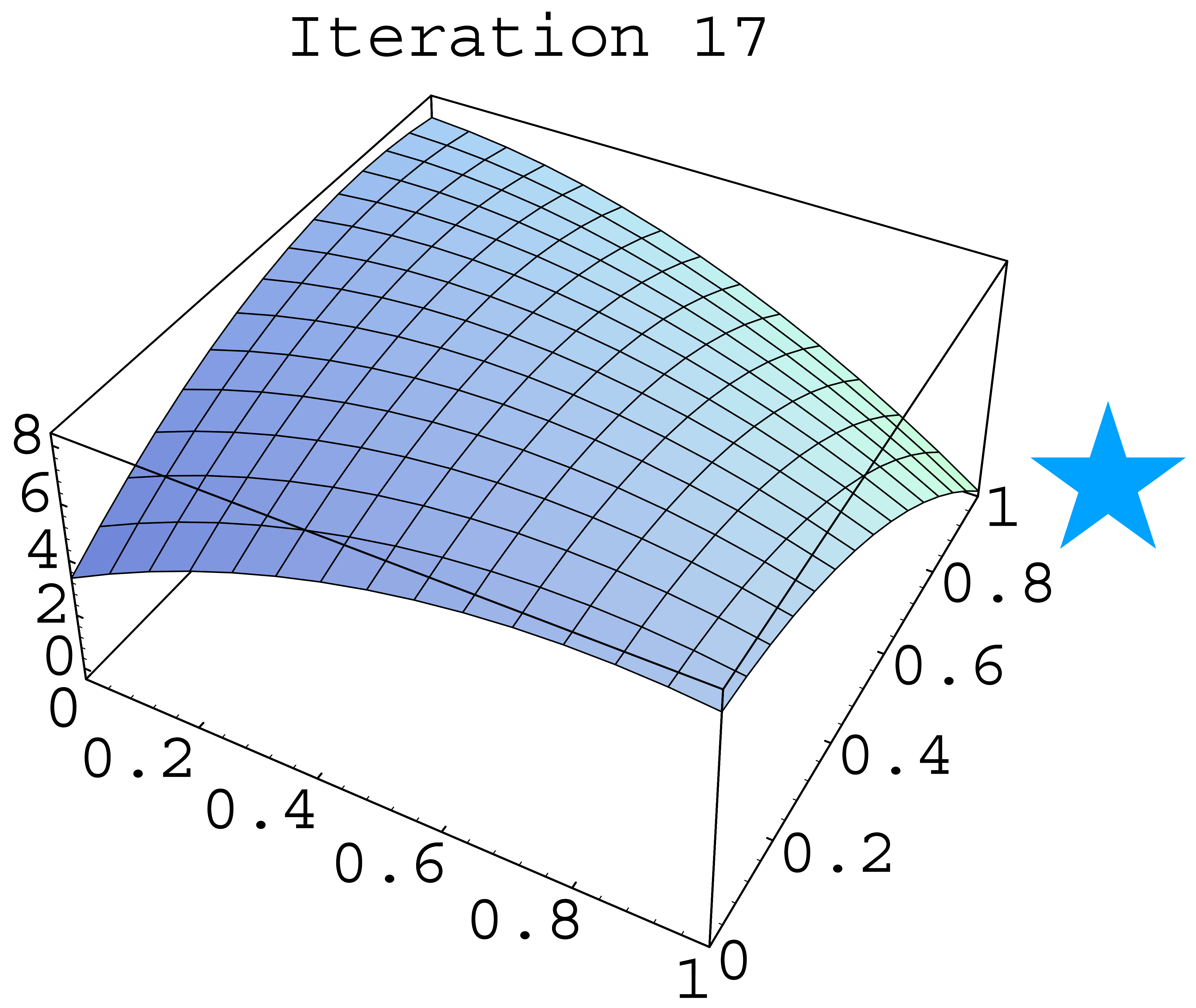




Optimal path

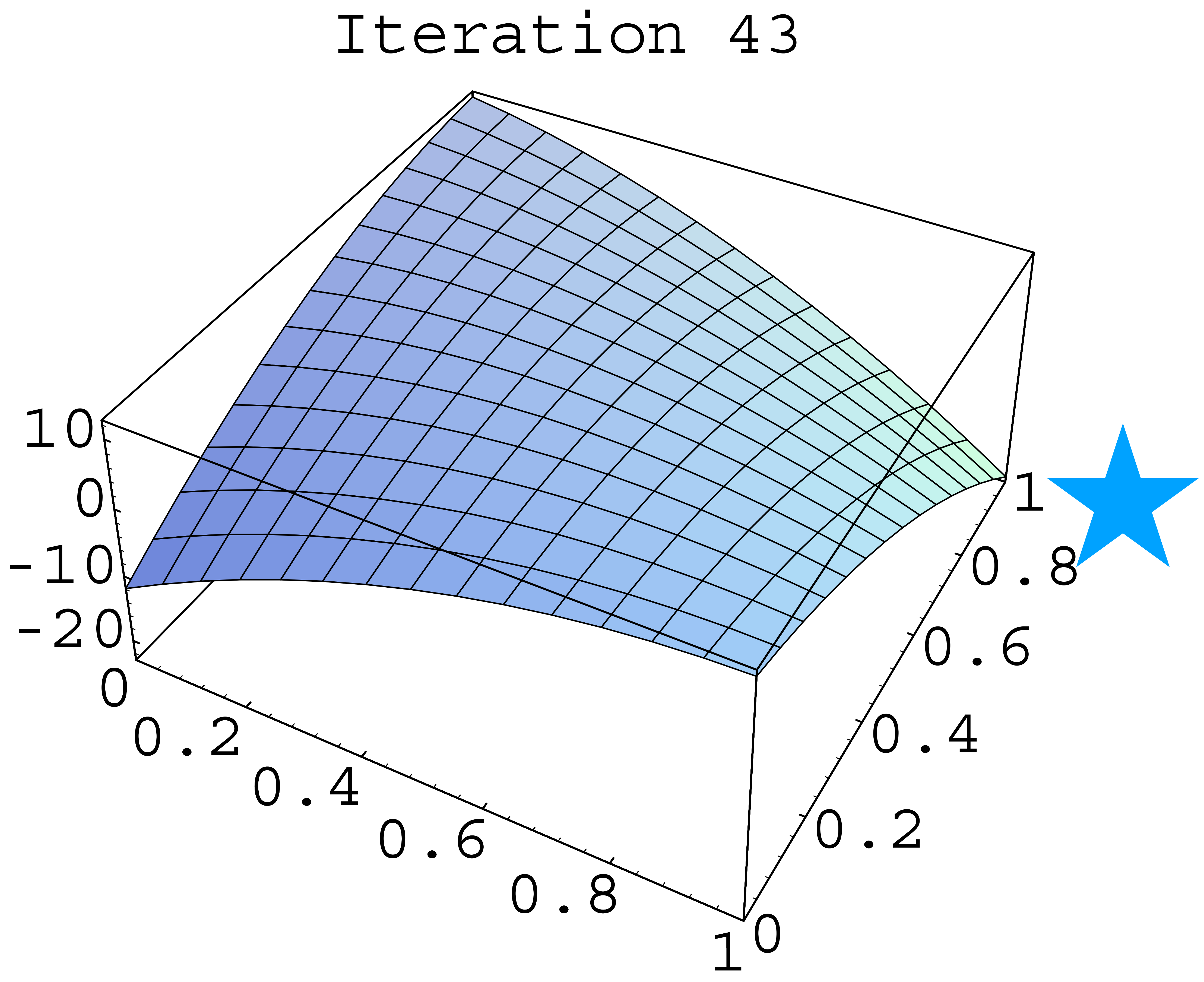
True value function

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

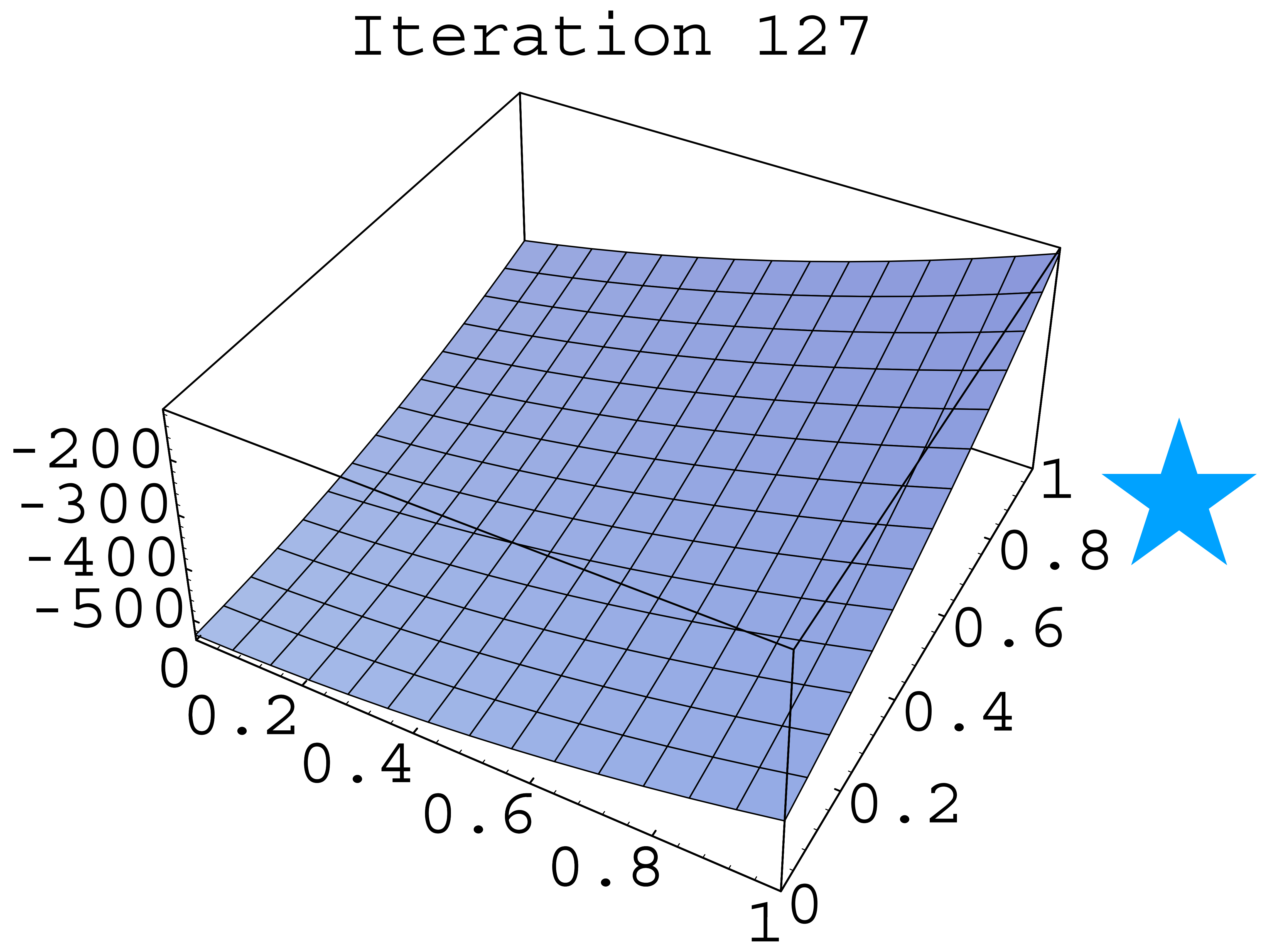
# What happens when we run value iteration with a *quadratic*?



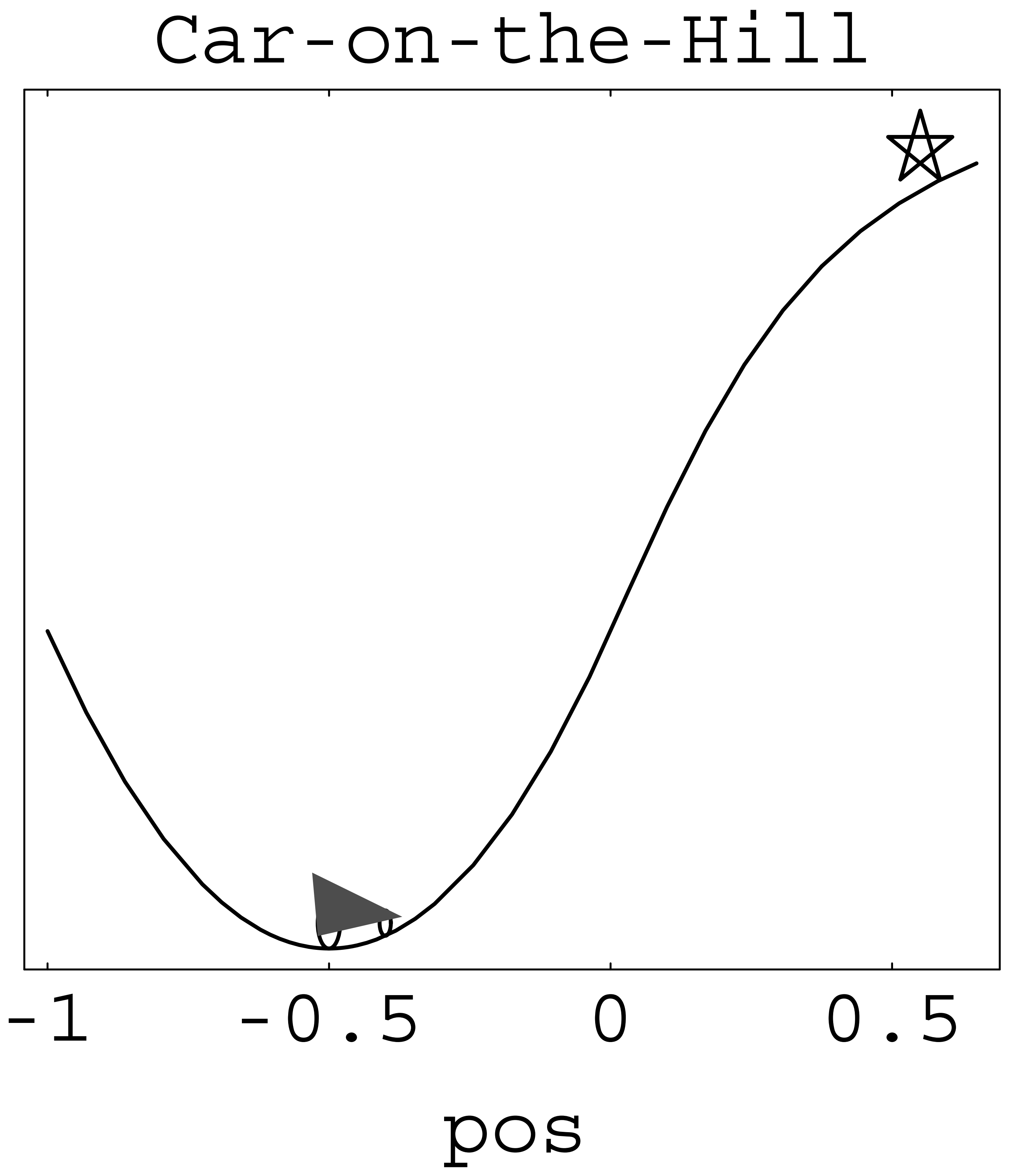
# What happens when we run value iteration with a *quadratic*?

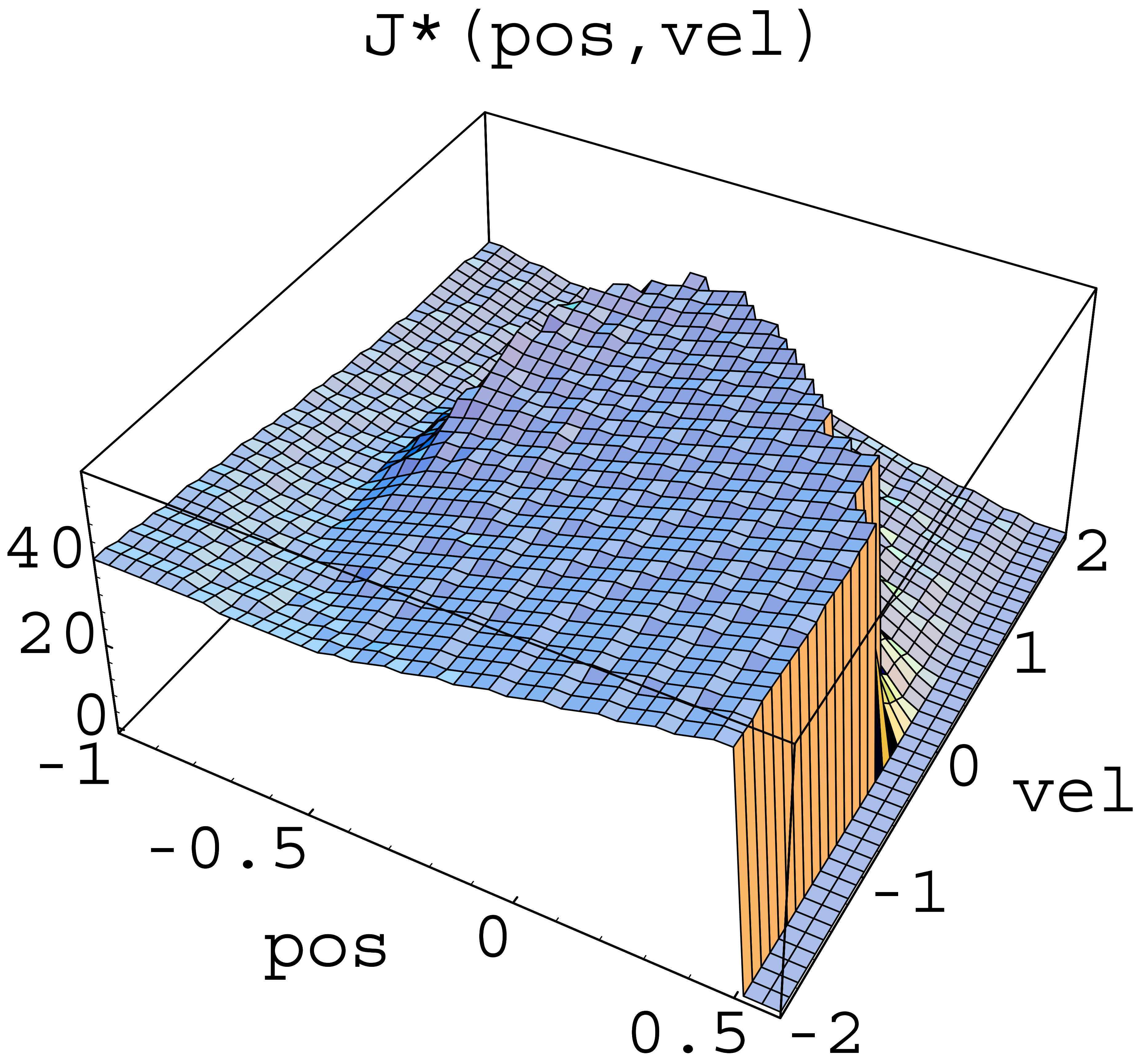


# What happens when we run value iteration with a *quadratic*?

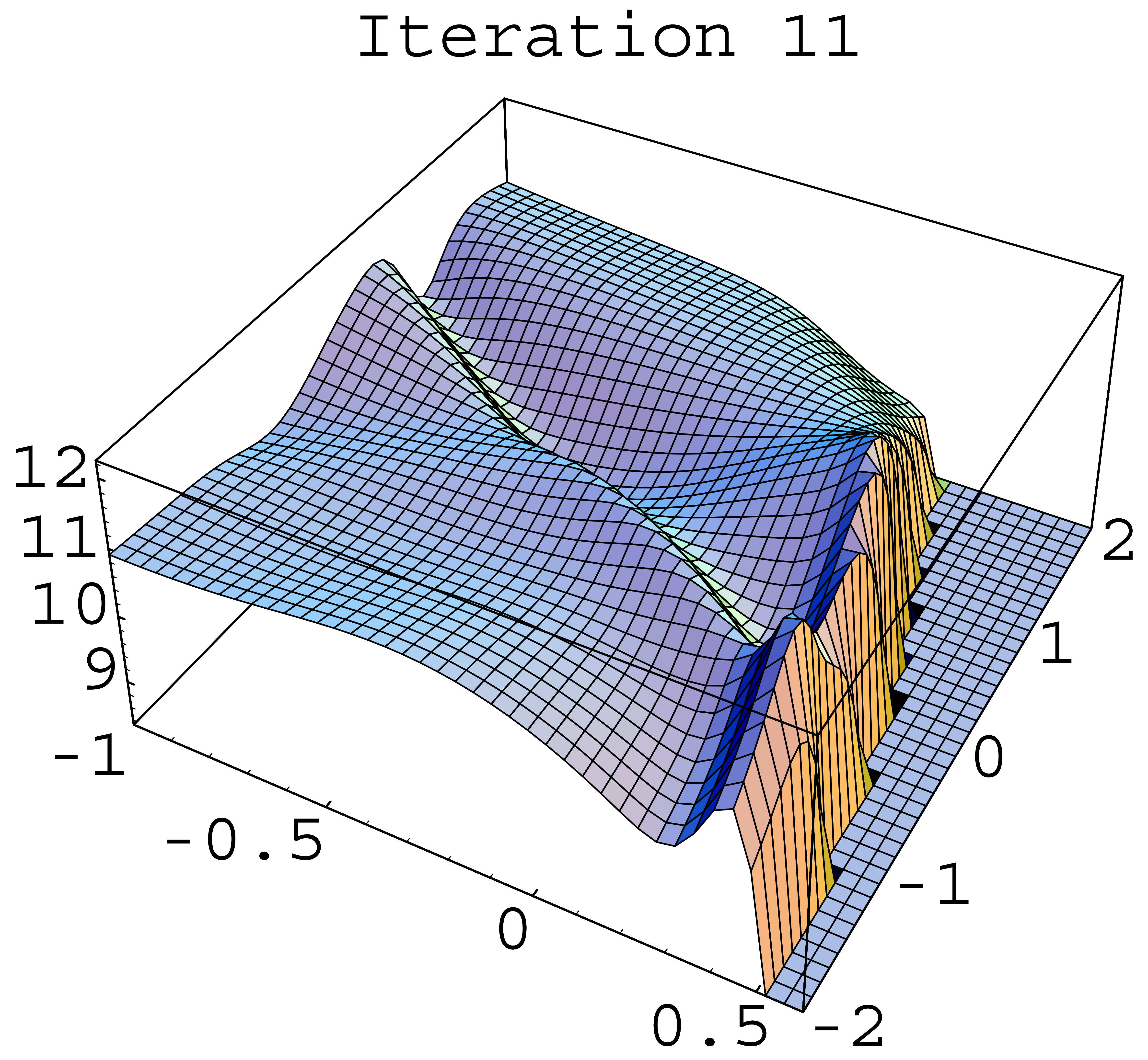


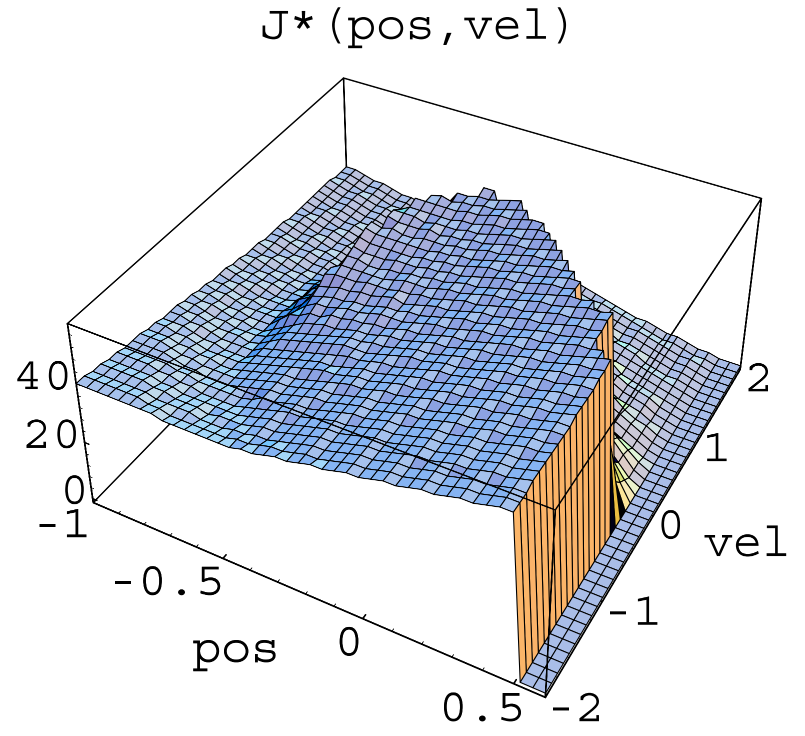
# Another Example: Mountain Car!



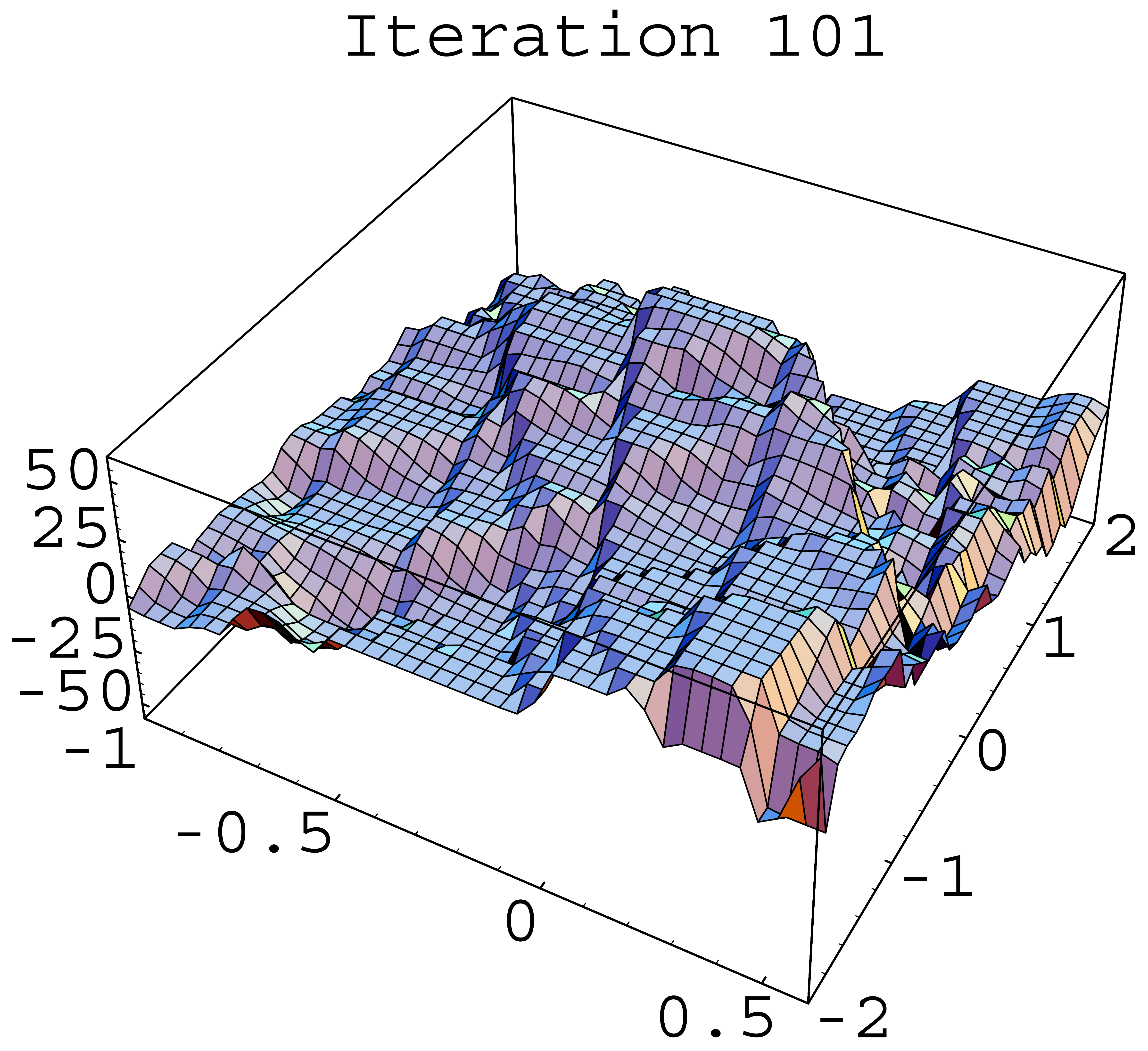



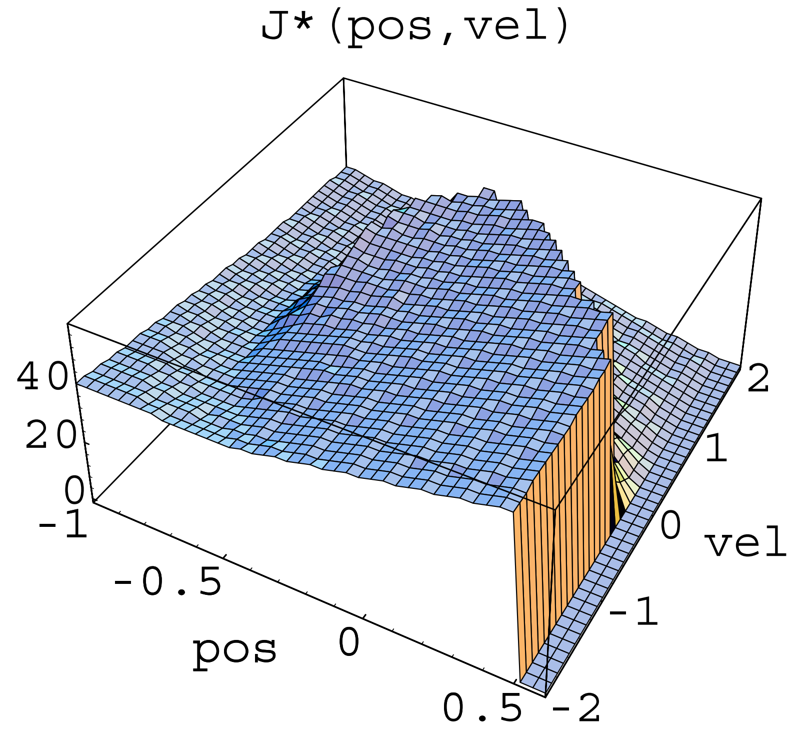
# What happens when we run value iteration with a *2 Layer MLP?*



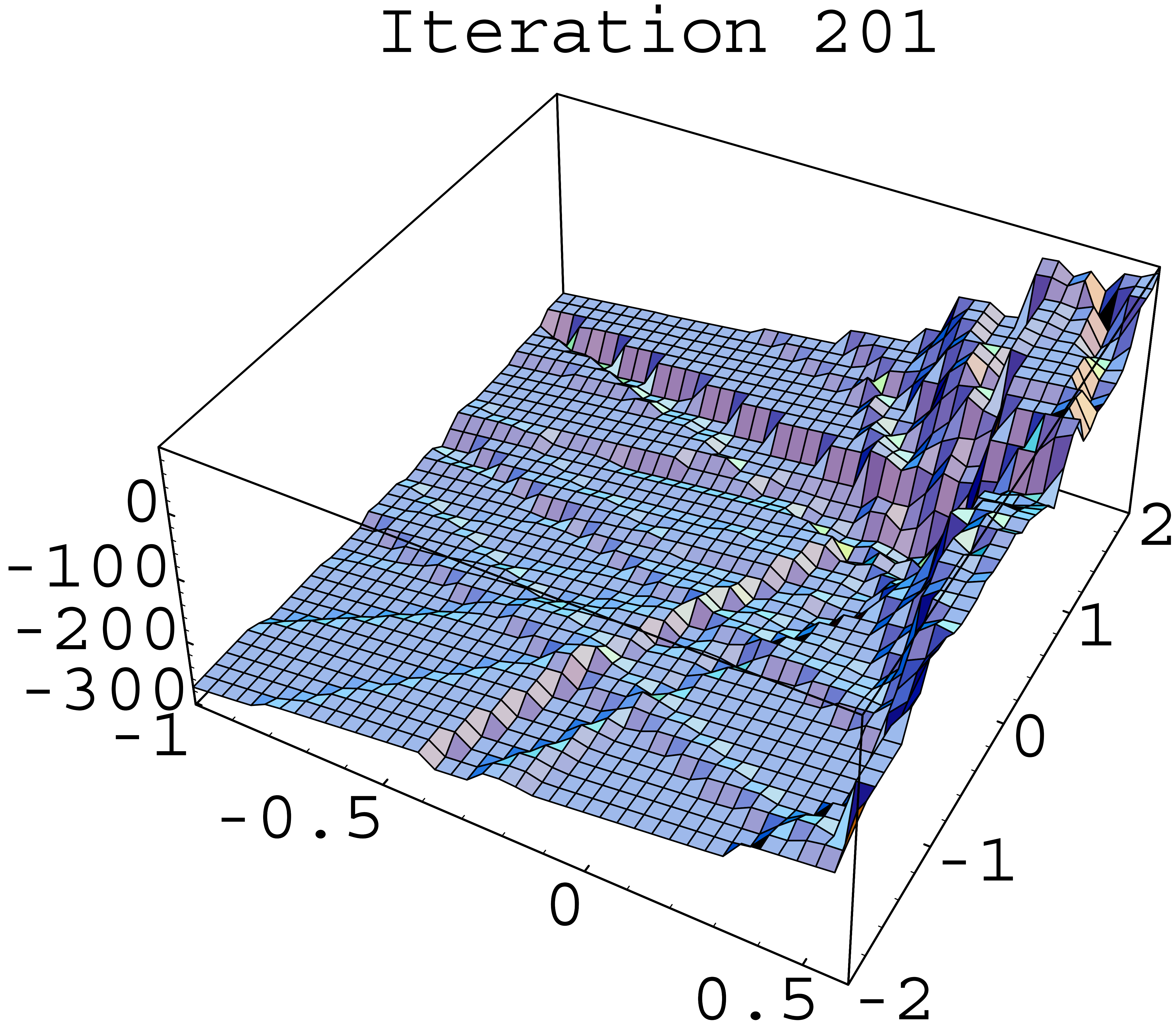



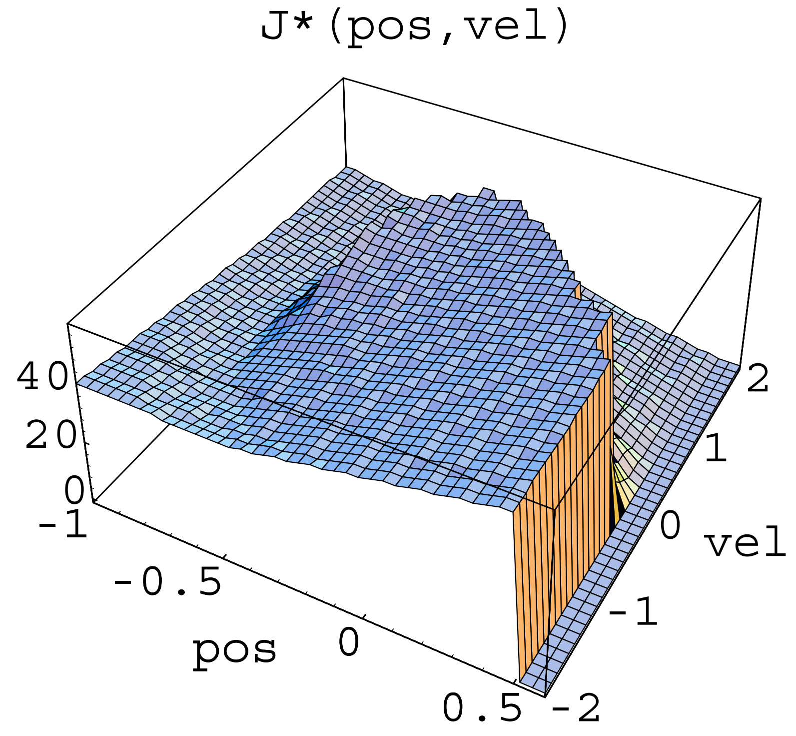
# What happens when we run value iteration with a *2 Layer MLP?*



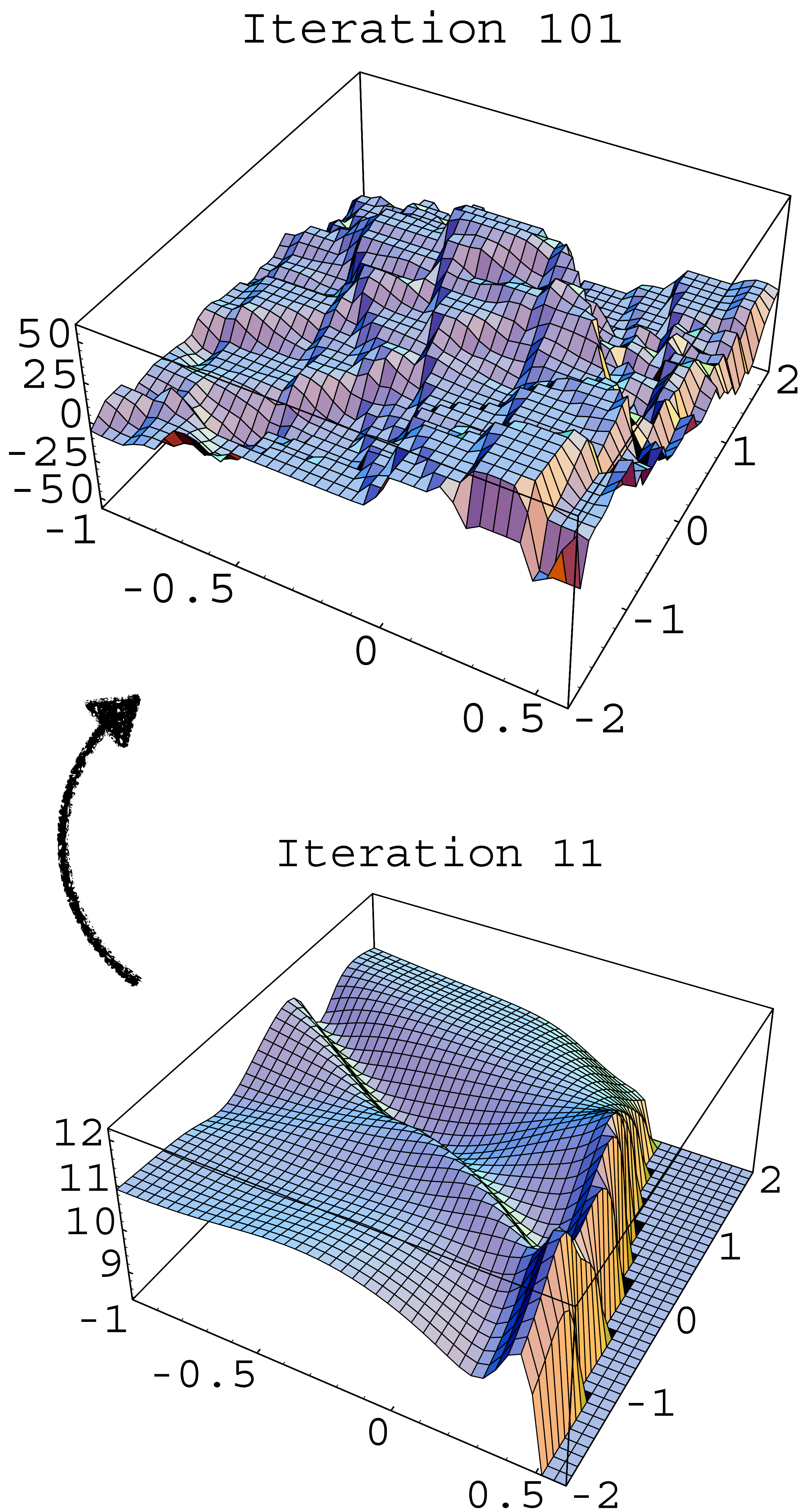
# What happens when we run value iteration with a *2 Layer MLP?*



# The problem of Bootstrapping!

max()



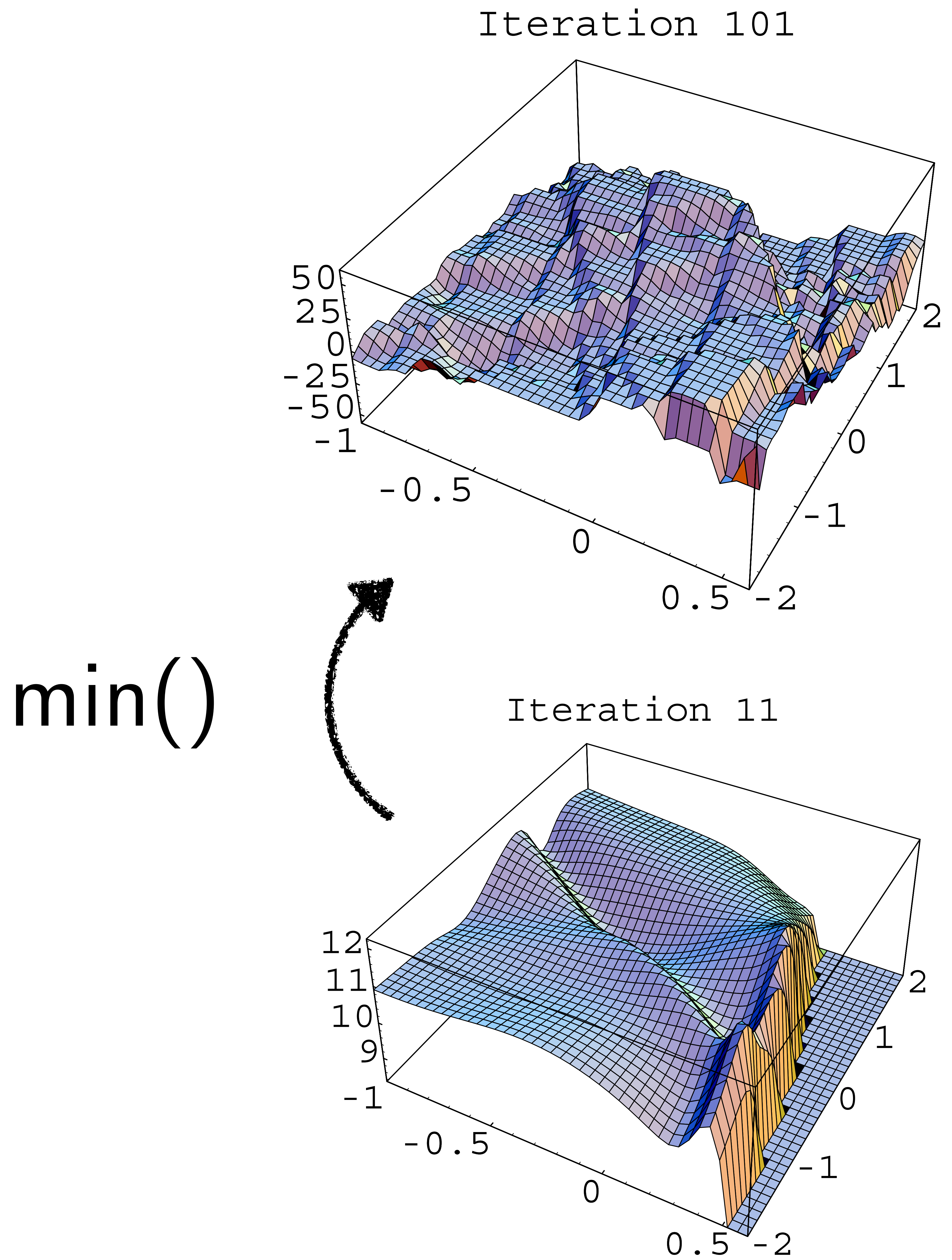
# The problem of Bootstrapping!

Errors in approximation are amplified

Key reason is the minimization

Minimization operation visit states where approximate values is less than the true value of that state – that is to say, states that look more attractive than they should.

Typically states where you have very few samples

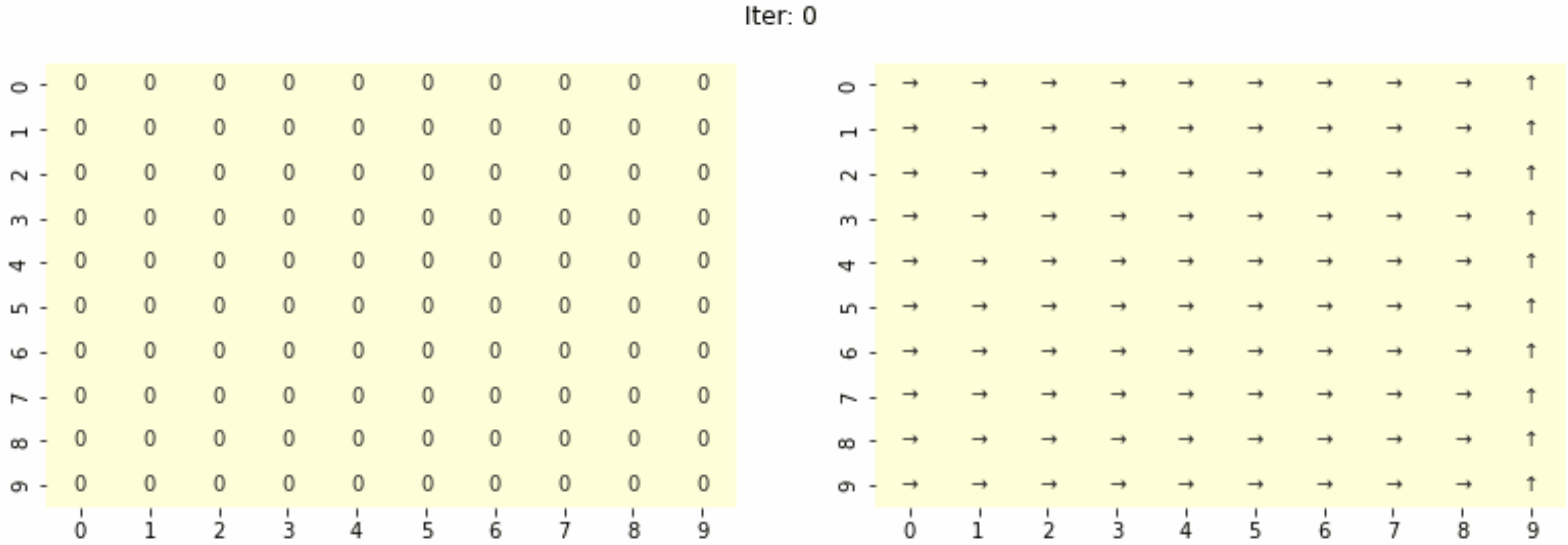


What about policy iteration?

# Policy Iteration

## Policy Evaluation

## Policy Improvement



# Policy Iteration

Init with some policy $\pi$

Repeat forever

Evaluate policy

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

Improve policy

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

# Fitted Policy Iteration

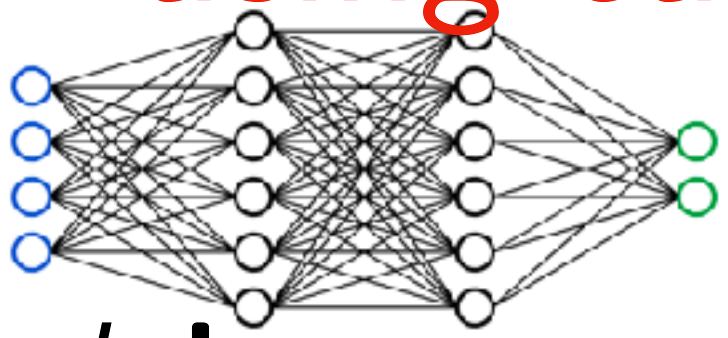
## *Fitted* policy evaluation

Collect data

using current policy $\pi$

**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** $Q_\theta$

## Policy Improvement

This remains the same!

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

# 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**

This is fine..

$D \leftarrow \emptyset$

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

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

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

No min() step

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

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

**return** $Q_\theta$

## Policy Improvement

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

# 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** $i \in 1, \ldots, n$

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** $Q_\theta$

This is fine..

## Policy Improvement

But this has the min() step!

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