Reinforcement Learning IV
Model-based Planning and Value Function Approximation

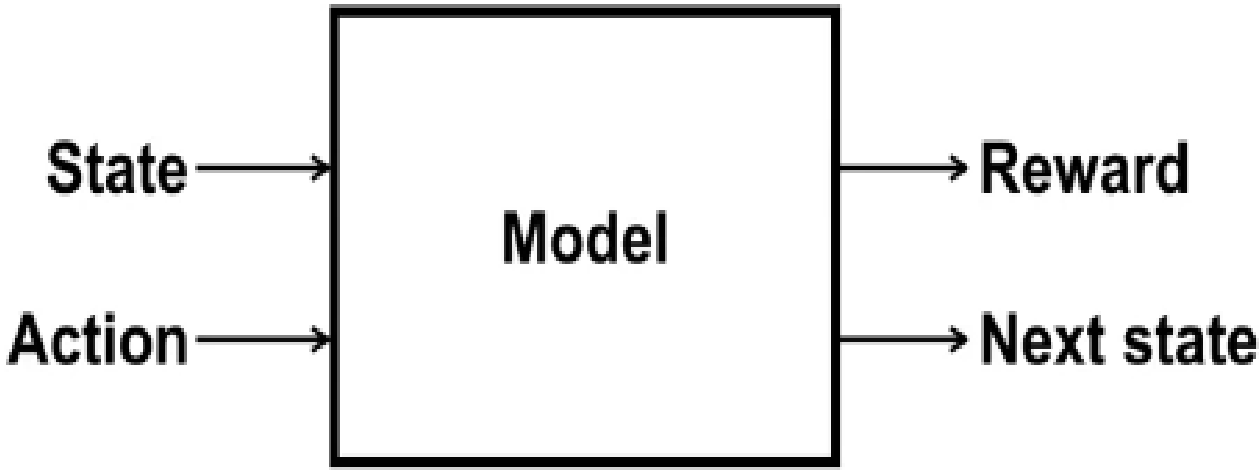
Antoine SYLVAIN

EPITA

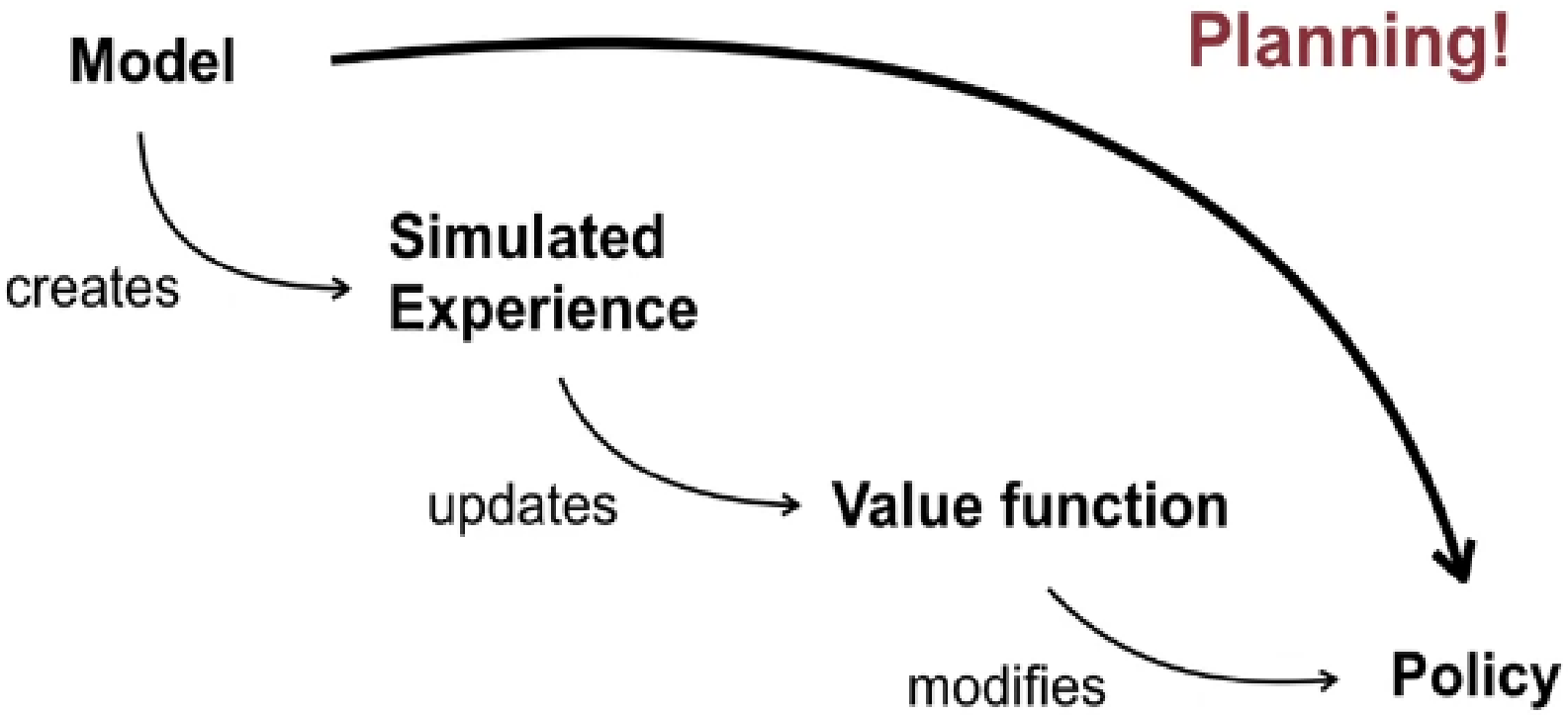
2021

Contents

What is a model ?



What is a model ?



Types of models

- We distinguish two types of models:

Types of models

- We distinguish two types of models:
  - **Sample models**: produces an actual outcome drawn from some underlying probabilites
    e.g.: generating a sequence of heads and tails by flipping a coin

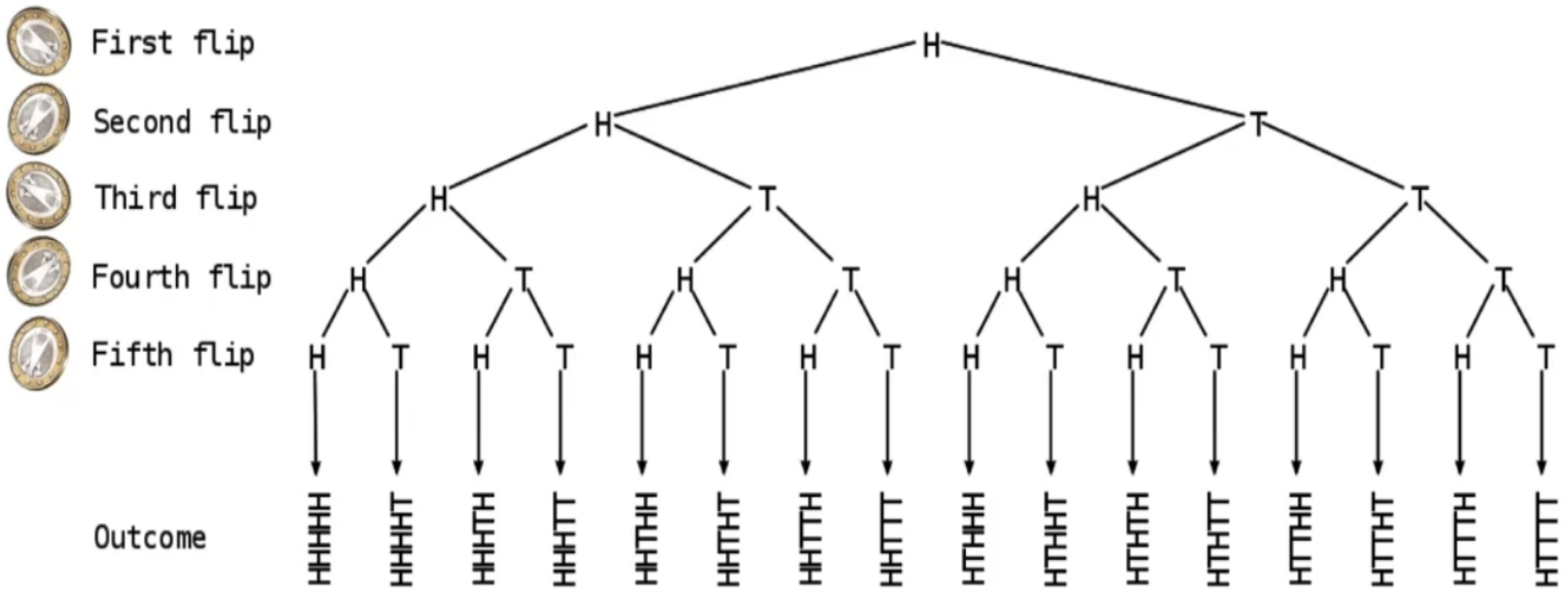
Types of models

- We distinguish two types of models:
  - **Sample models**: produces an actual outcome drawn from some underlying probabilites
    e.g.: generating a sequence of heads and tails by flipping a coin
  - **Distribution models**: specifies completely the likelihood or probability of every outcome
    e.g.: generating the tree of possible outcomes of a sequence of coins flipping

Sample model

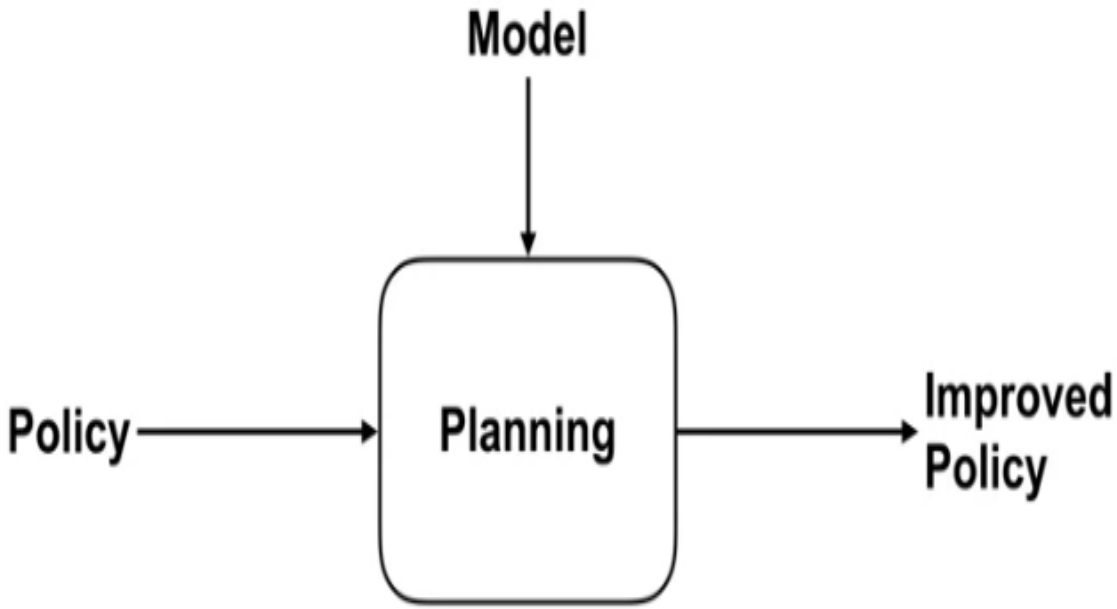
Head, Tail, Tail, Head, Tail

Distribution model

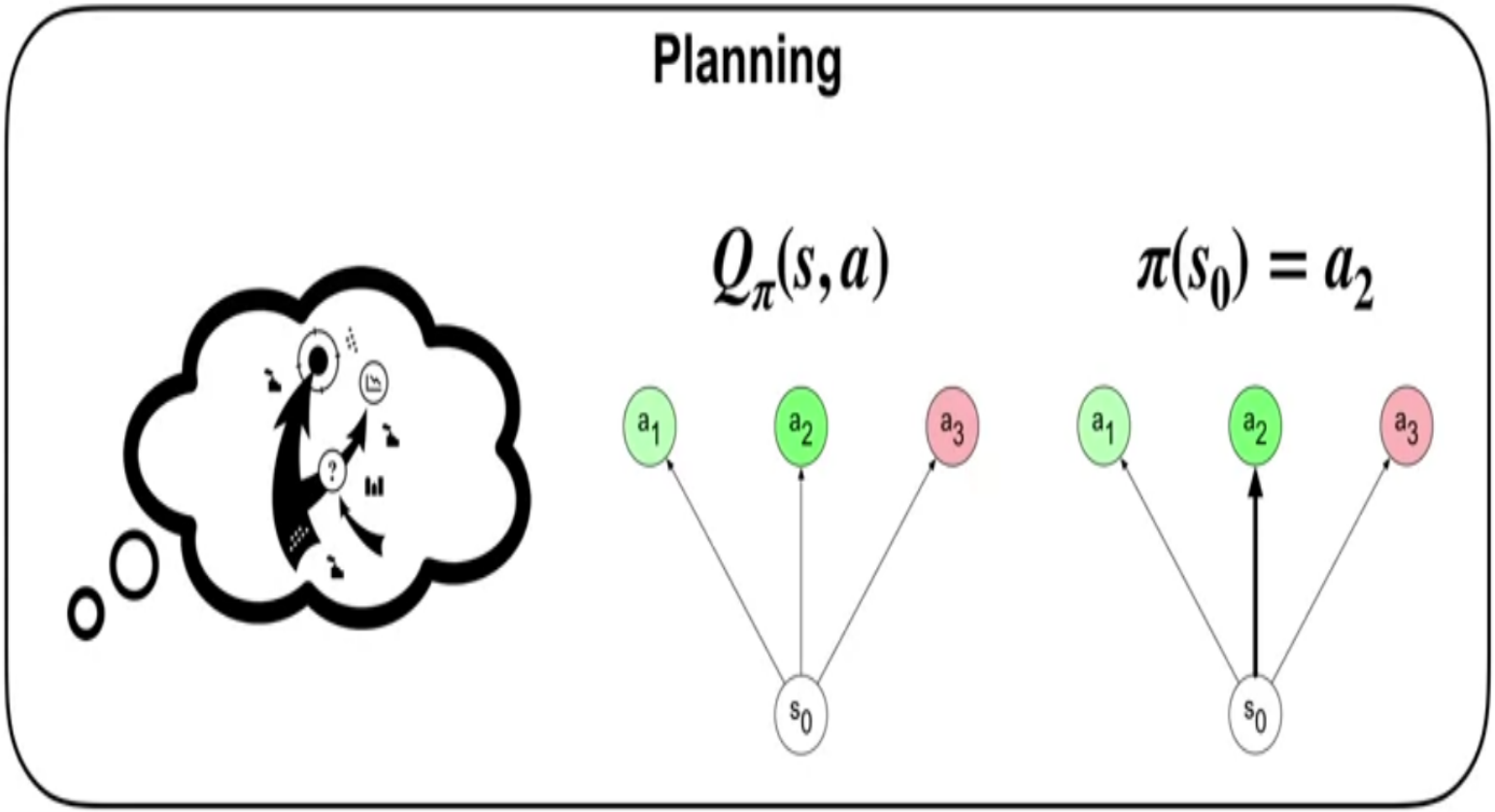


$$p = \frac{1}{32}$$

Planning



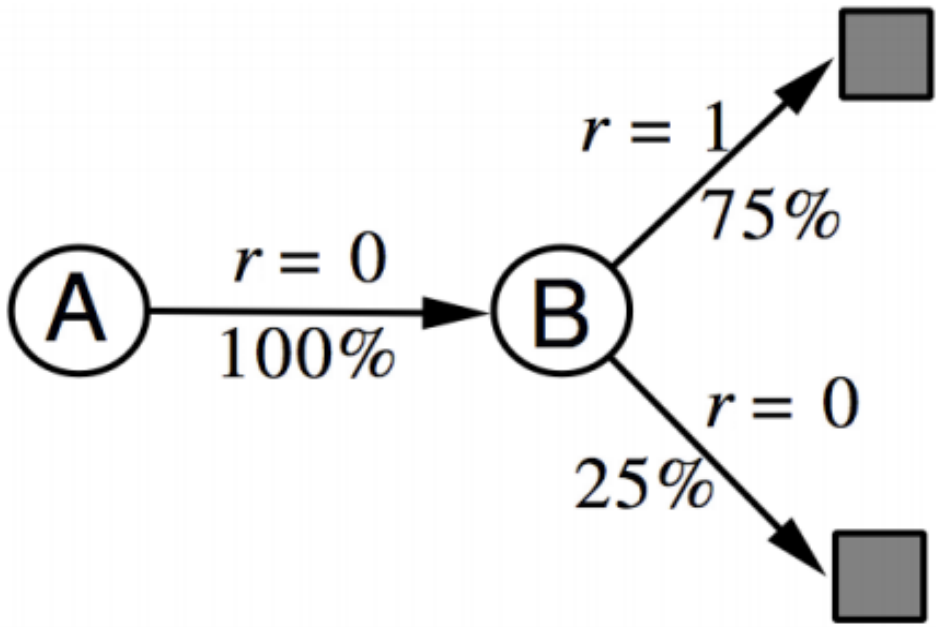
Planning



AB Example

Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



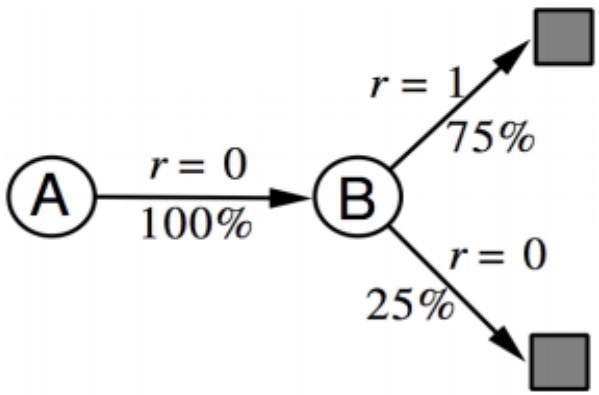
We have constructed a table lookup model from the experience

AB Example with a model

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

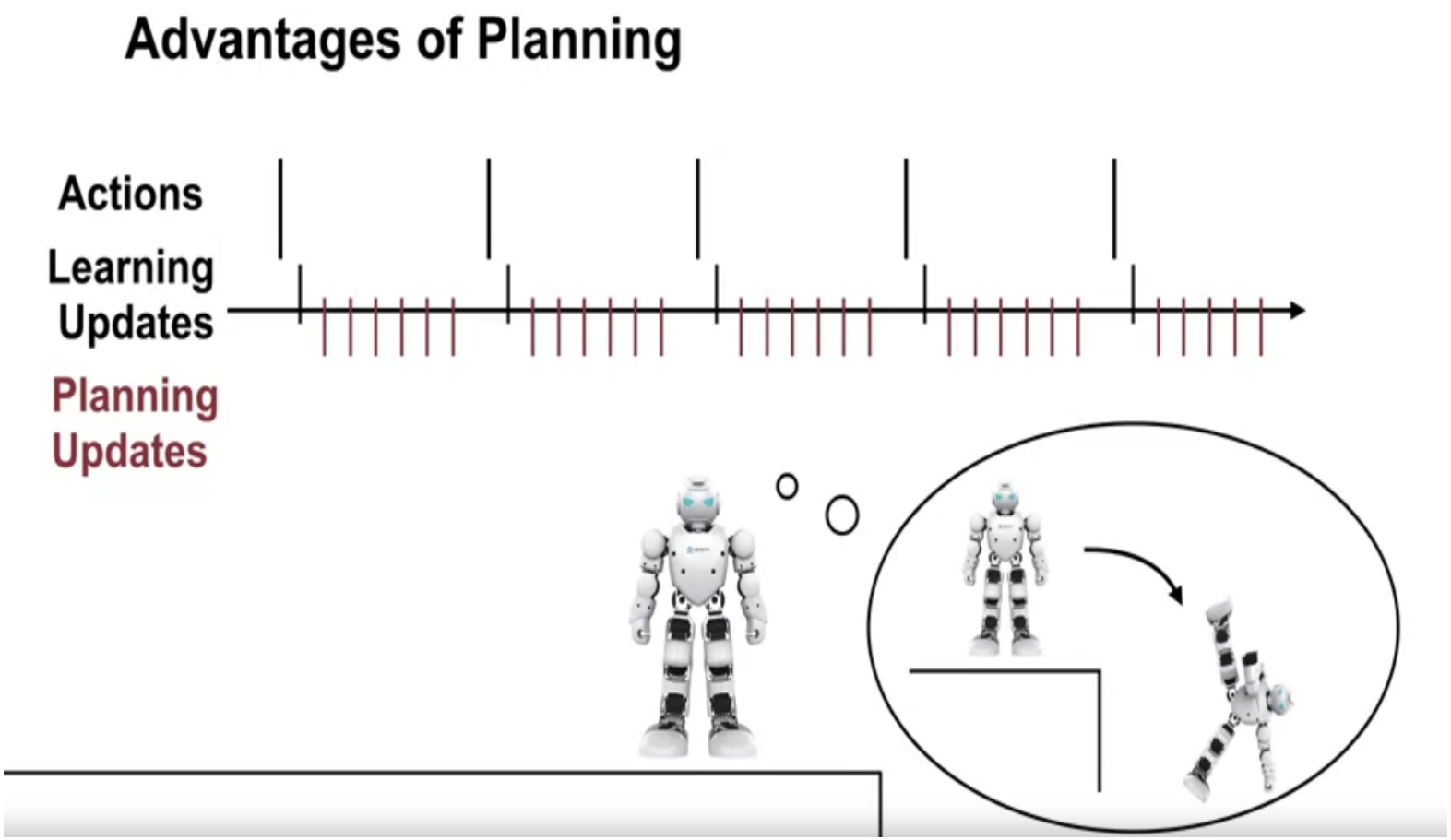


Sampled experience

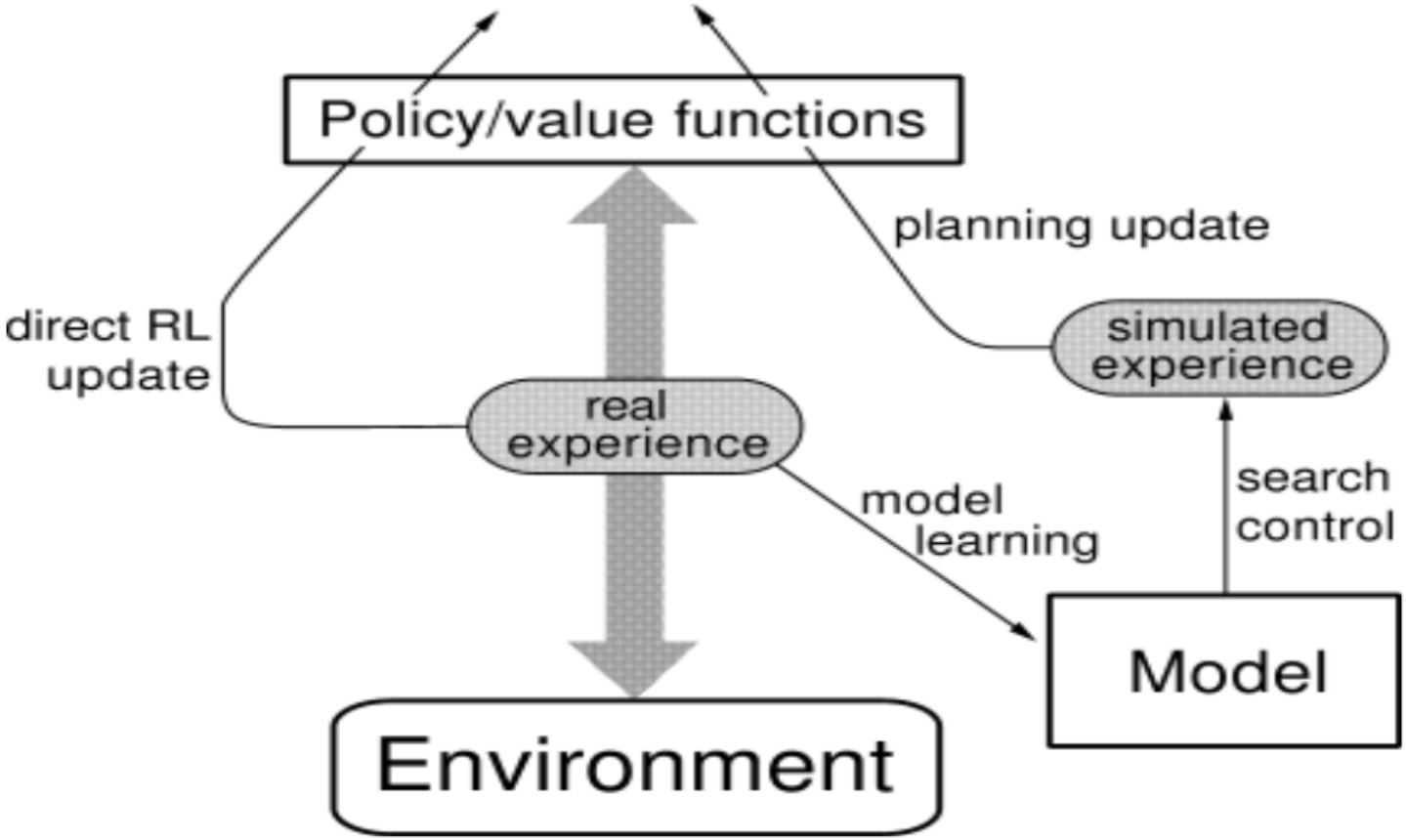
B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Planning


Advantages of Planning
Actions
Learning Updates
Planning Updates

Dyna Architecture



Dyna-Q Algorithm

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

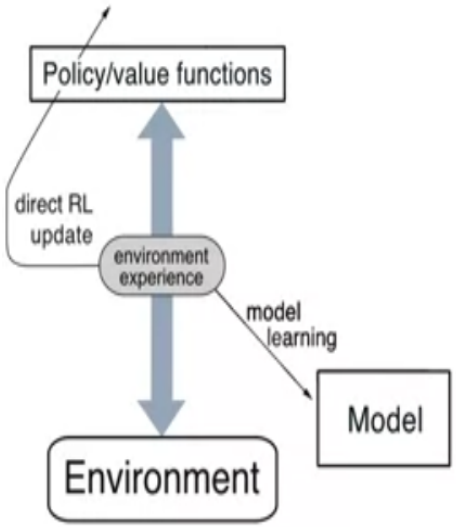
(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Dyna-Q Algorithm



Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

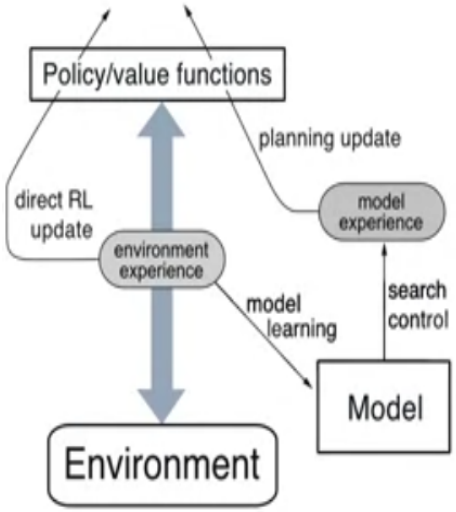
(b) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

Dyna-Q Algorithm



Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Loop repeat n times:

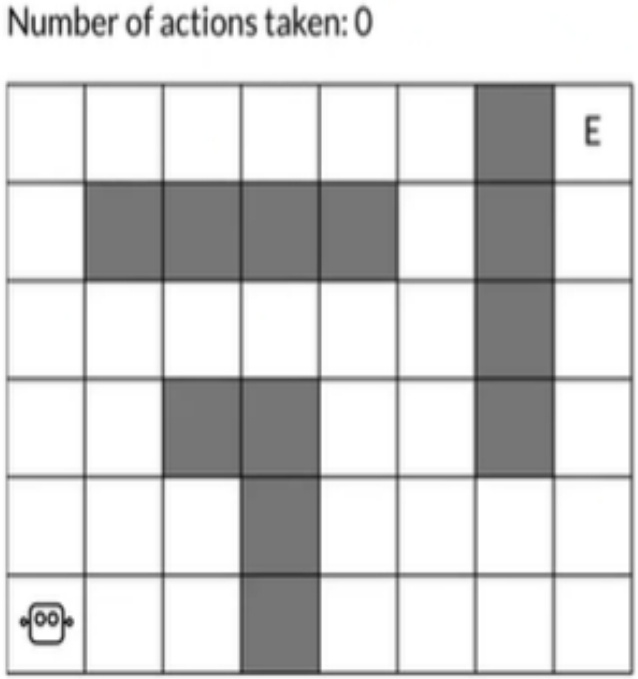
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

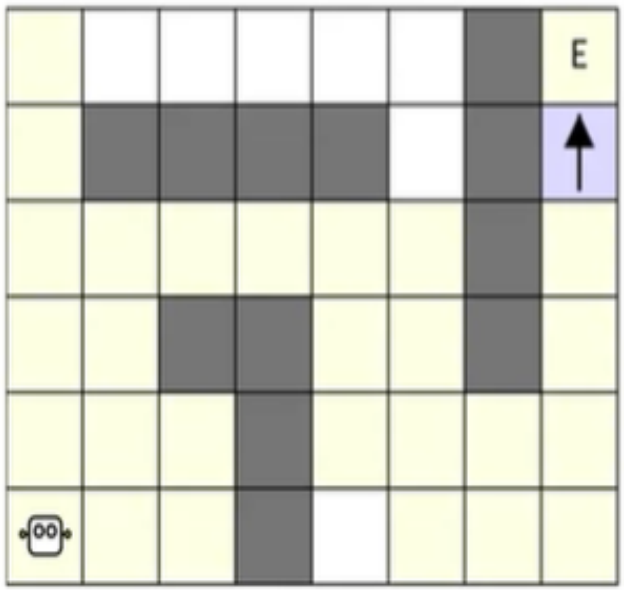
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Dyna-Q Maze Example

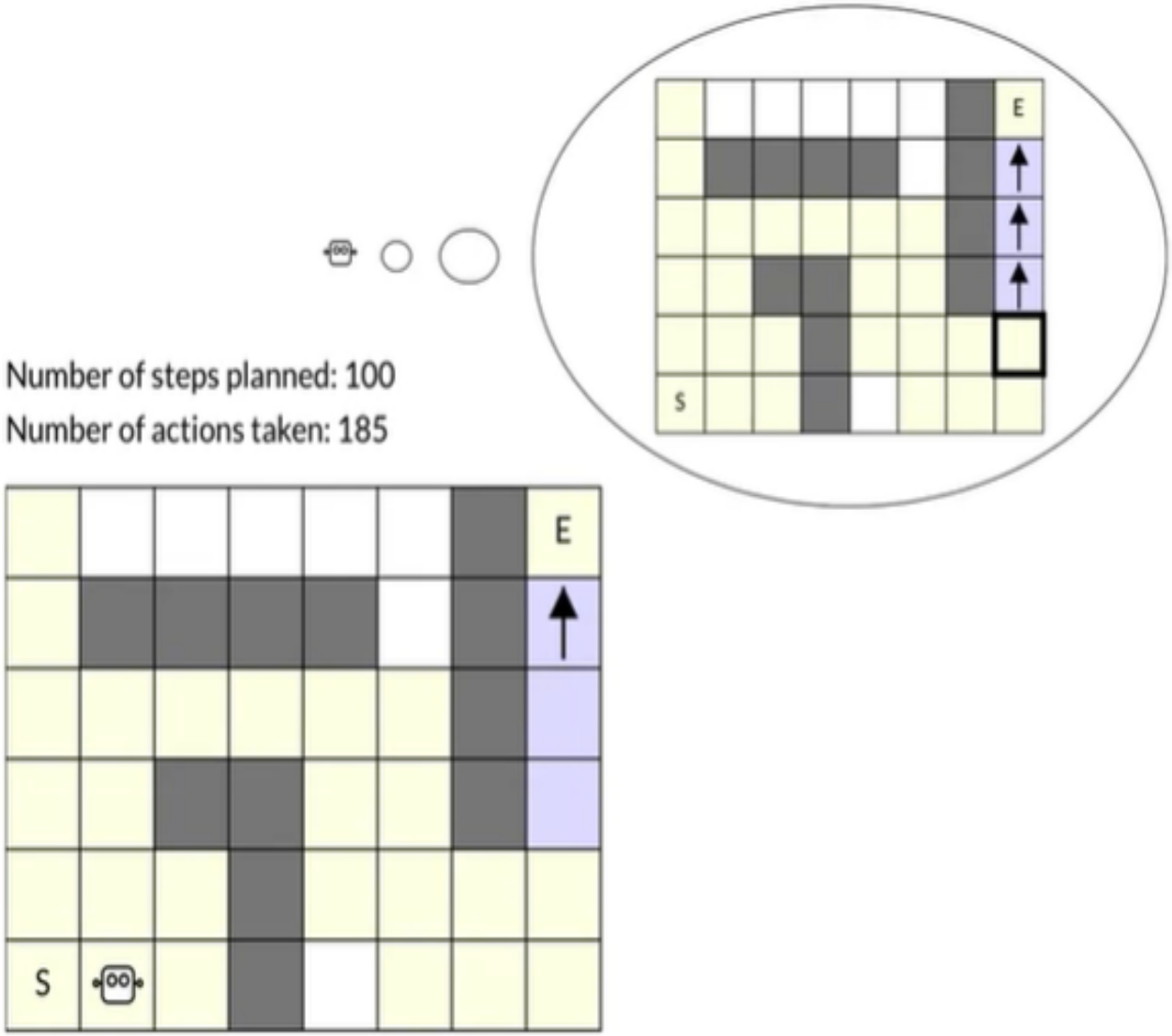


Dyna-Q Maze Example

Number of actions taken: 184



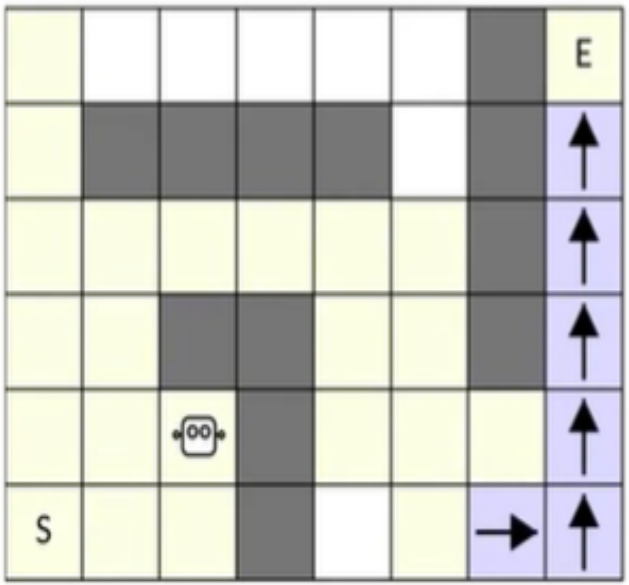
Dyna-Q Maze Example



Dyna-Q Maze Example

Number of steps planned: 400

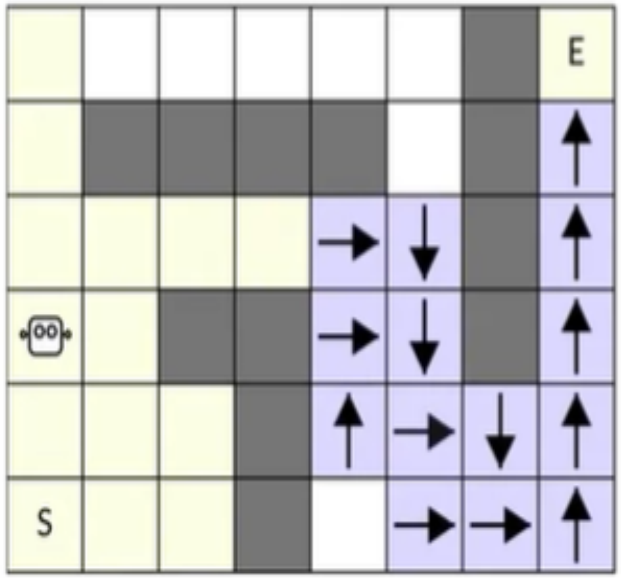
Number of actions taken: 188



Dyna-Q Maze Example

Number of steps planned: 1200

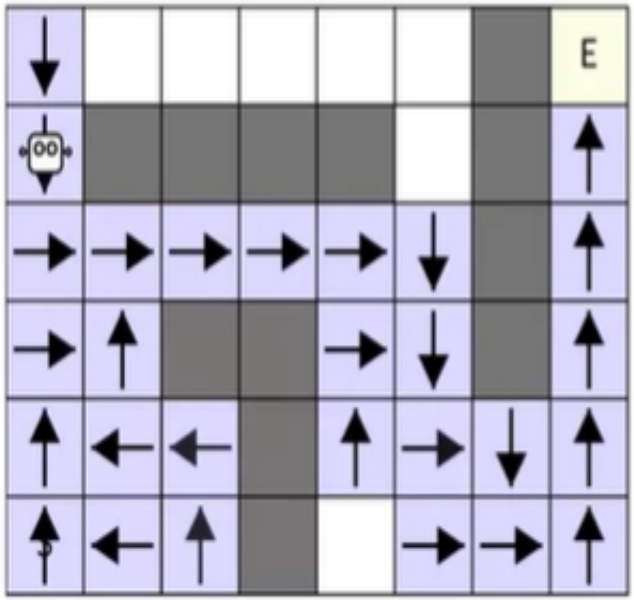
Number of actions taken: 196



Dyna-Q Maze Example

Number of steps planned: 1800

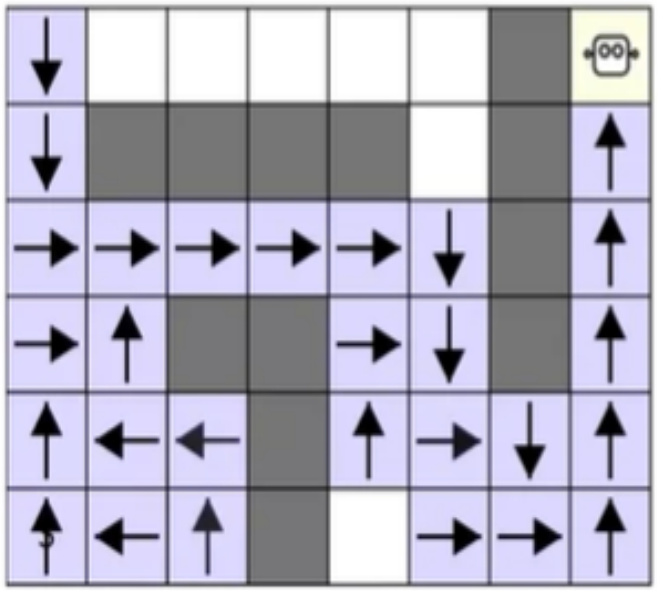
Number of actions taken: 202



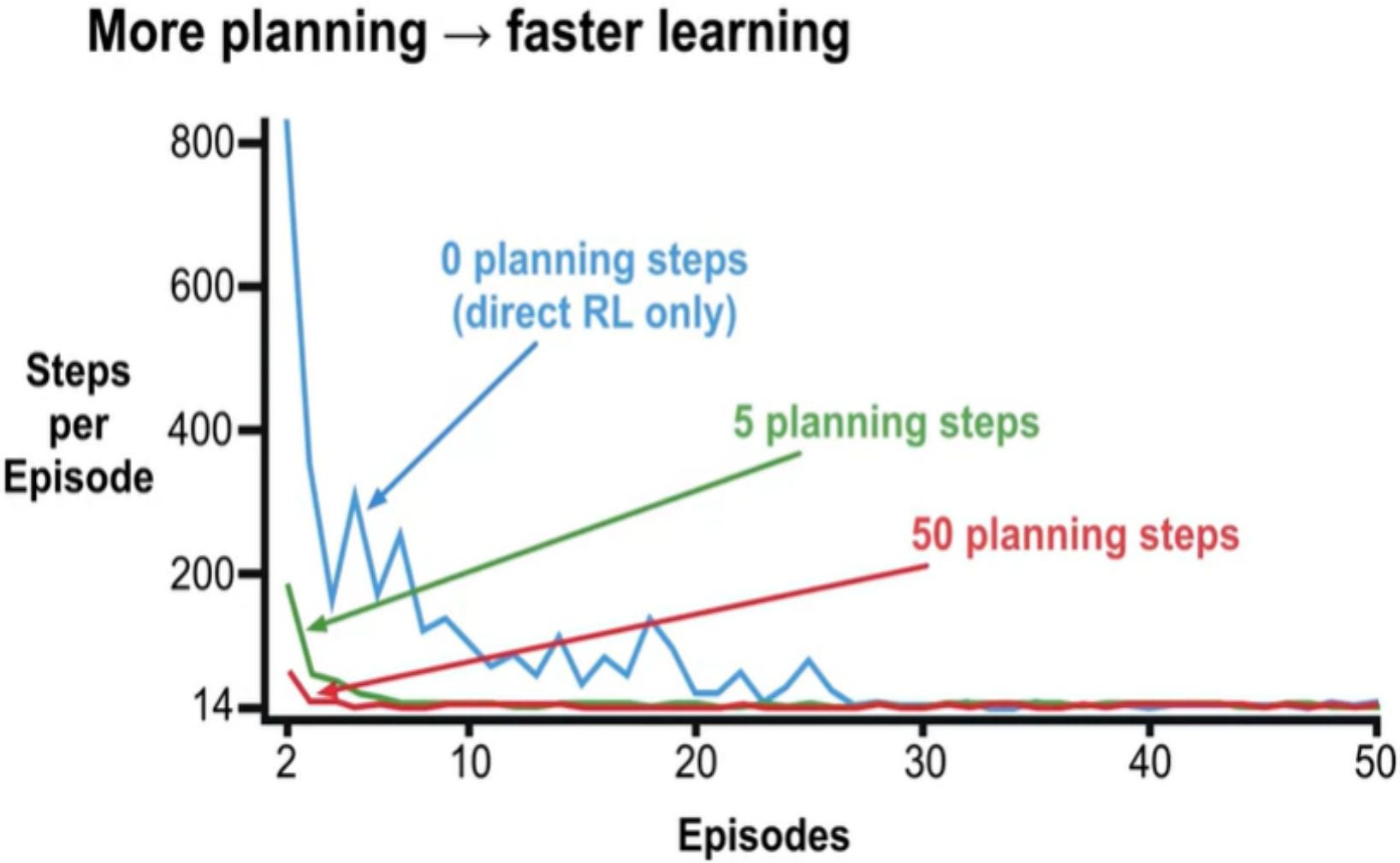
Dyna-Q Maze Example

Number of steps planned: 3800

Number of actions taken: 223



Dyna-Q Planning Efficiency

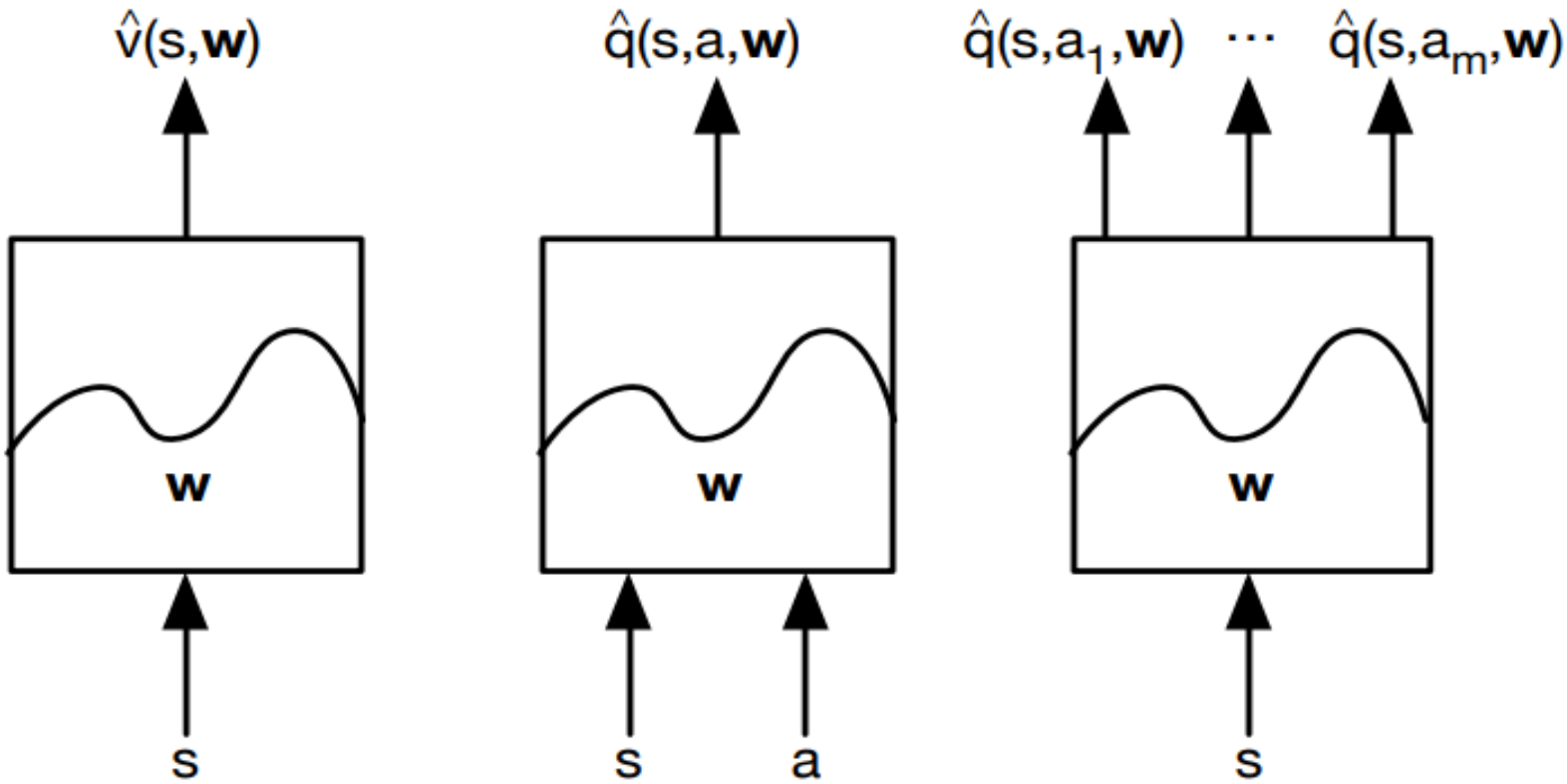


Contents

Value Function Approximation

- Estimates value function with function approximation:
  $\hat{v}(s, w) \approx v_\pi(s)$
  $\hat{q}(s, a, w) \approx q_\pi(s, a)$
- Generalise from seen states to unseen states
- Update parameter w (e.g. with Monte-Carlo or Temporal-Difference Learning)

Types of Approximators



Function Approximators

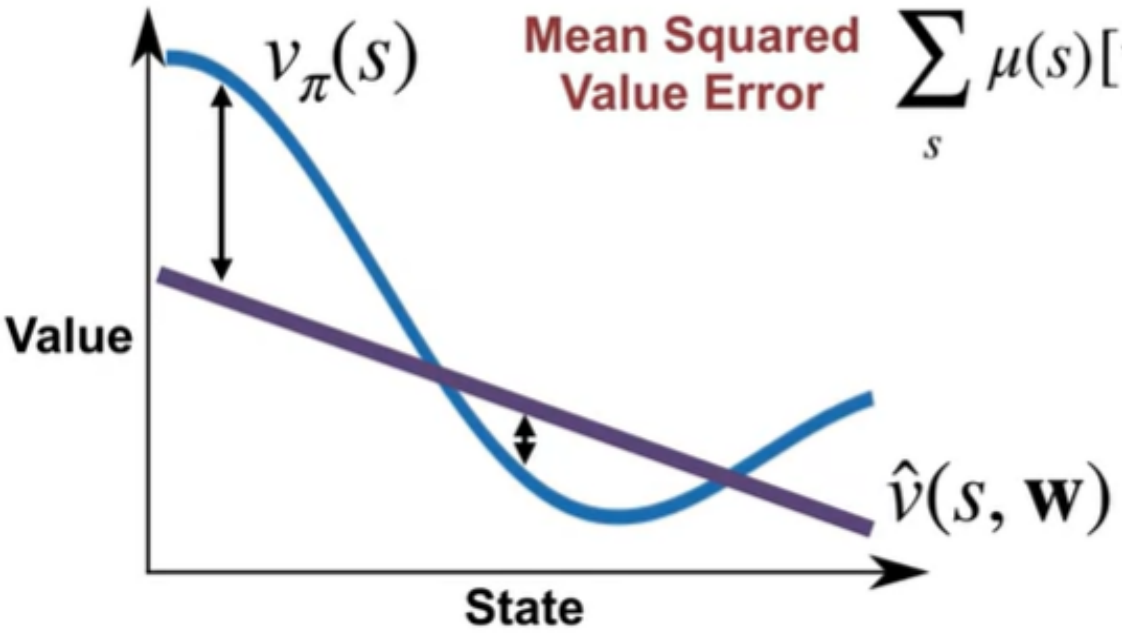
- Linear combination of features
- Neural Networks
- Decision Trees
- Nearest neighbour
- Fourier / wavelets base
- ...

Function Approximators

- Linear combination of features
- Neural Networks
- Decision Trees
- Nearest neighbour
- Fourier / wavelets base
- ...
- Then, a training method

Value Error Objective

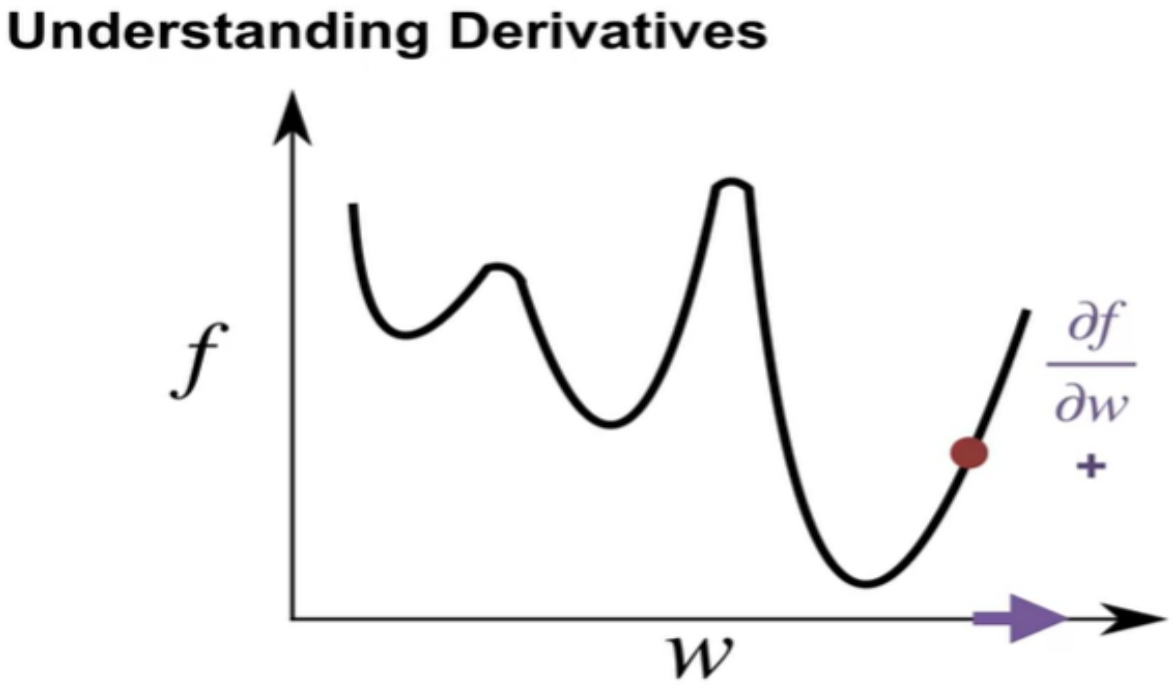
The Mean Squared Value Error Objective



Mean Squared Value Error $\sum_s \mu(s)[v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$

Contents

Gradient Derivatives



Gradient: Derivatives in multiple dimensions

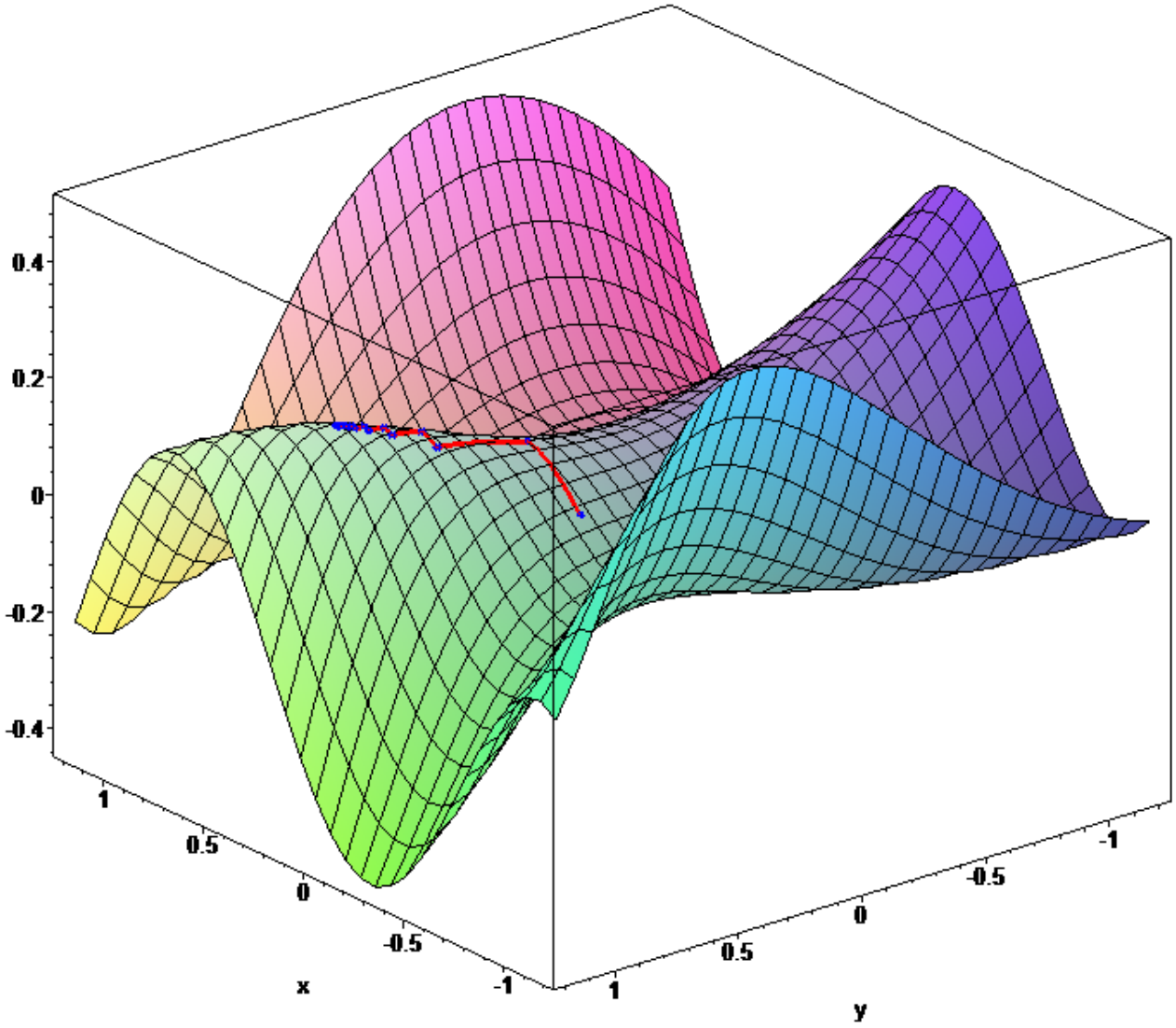
$$\mathbf{w} \doteq \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{bmatrix} \qquad \nabla f \doteq \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \cdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}$$

The direction to change w_2 in order to increase f

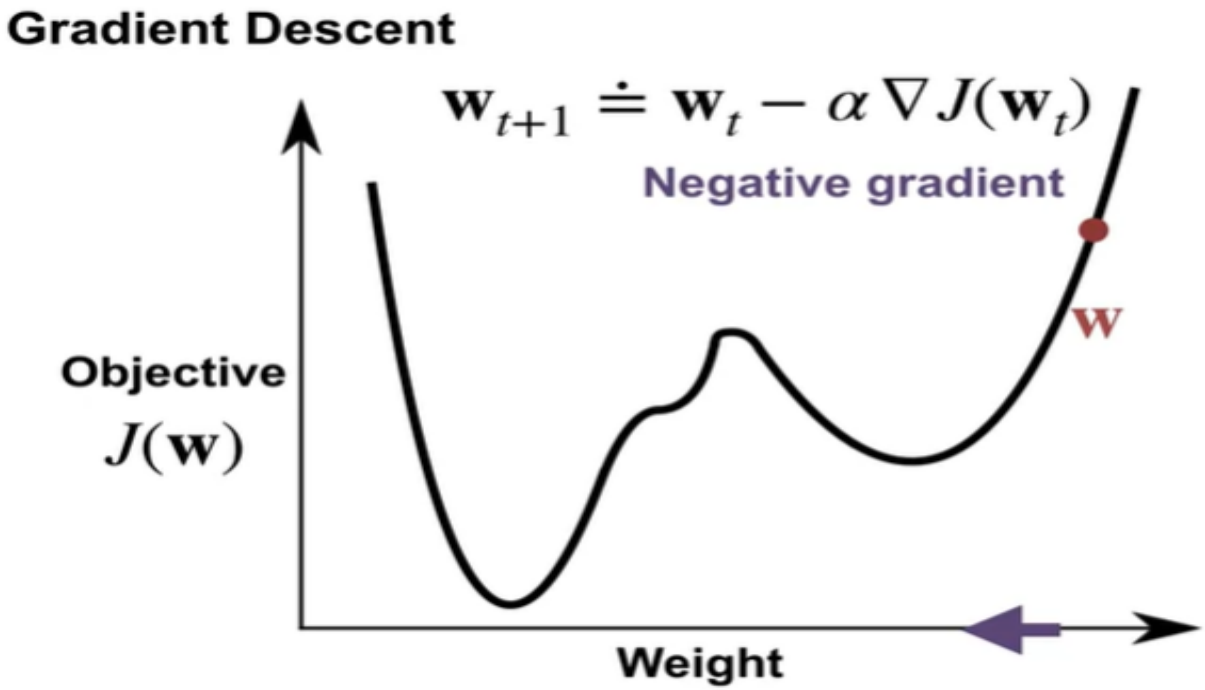
How quickly f changes

The gradient gives the direction of steepest ascent

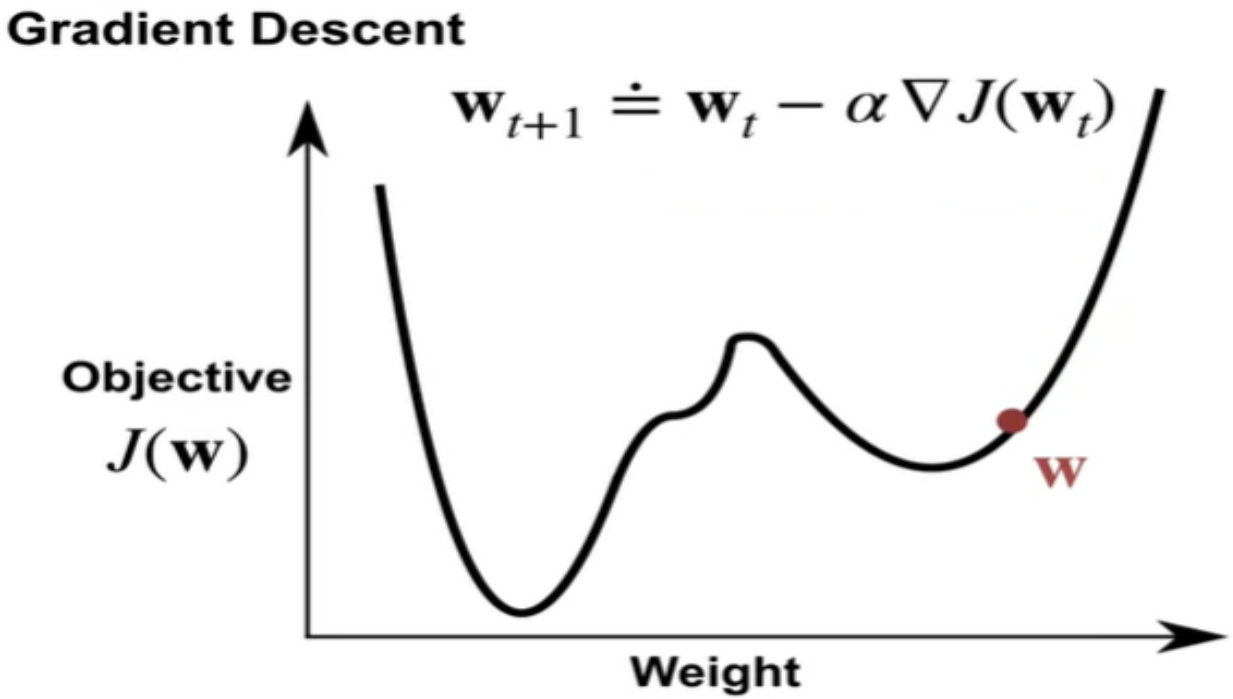
Gradient: Derivatives in multiple dimensions



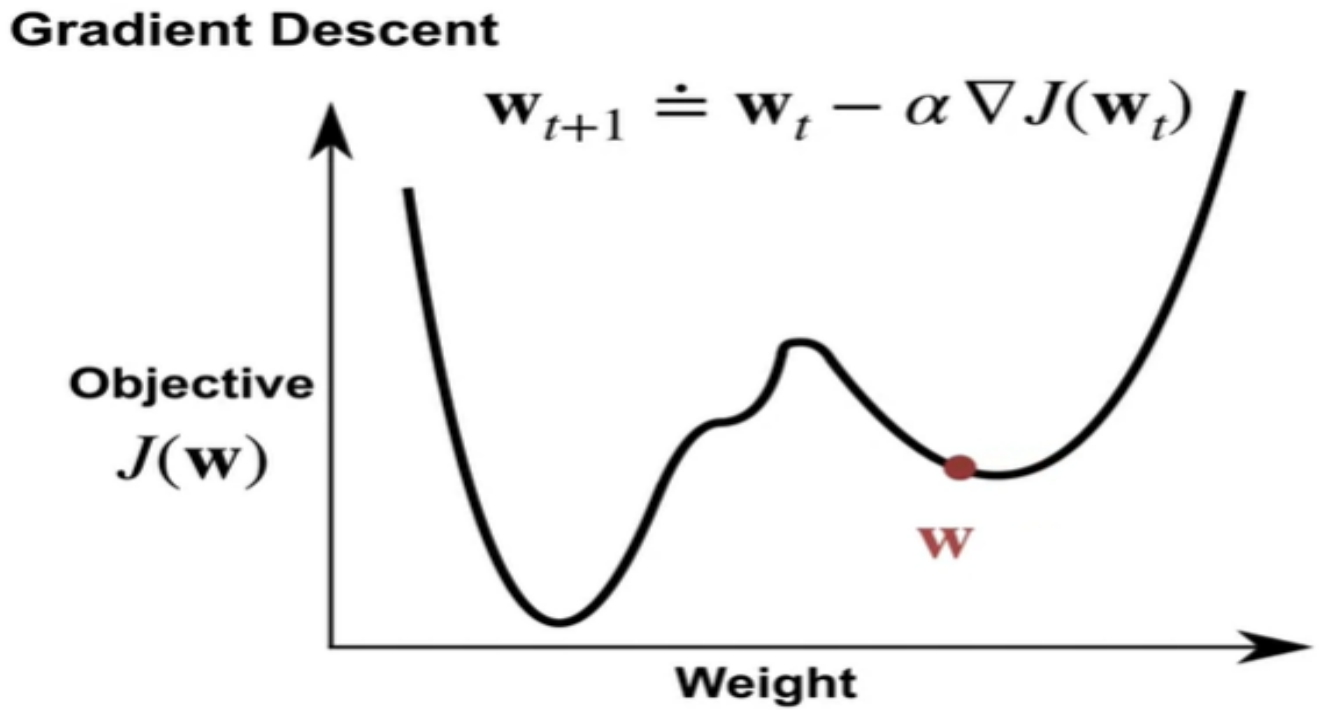
Gradient Descent



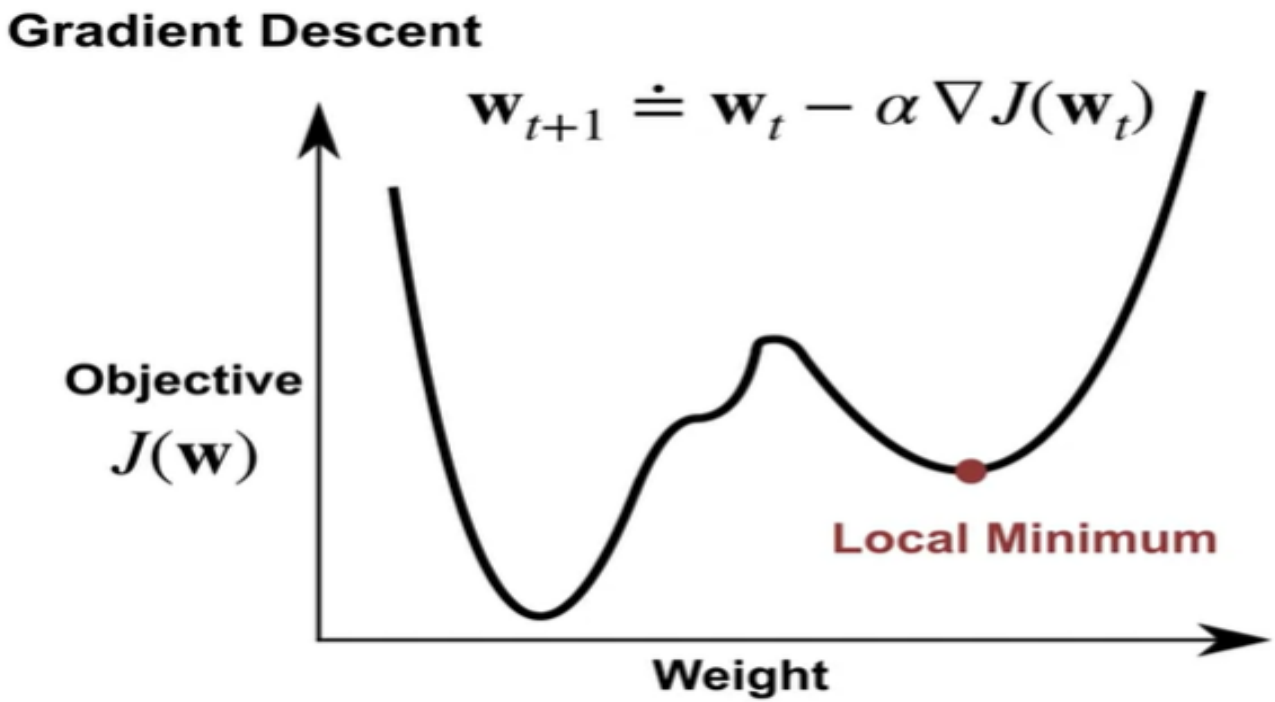
Gradient Descent



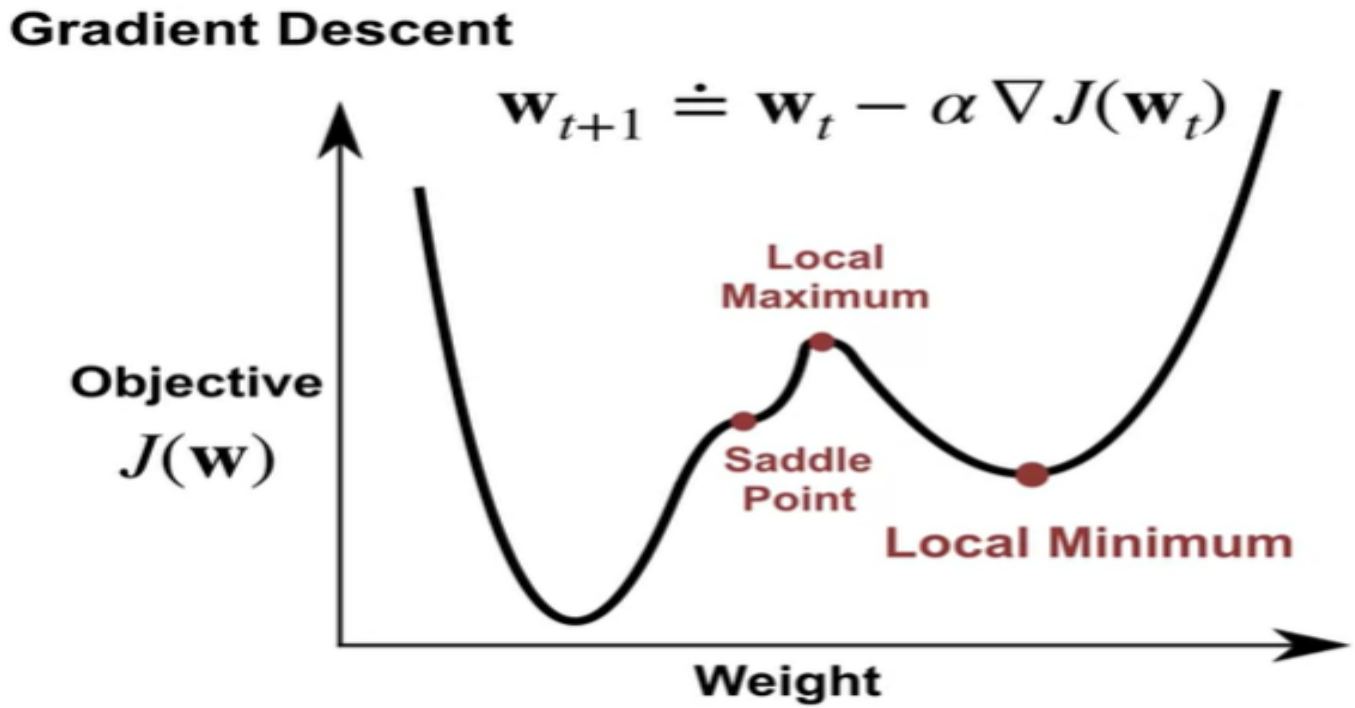
Gradient Descent



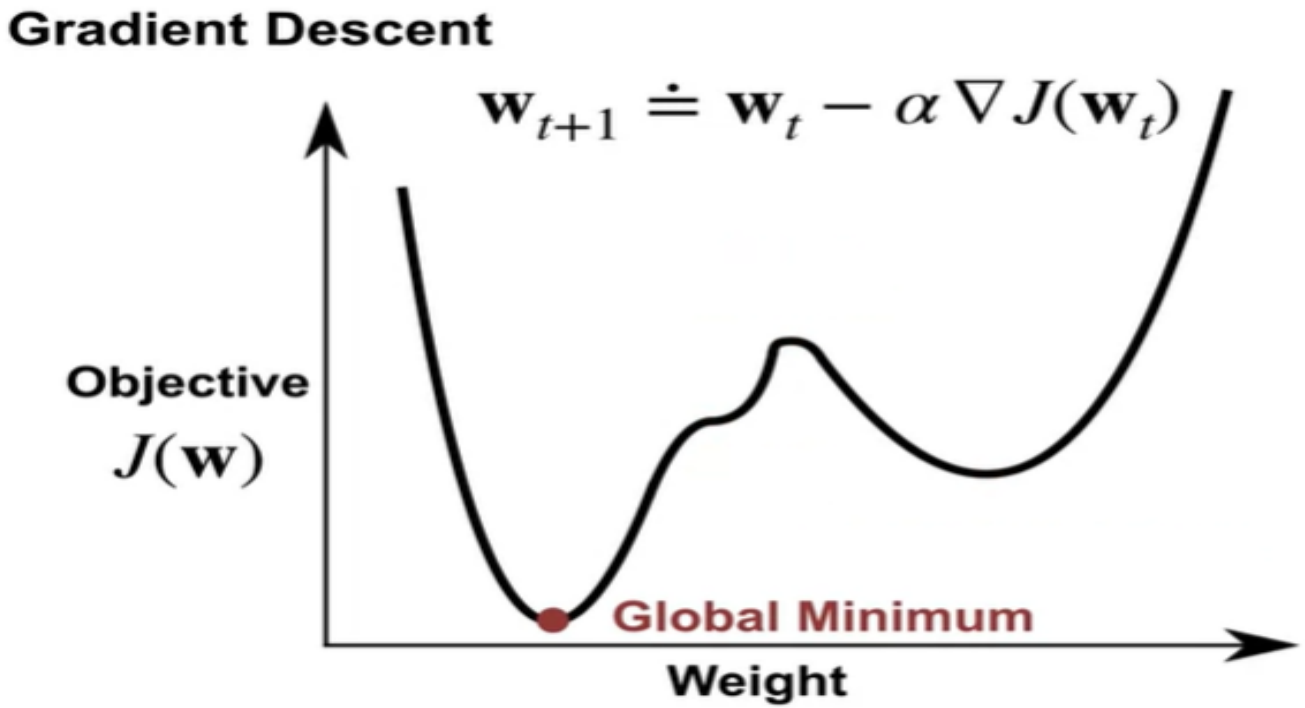
Gradient Descent



Gradient Descent



Gradient Descent



Gradient Monte-Carlo

Gradient Monte Carlo

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)] \nabla \hat{v}(S_t, \mathbf{w}_t)$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[G_t - \hat{v}(S_t, \mathbf{w}_t)] \nabla \hat{v}(s, \mathbf{w}_t)$$

Recall that

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$\mathbb{E}_\pi[2[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})]$$

$$= \mathbb{E}_\pi[2[G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})]$$

Gradient MC Algorithm

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : S \times \mathbb{R}^d \to \mathbb{R}$
Algorithmic parameter: step size $\alpha > 0$
Initialize arbitrarily value function weights $w \in \mathbb{R}^d$ (e.g. $w = 0$)

Loop (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, ..., R_\tau, S_\tau$ using π
  Loop for each episode step $t = 0, 1, ..., \tau - 1$:
    $w \leftarrow w + \alpha[G_t - \hat{v}(S_t, w)] \bigtriangledown \hat{v}(S_t, w)$

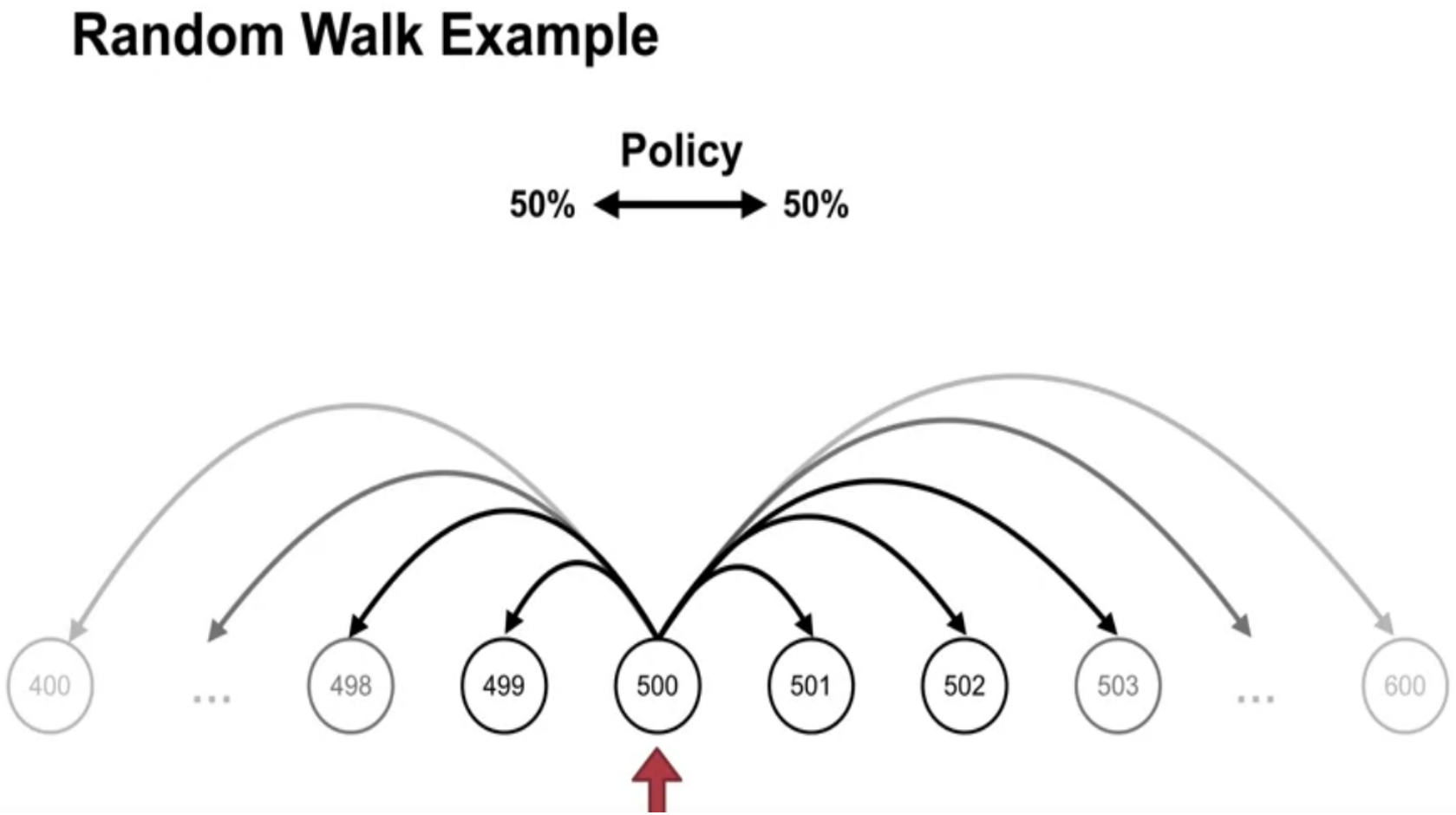
State Aggregation with Monte-Carlo

Random Walk Example

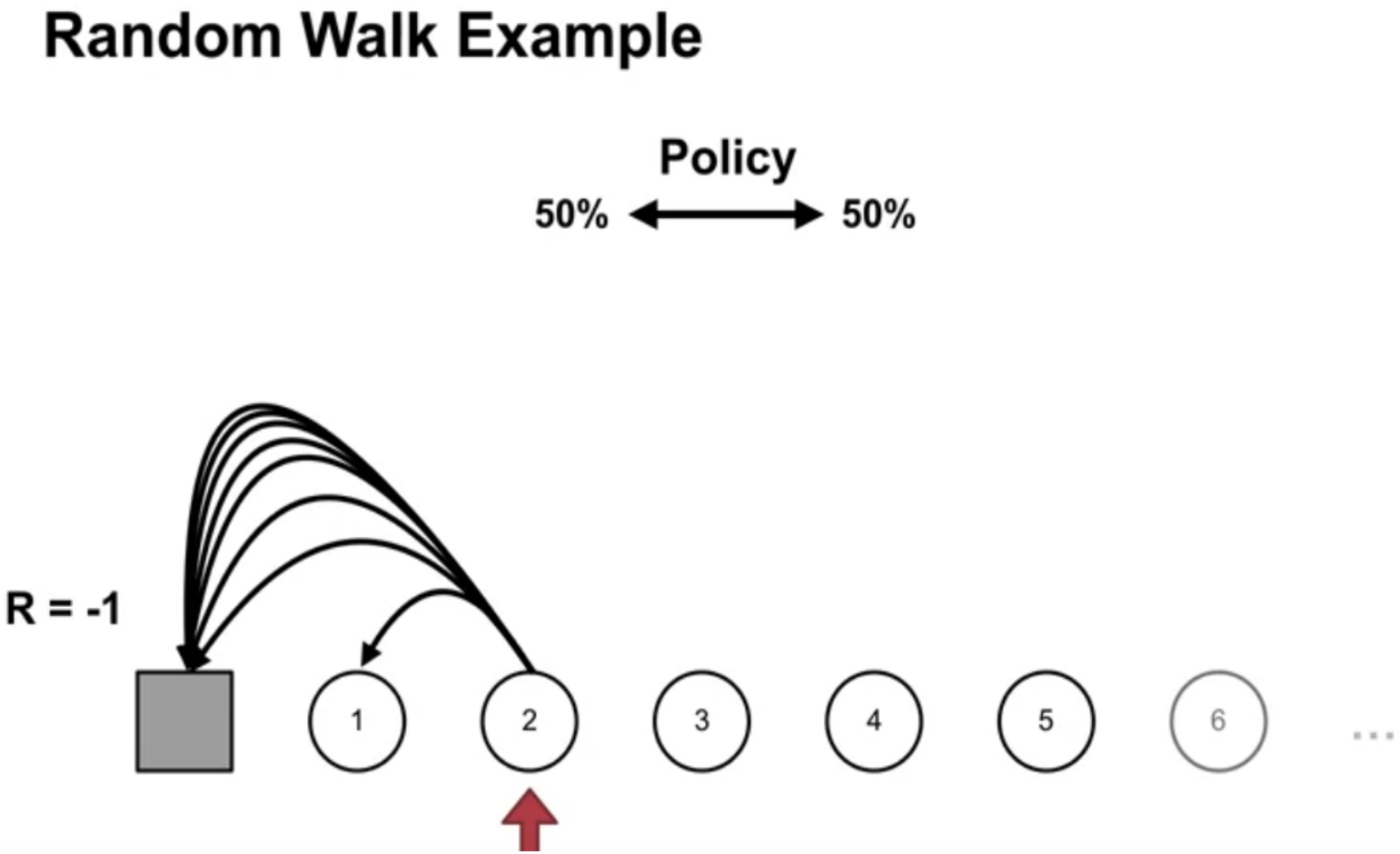
State Aggregation with Monte-Carlo

Random Walk Example

State Aggregation with Monte-Carlo



State Aggregation with Monte-Carlo



State Aggregation with Monte-Carlo

Random Walk Example

Policy

50% ⟷ **50%**

$$\gamma = 1$$

R = +1

State Aggregation with Monte-Carlo

State Aggregation

State	Value
s_1	3
s_2	3
s_3	3
s_4	3
s_5	0
s_6	0
s_7	0
s_8	0

$$\mathbf{x}(s) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \hat{v}(s, \mathbf{w}) = w_1$$

$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \hat{v}(s, \mathbf{w}) = w_2$$

State Aggregation with Monte-Carlo

How to Compute the Gradient for Monte Carlo with State Aggregation

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

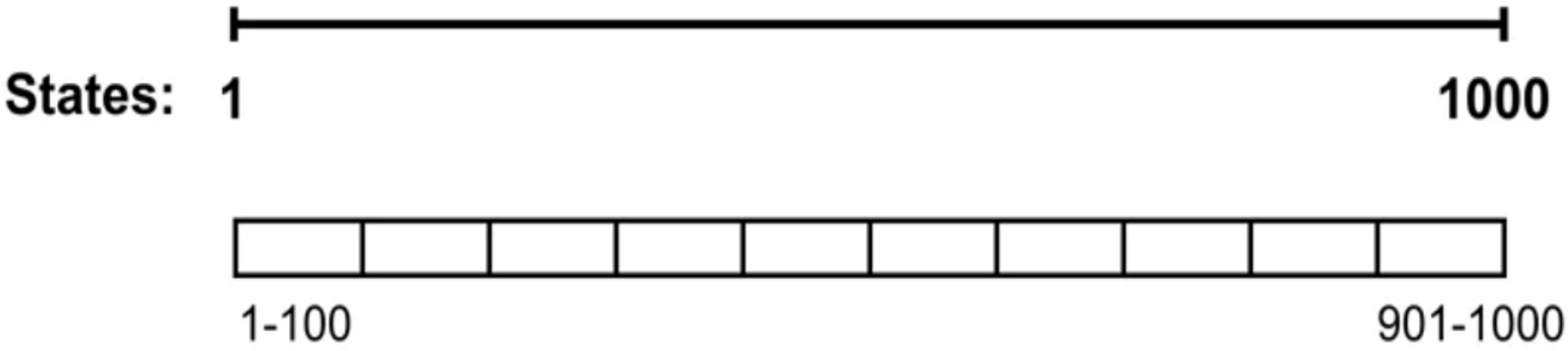
$$w_i \leftarrow w_i + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]0$$

$$w_j \leftarrow w_j + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]1$$

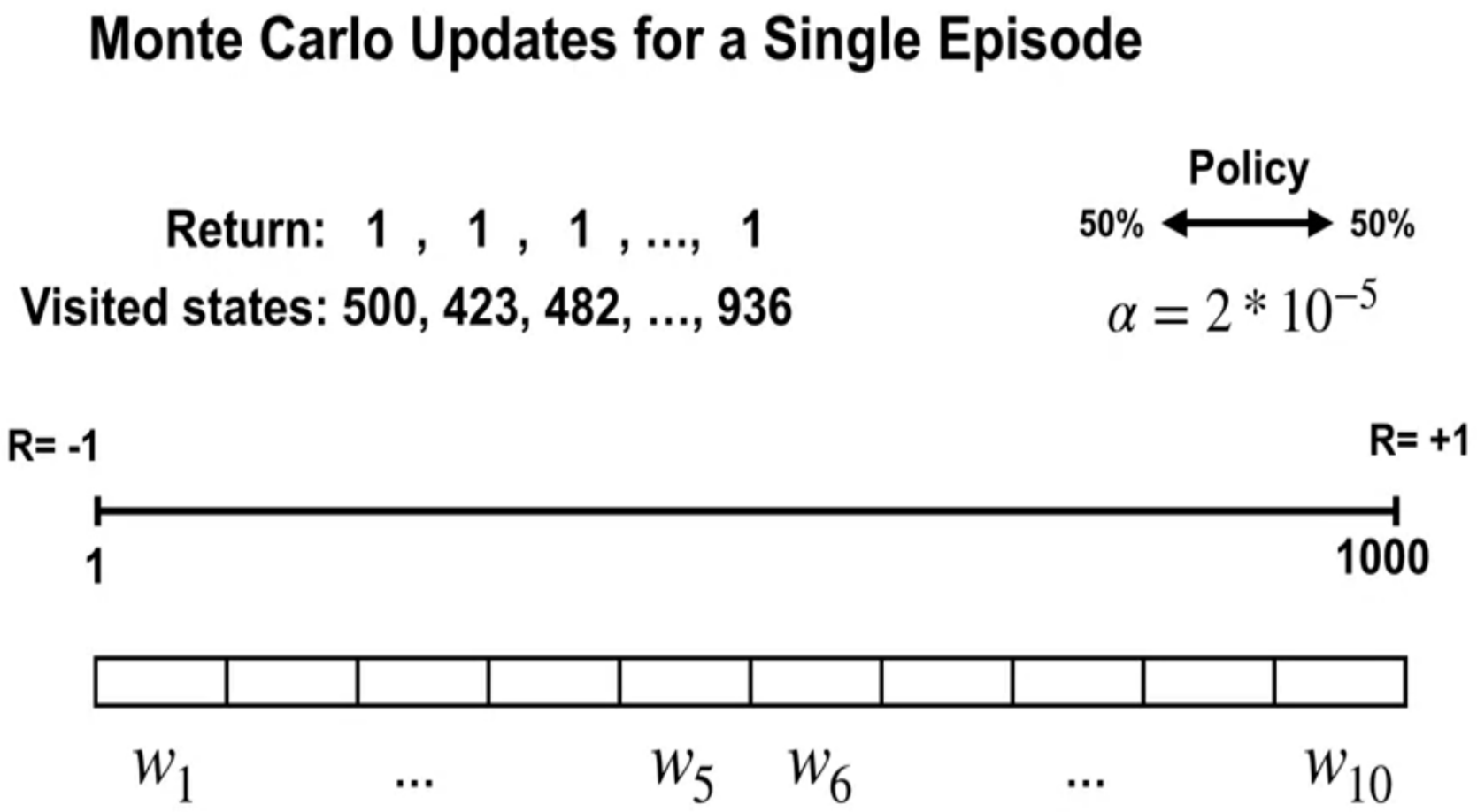
$$\nabla \hat{v}(S_t, \mathbf{w}) = \mathbf{x}(S_t) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

State Aggregation with Monte-Carlo

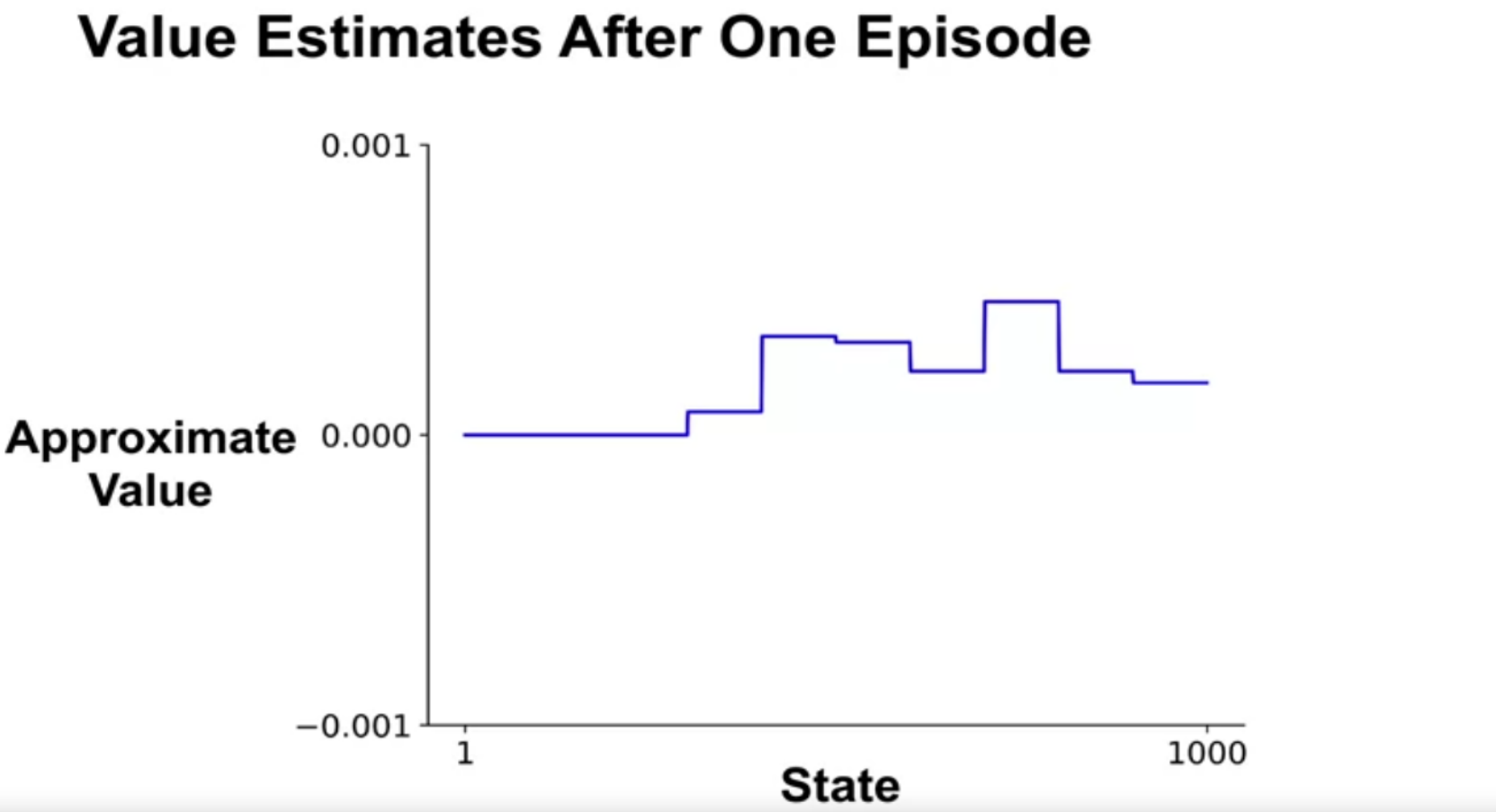
Constructing a State Aggregation for the Random Walk



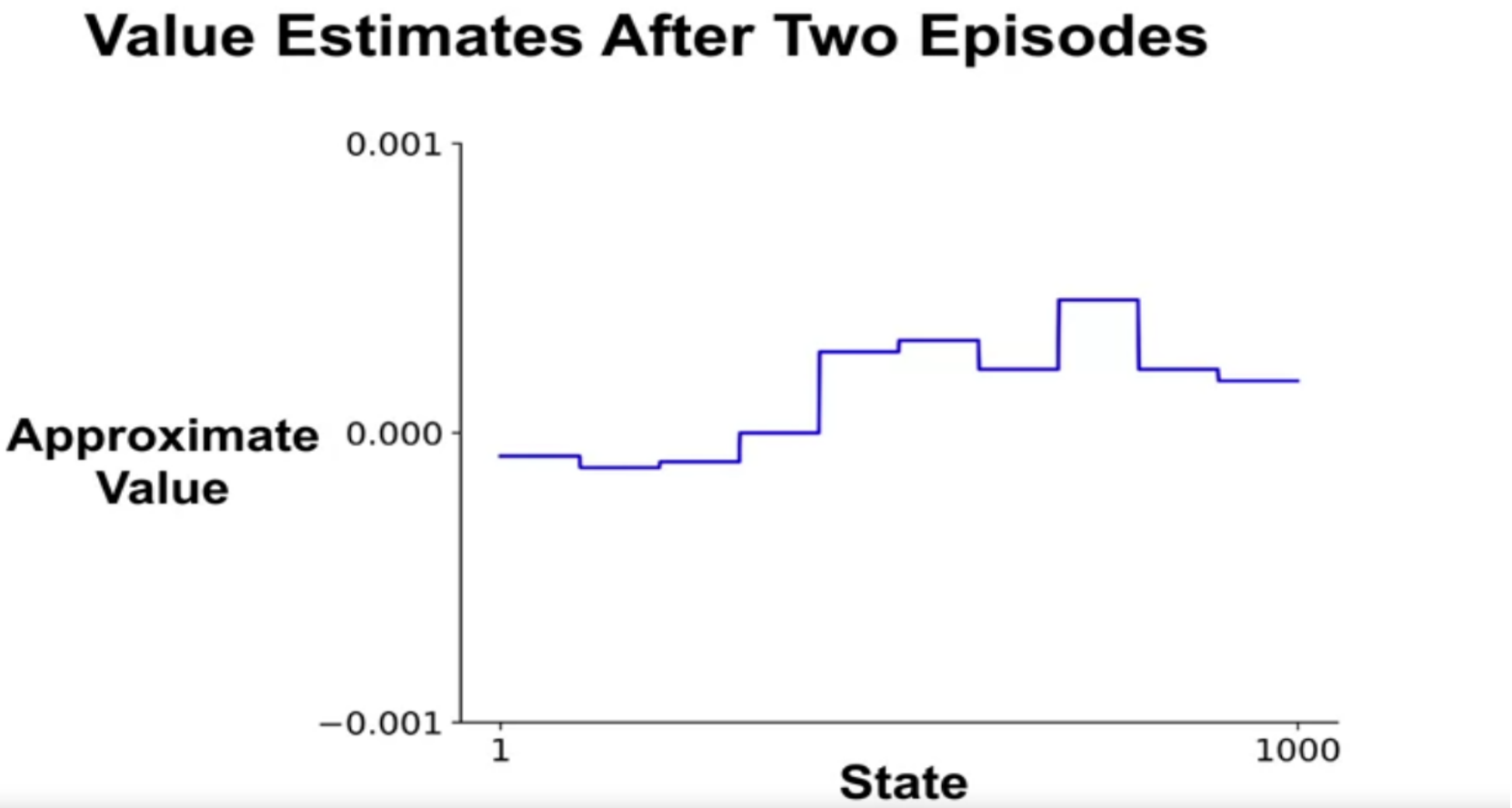
State Aggregation with Monte-Carlo



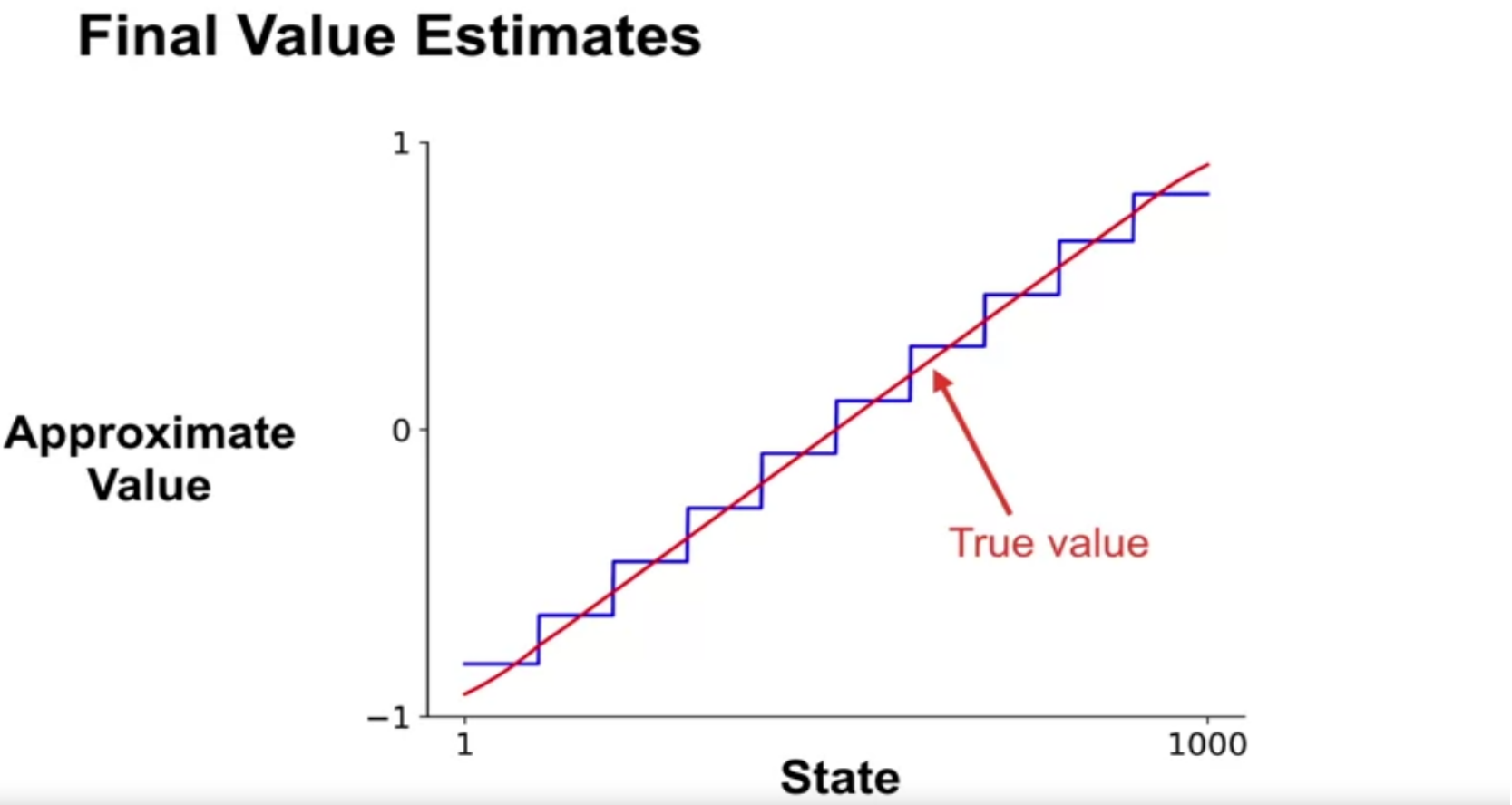
State Aggregation with Monte-Carlo



State Aggregation with Monte-Carlo



State Aggregation with Monte-Carlo



Semi-Gradient TD

$$w \leftarrow w + \alpha[U_t - \hat{v}(S, w)] \bigtriangledown \hat{v}(S, w)$$

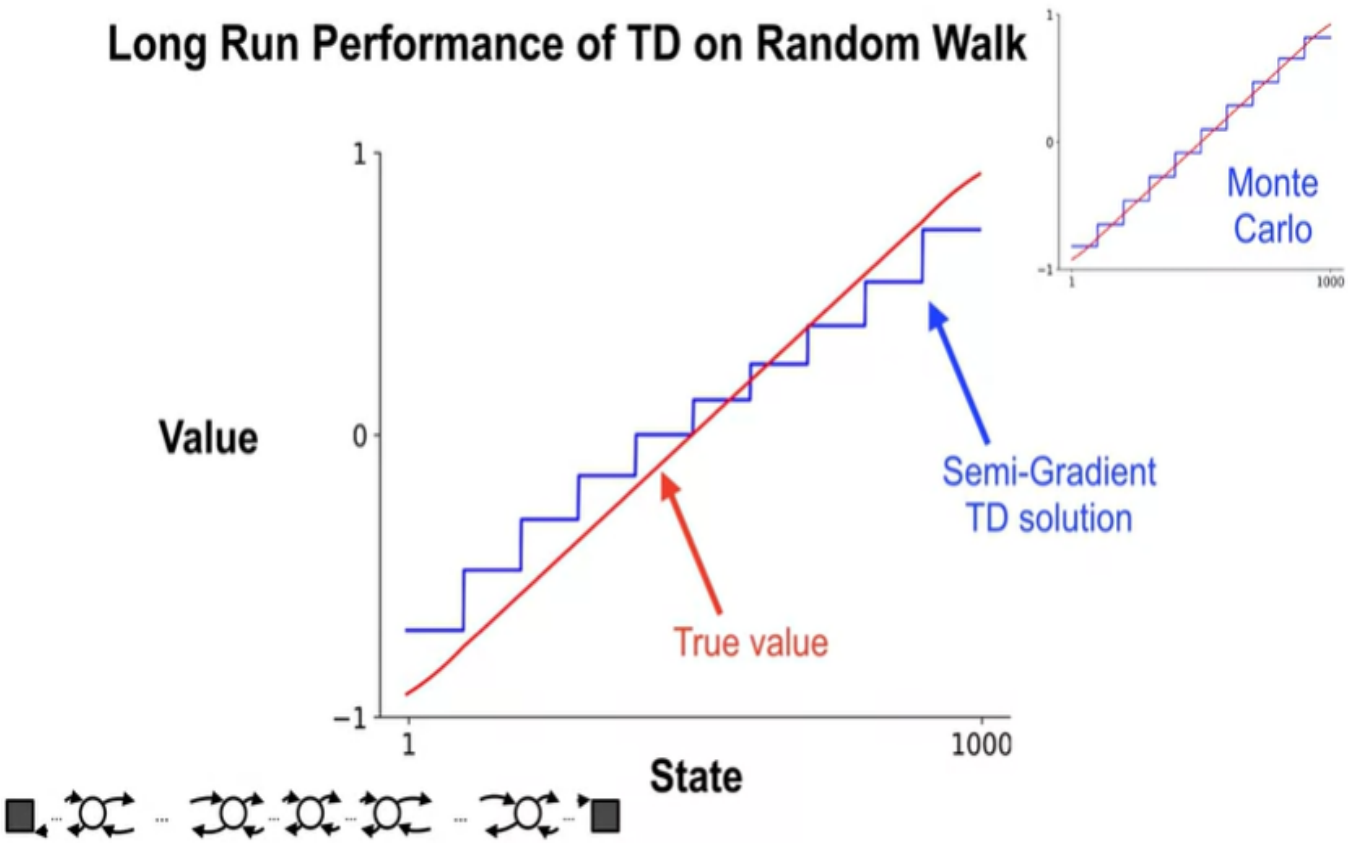
$$U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$$

Semi-Gradient TD(0) Algorithm for estimating $\hat{v} \approx v_\pi$

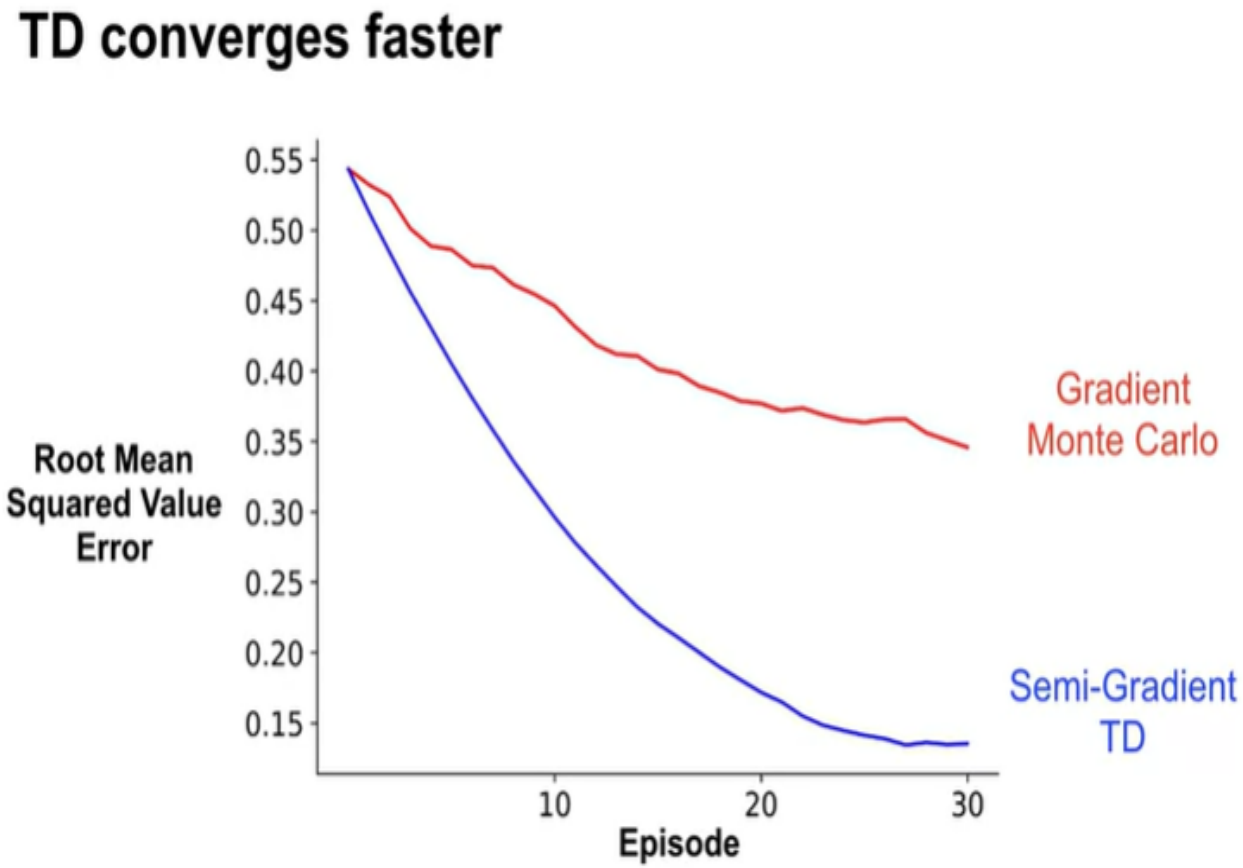
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : S^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(term, .) = 0$
Algorithmic parameter: step size $\alpha > 0$
Initialize arbitrarily value function weights $w \in \mathbb{R}^d$ (e.g. $w = 0$)

Loop for each episode:
  Initialize S
  Loop for each episode step:
    Choose $A \sim \pi(.|S)$
    Take action A, observe R, S'
    $w \leftarrow w + \alpha[R + \gamma \hat{v}(S', w) - \hat{v}(S, w)] \bigtriangledown \hat{v}(S, w)$
    $S \leftarrow S'$
  Until S is terminal

Long-run TD vs MC

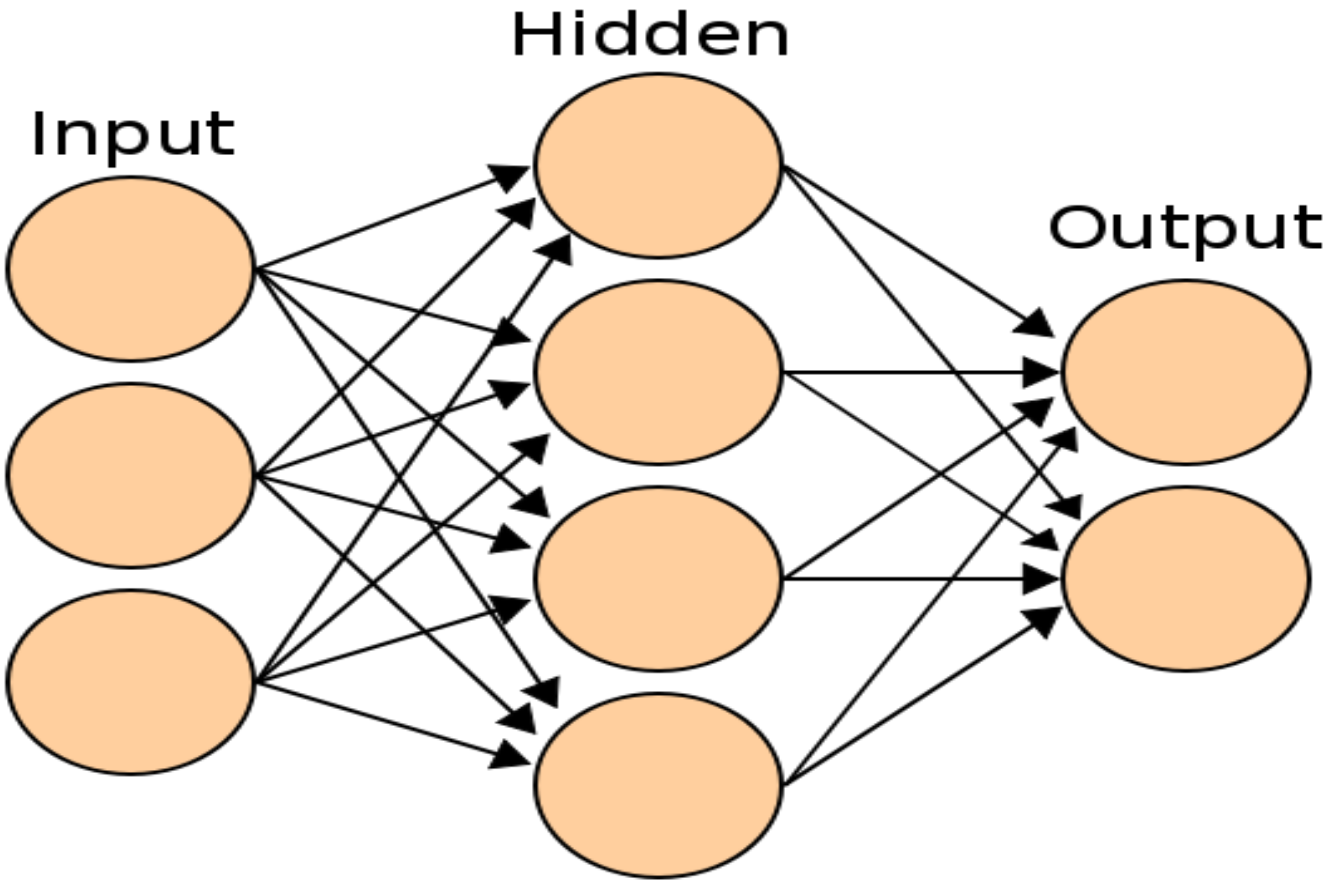


TD vs MC learning speed

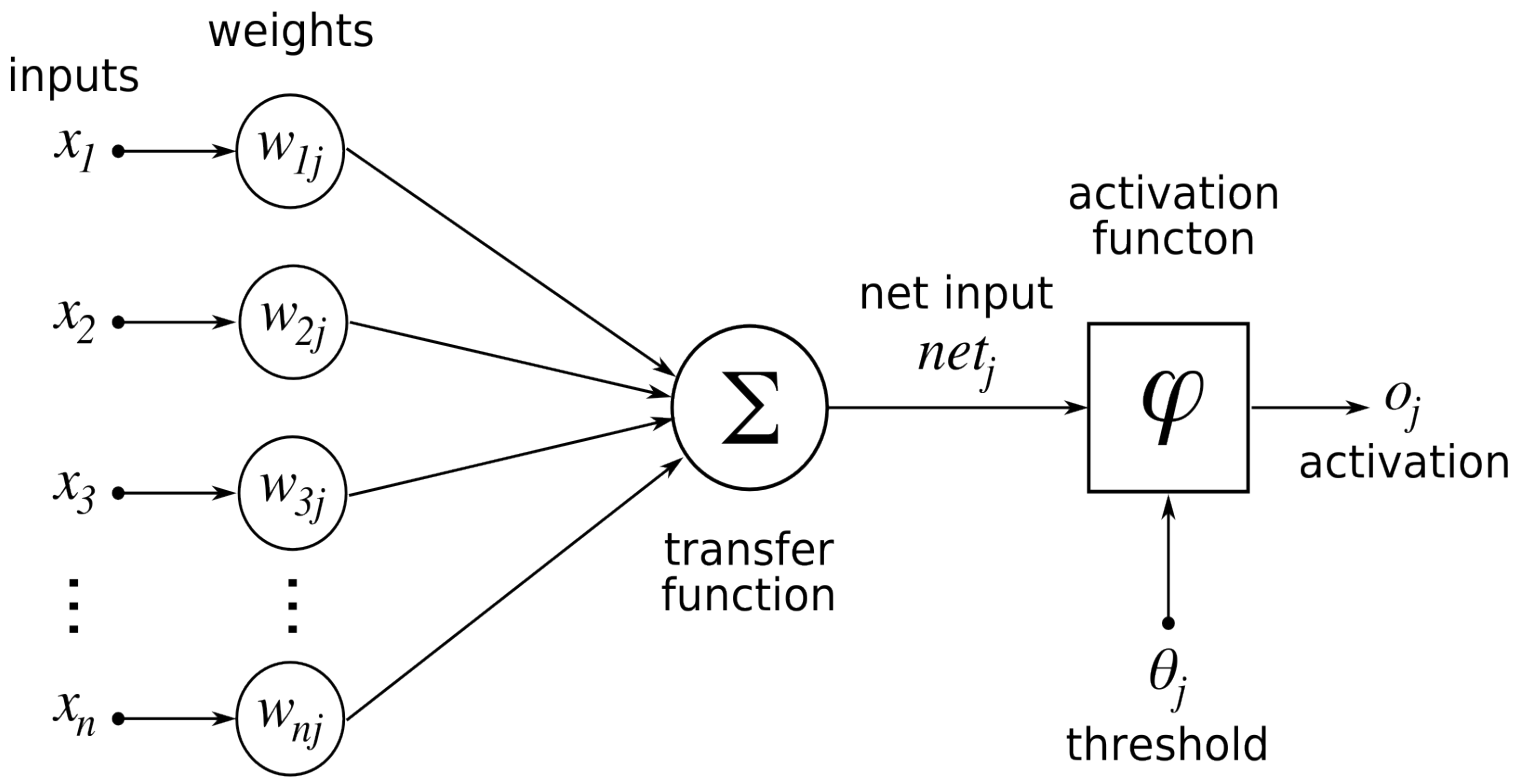


Contents

Neural Networks



Neural Networks



Training algorithm

- **Backpropagation**
- Evolutionary algorithms
- Pruning
- ...

Deep Q-Learning

- In Q-Learning, we stored every possible state-action value
- If the environment is too big, it becomes impossible in practice

Deep Q-Learning

- In Q-Learning, we stored every possible state-action value
- If the environment is too big, it becomes impossible in practice
- In Deep Q-Learning, we want to use a neural network to approximate the Q-value of all possible action

Deep Q-Learning

- In Q-Learning, we stored every possible state-action value
- If the environment is too big, it becomes impossible in practice
- In Deep Q-Learning, we want to use a neural network to approximate the Q-value of all possible action
- In a Deep Q-Network, state will be the input
- Action will be the output

Deep Q-Learning