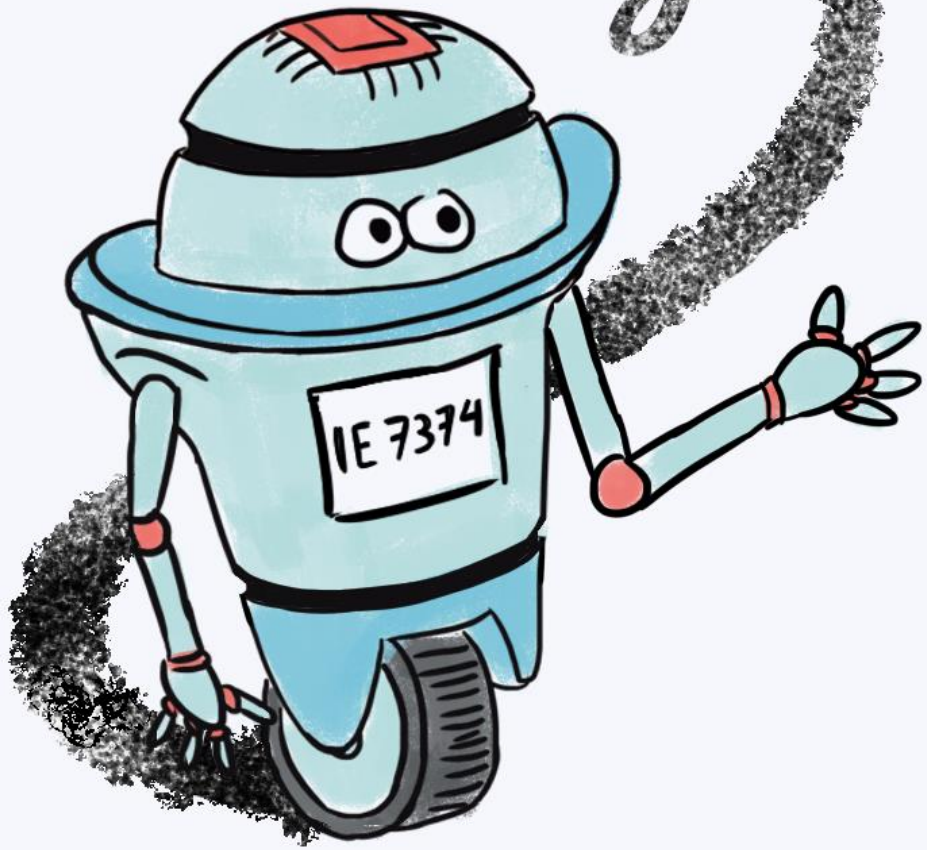
Reinforcement Learning



On-policy Prediction with Approximation (Part a)

Mohammad **Dehghani**

Mechanical and Industrial Engineering Department

Northeastern University

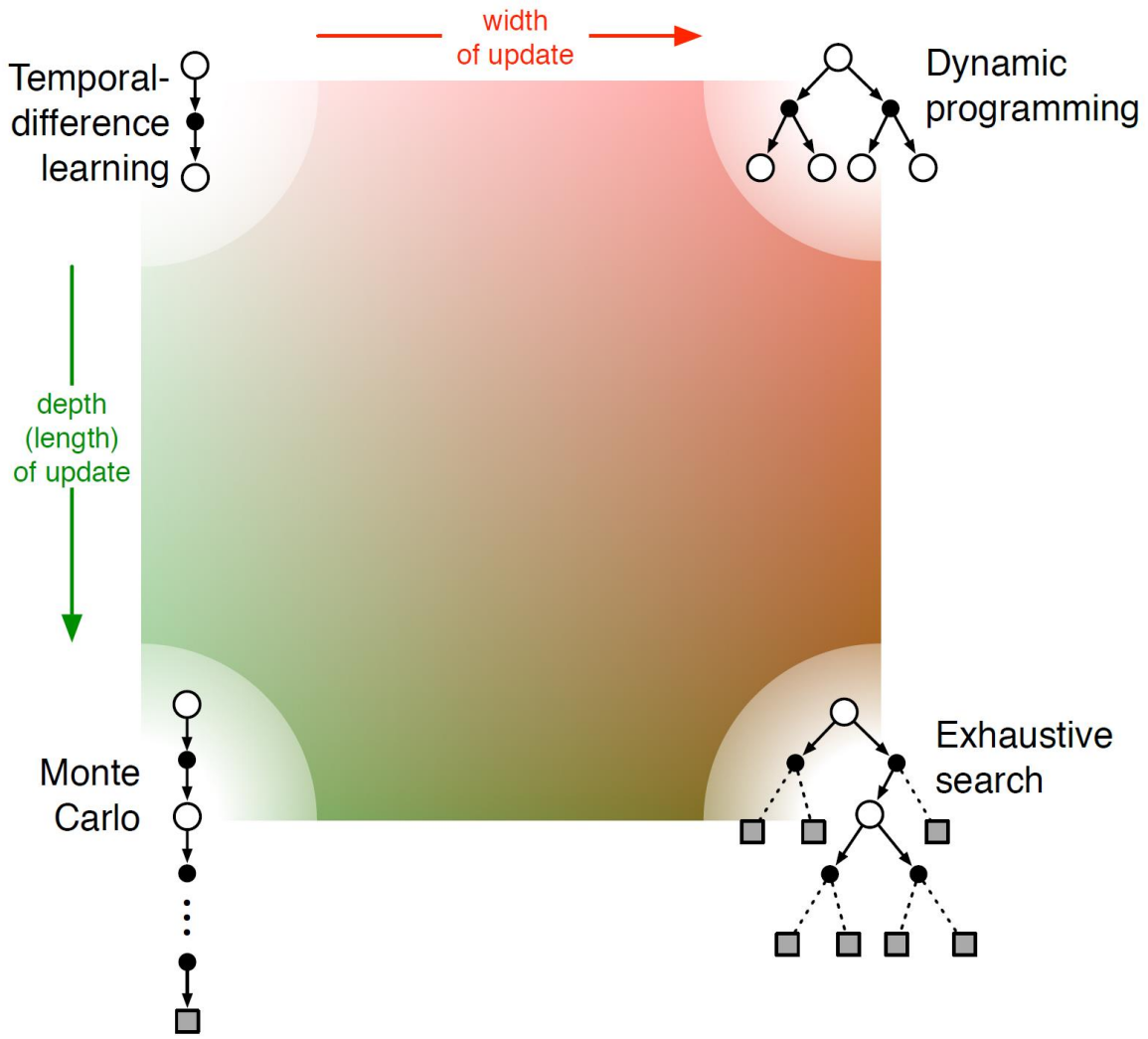
Apr-20

Introduction

Function approximation journey

Introduction

Overview of RL Methods



Introduction

Function approximation in reinforcement learning

- **So far we have represented value function by a lookup table**
  - Tabular representation
    - Every state s has an entry $V(s)$, or
    - Every state-action pair s; a has an entry $Q(s, a)$

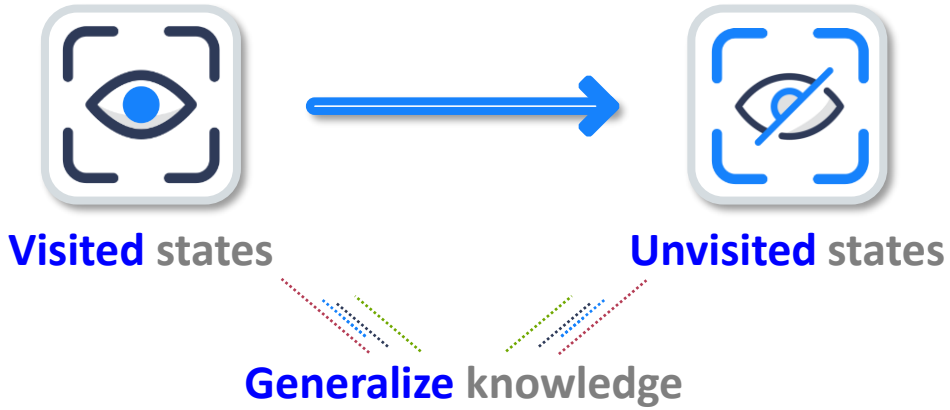
- **In many of real world problems, the state space is combinatorial and enormous;**
  - is much larger than the number of atoms in the universe.
    - **Backgammon:** 10^{20} states
    - **Computer Go:** 10^{170} states
    - **Helicopter:** continuous state space

Introduction

Generalization

- **Problem with large MDPs:**
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually.
    - Tabular methods become inefficient to store the values for all states in a table
    - In fact, we can't even guarantee we will see the same state more than once.

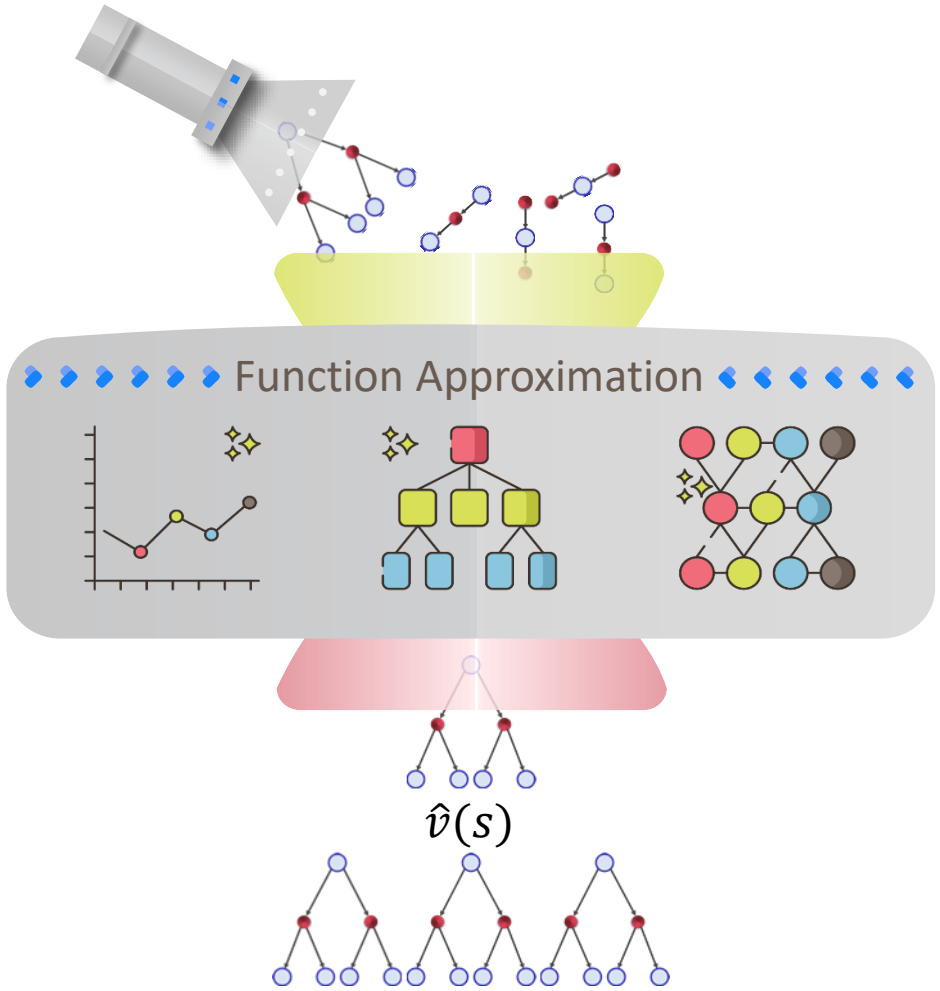
- **Solution for large MDPs:**
  - Generalize from seen states to unseen states



Introduction

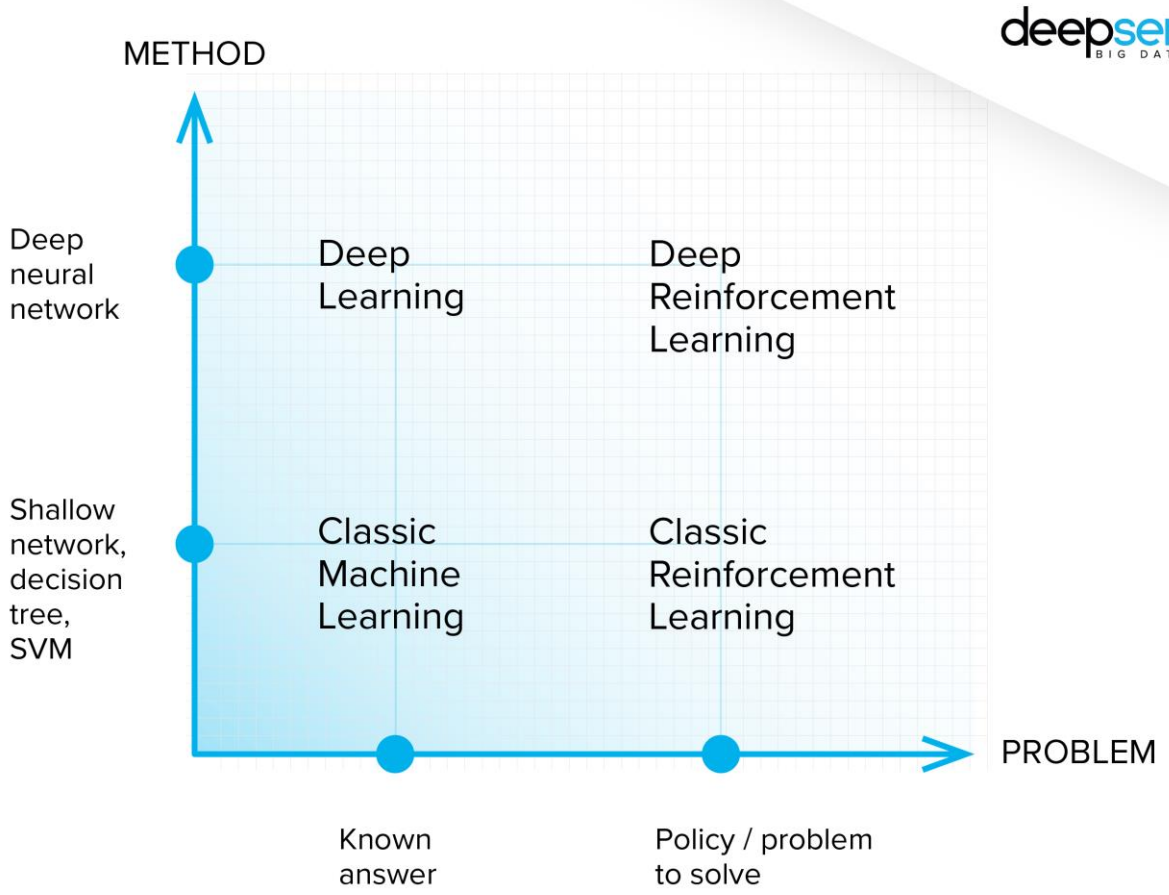
Function approximation setting

- **Goal:** Approximation of the entire function
- ***Function approximation* is an instance of *supervised learning***
  - **Statistical curve fitting**
  - **Decision tree**
  - **Artificial neural networks**

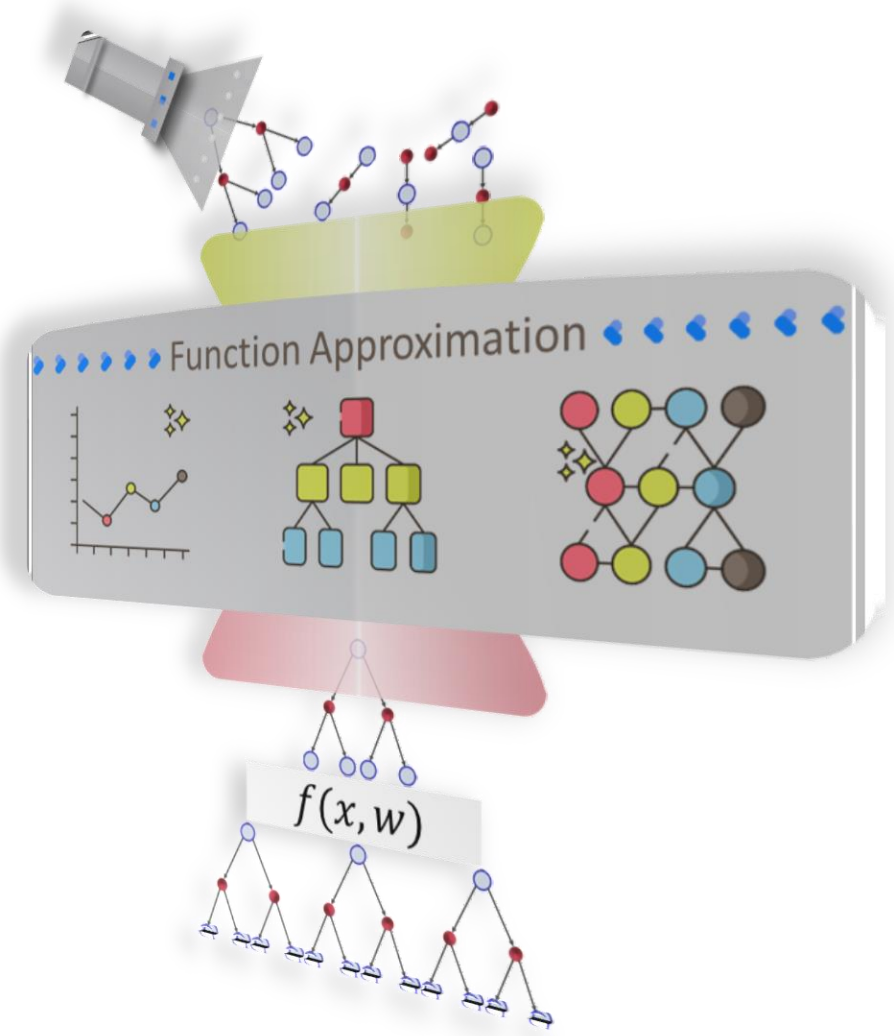


Introduction

Outline



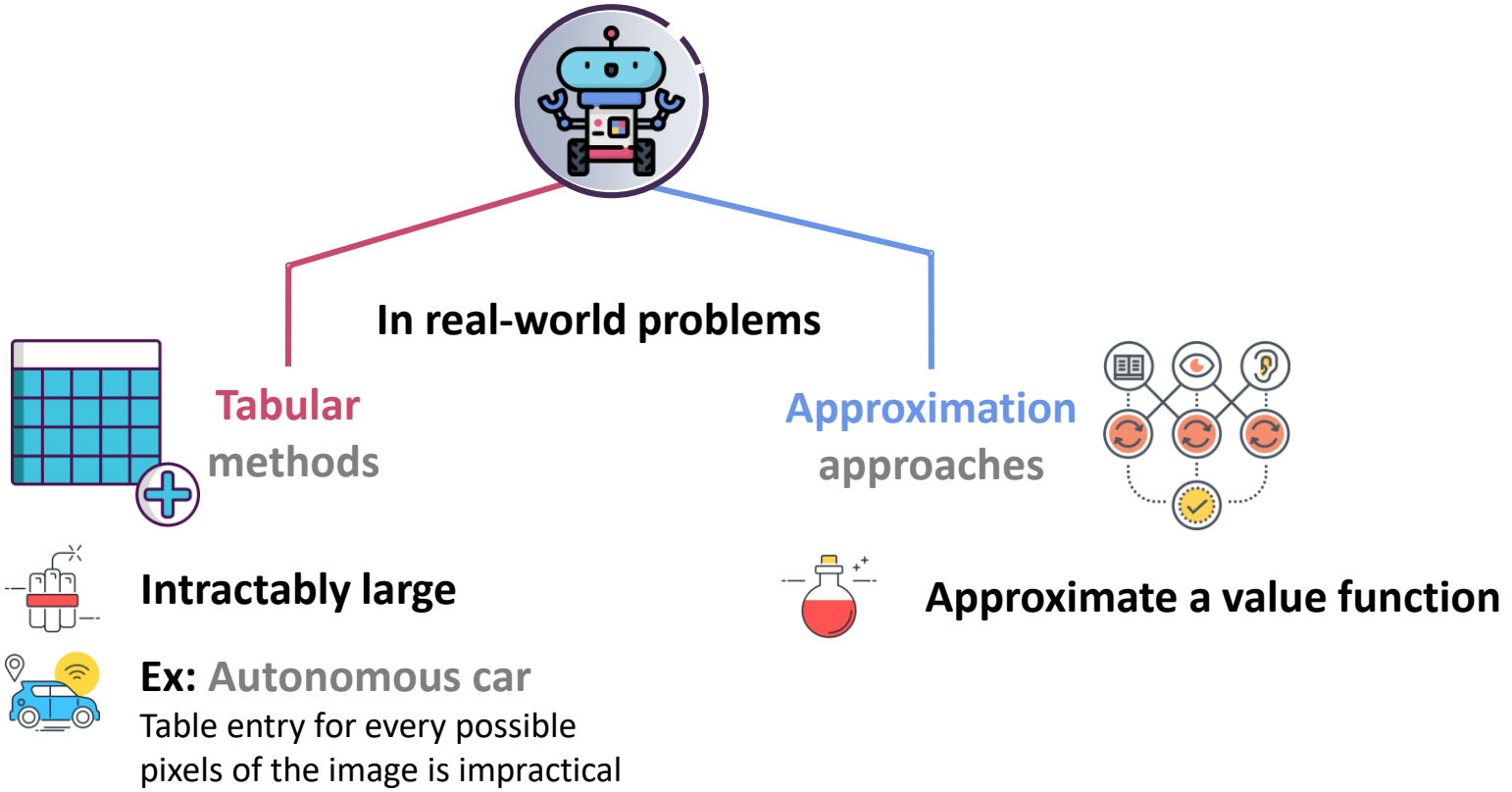
Value Function Approximation



Value Function Approximation

Tabular methods vs. Approximation approaches

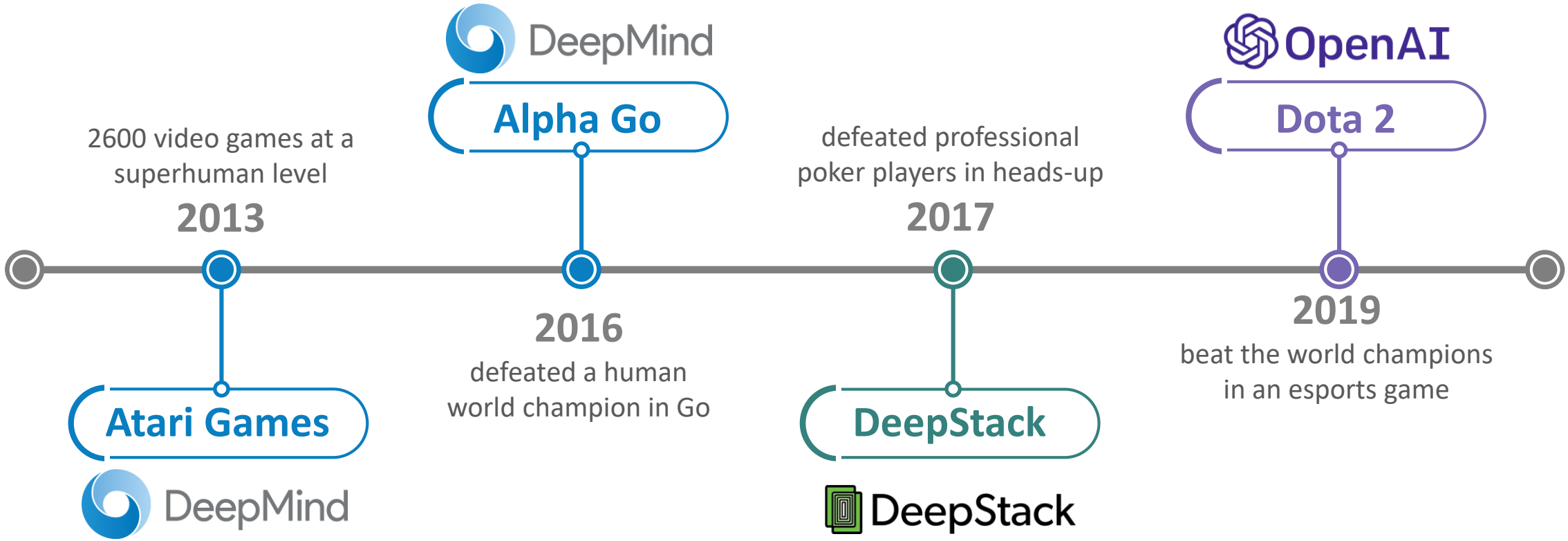
- **Tabular approaches are inherently limited to *fairly low-dimensional* problems and suffer from *lack of scalability***



Value Function Approximation

Trendline in games and applications

- **Tabular case is a special case of linear value function approximation**



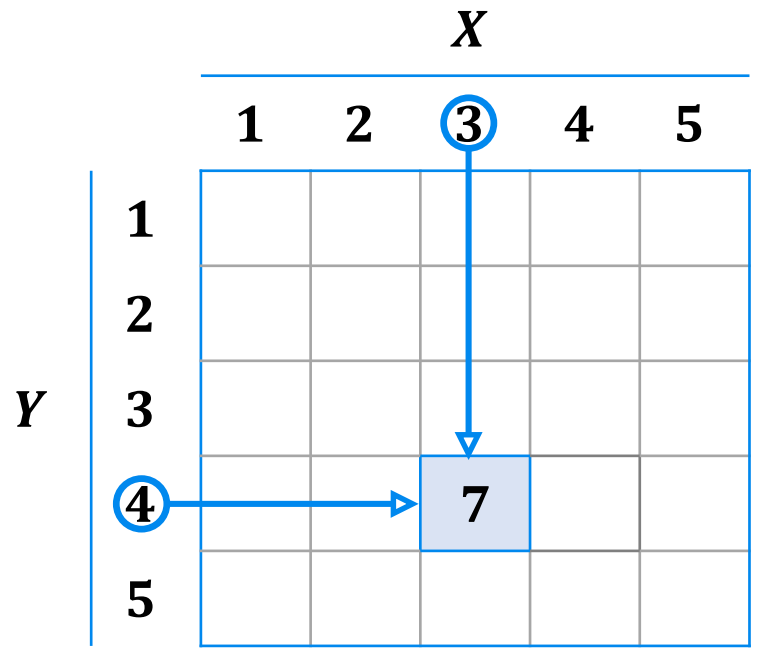
Value Function Approximation

Value function representation

Tabular representation

State	Value
S_1	20
S_2	0
S_3	−10
S_4	8
...	...
S_{25}	6

Function representation

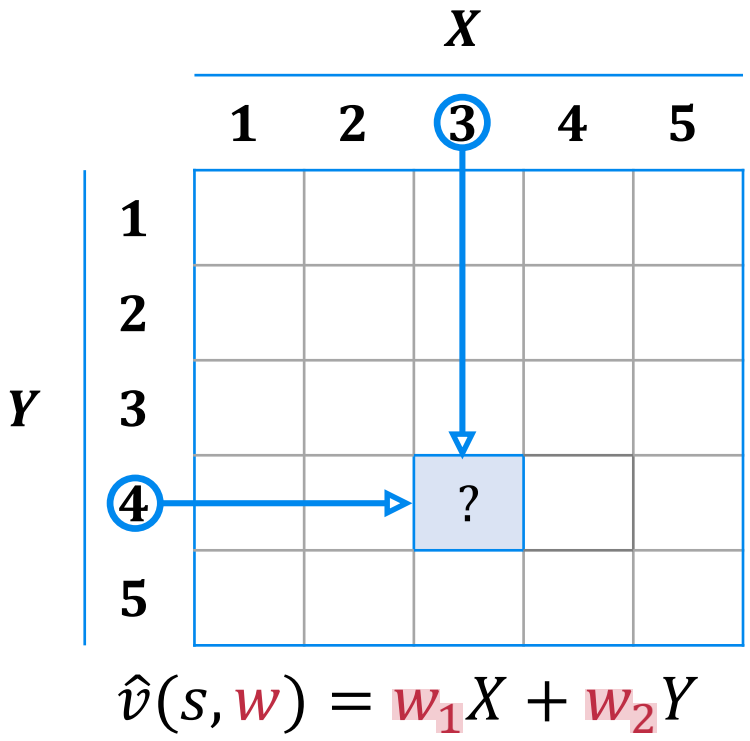


$$\hat{v}(s) = X + Y$$

Value Function Approximation

Parametrizing the Value Function

Function representation

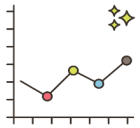


$$\hat{v}(s, w) = w_1 X + w_2 Y$$

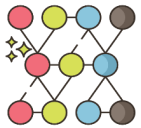
approximation

$$\hat{v}(s, w) \approx v_\pi(s)$$

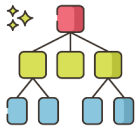
weights



Linear function approximation



Multi-layer artificial neural network



Decision tree

Value Function Approximation

Characteristics

$$\hat{v}(s, w) \approx v_\pi(s)$$

- **Typically, the number of weights** (the dimensionality of w) **is much less than the number of states** $(d \ll |\mathcal{S}|)$
- **Changing one weight changes the estimated value of many states.**
  - Such ***generalization*** makes the learning potentially more powerful but also potentially more difficult to manage and understand.
- **Applicable to partially observable problems**

Value Function Approximation

Advantages and challenges

- **Advantages:**
  - **Flexible for high dimensional problems:**
    - Parameterized value function is flexible enough to represent every function mapping to overcome the high dimensional problems (possibly continuous space - $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$)
  - **Faster:**
    - Allows the model to learn in reasonable time and space
  - **Less memory requirements:**
    - Instead of recording all state (or state-action) values in a tabular format, function approximation provides estimation using weights.
    - Weights parametrize the function, and the only values that needs to be stored.

Tabular representation

State	Value
S_1	20
S_2	0
S_3	−10
S_4	8
...	...
S_{25}	6

- **Challenges**
  - **Setting weights**
    - Our learning outcomes will modify the weights, instead of the individual state values

Function representation

$$\hat{v}(s, w) = w_1 X + w_2 Y$$

Value Function Approximation

Linear Function Approximation

Definition

Linear Function Approximation is a linear combination of features in order to approximate the value function.

- **Linear combination means that the parameters are linearly combined.**

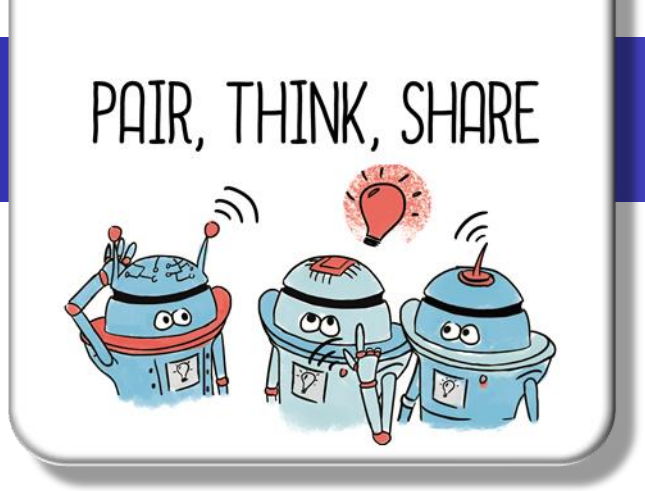
weights

Features

$$\hat{v}(s, w) \doteq \sum w_i \, x_i(s)$$

$$= \langle w, x(s) \rangle$$

- The features can be ***the position of a robot***, **angular position** and ***speed of an inverted pendulum***, ***configurations of the stones*** in a Go game, etc.



Tabular value functions vs Linear functions

- In the Tabular methods, state and action spaces are small enough for the approximate value functions to be represented as arrays, or tables.
- Function approximation methods such as linear functions, takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation of the entire function.

Question

- **Can we consider tabular case as a *special case* of linear value function approximation?**

Tabular representation

State	Value
s_1	w_1
s_2	w_2
⋮	⋮
s_{10}	w_{10}
⋮	⋮
s_{25}	w_{25}

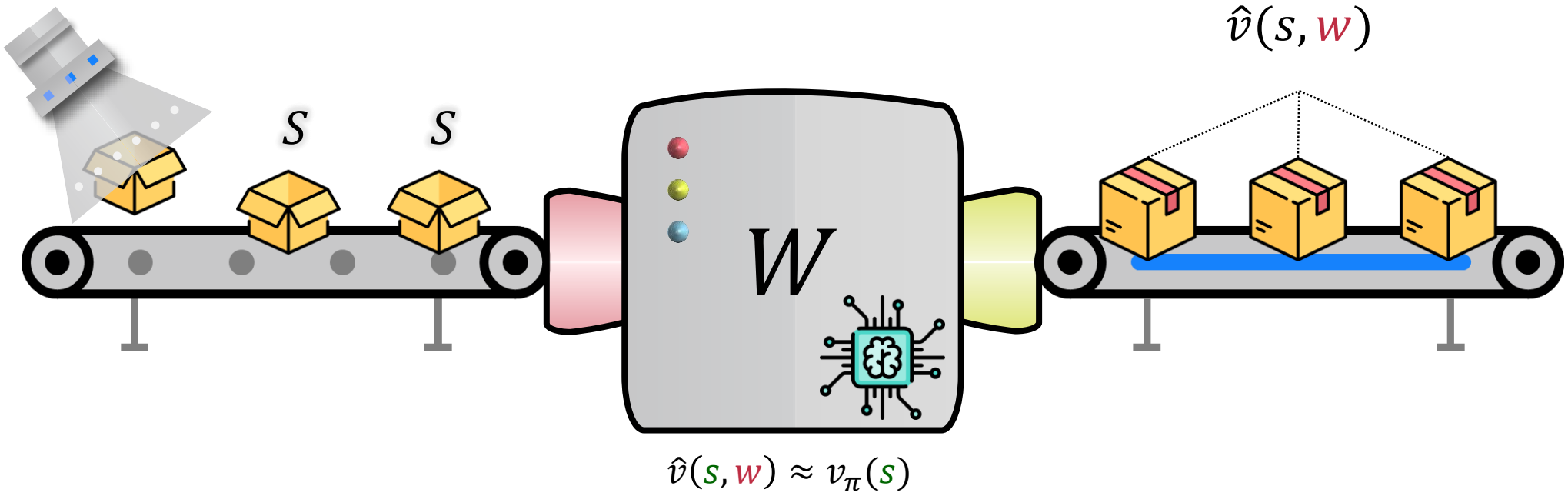
$$x(s_i) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Function representation

$$\hat{v}(s_i, w) \doteq \langle \boldsymbol{w}, x(s_i) \rangle = w_i$$

Value Function Approximation

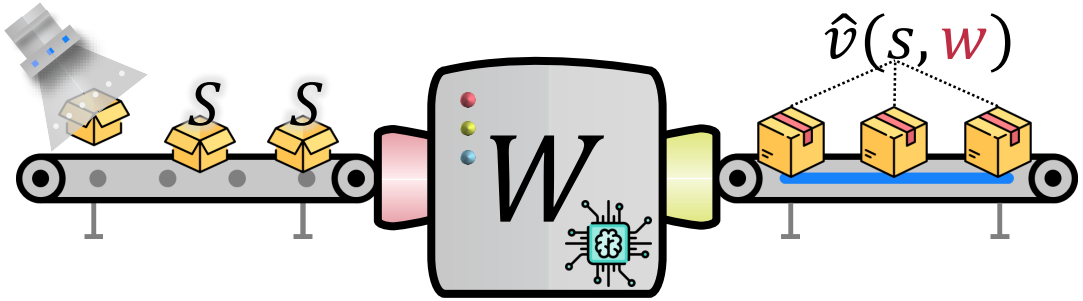
Function Approximation methods: General setting



Value Function Approximation

Applicability of supervised Learning methods

- **Estimating a function from samples is addressed by the *supervised learning* paradigm**



RL Updates

Monte Carlo

$S_t \mapsto G_t$

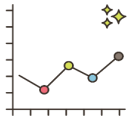
TD(0)

$S_t \mapsto R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t)$

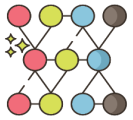
DP

$s \mapsto \mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) | S_t = s]$

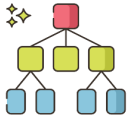
Function Approximation



Statistical curve fitting



Artificial neural networks

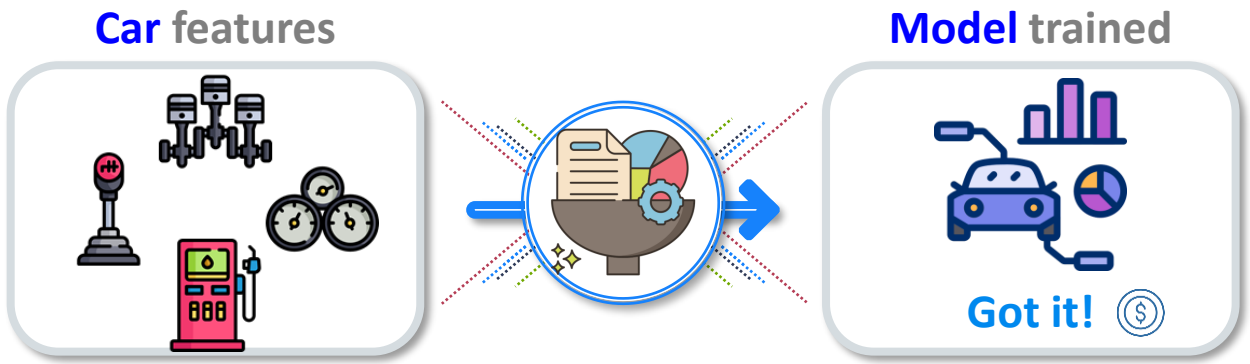


Decision tree

Value Function Approximation

Challenges with using supervised Learning methods (1)

- **Estimating a function from samples is addressed by the *supervised learning* paradigm**
  - **Ex:** Car price estimation



- **In reinforcement learning, the (state-action) values are *not* directly observed**
  - Indeed, values are never directly observed, just rewards which define them.
  - This renders the underlying estimation problem more difficult.

Value Function Approximation

Challenges with using supervised Learning methods (2)

- **The most sophisticated artificial neural network and statistical methods all assume a *static* training set over which multiple passes are made**
  - **Ex:** Car price estimation

Input

Odometer	Cylinder	MPG
10,000	4	28
50,000	6	22
...	...	...
73,000	4	19

,

Target

Price
$ 30,000
$ 12,000
...
$ 3,000

Model trained

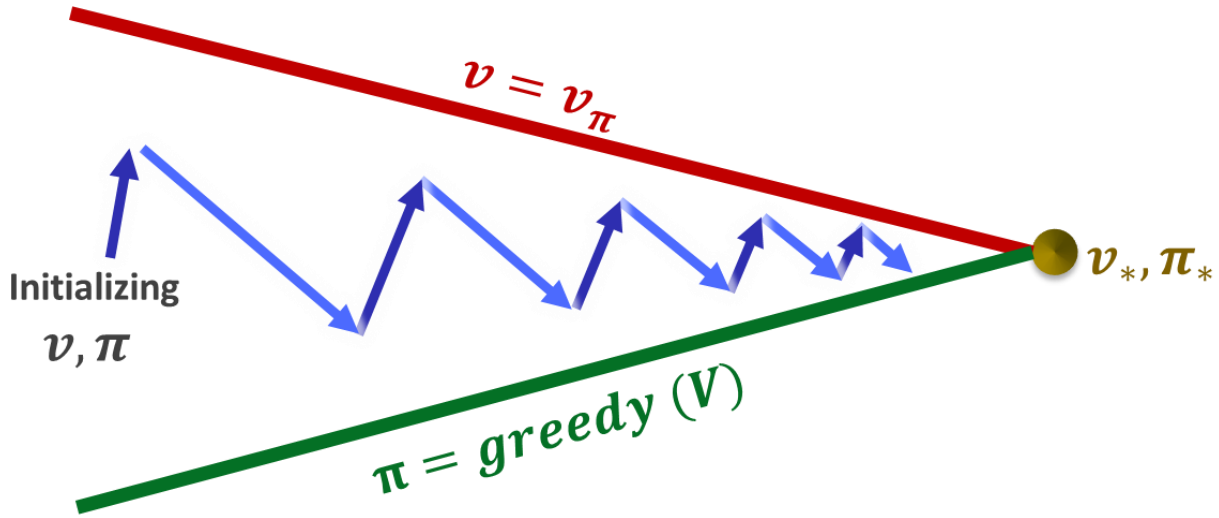
Got it!

- **In reinforcement learning, however, it is important that learning be able to occur online**
  - While the agent interacts with its environment or with a model of its environment.
- **Supervised learning methods are typically not designed for changing targets**

Value Function Approximation

Challenges with using supervised Learning methods (3)

- **Handle nonstationary:**
  - In addition, reinforcement learning generally requires function approximation methods able to handle **nonstationary** target functions (target functions that change over time)
  - **Ex:** Control methods based on GPI



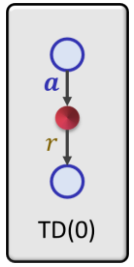
- **Methods that cannot easily handle such nonstationarity are less suitable for reinforcement learning.**

Value Function Approximation

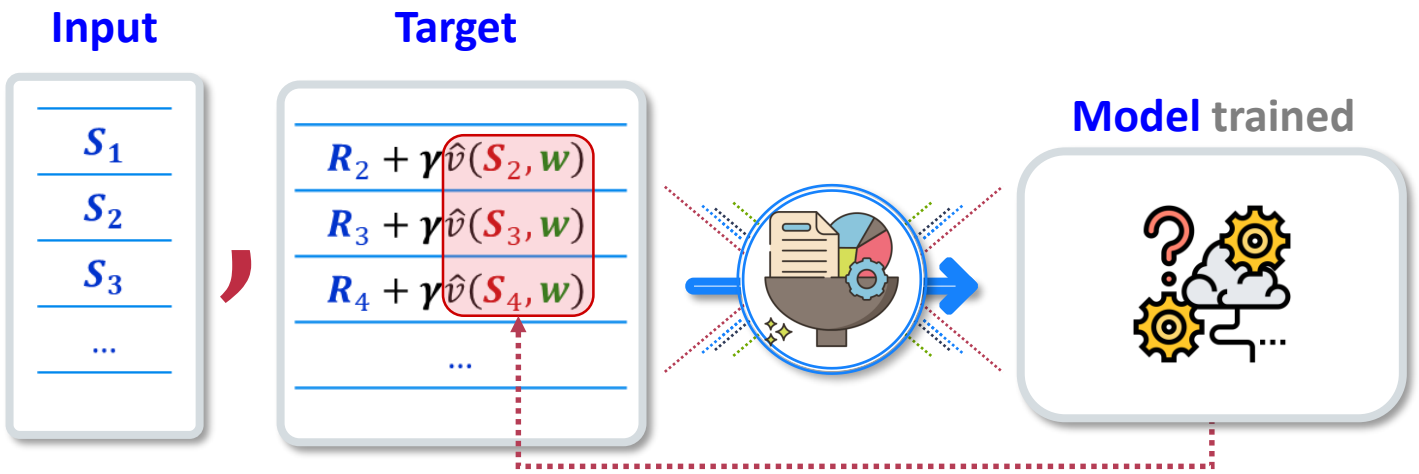
Challenges with using supervised Learning methods (4)

- **In bootstrapping methods** (i.e TD)**, predictions are used as targets during the course of learning.**
  - Agent learns state values based on trained estimates for ***subsequent*** states

$$V(S_1) \leftarrow V(S_1) + \alpha[R_2 + \gamma V(S_2) - V(S_1)]$$



- **In function approximation, the value of the *subsequent* state is also an approximation by itself.**

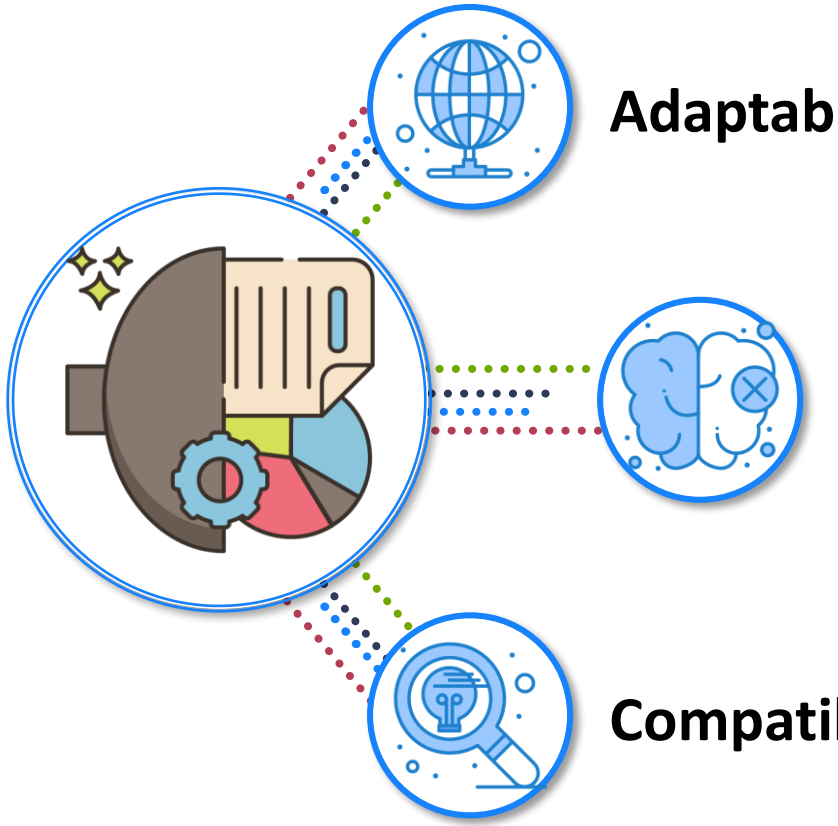


- **Supervised learning methods are typically not designed for targets computed from the agent's own estimates.**

Value Function Approximation

Summary: Appropriate supervised learnings methods for RL

- **Suitable function approximation methods for RL should be:**



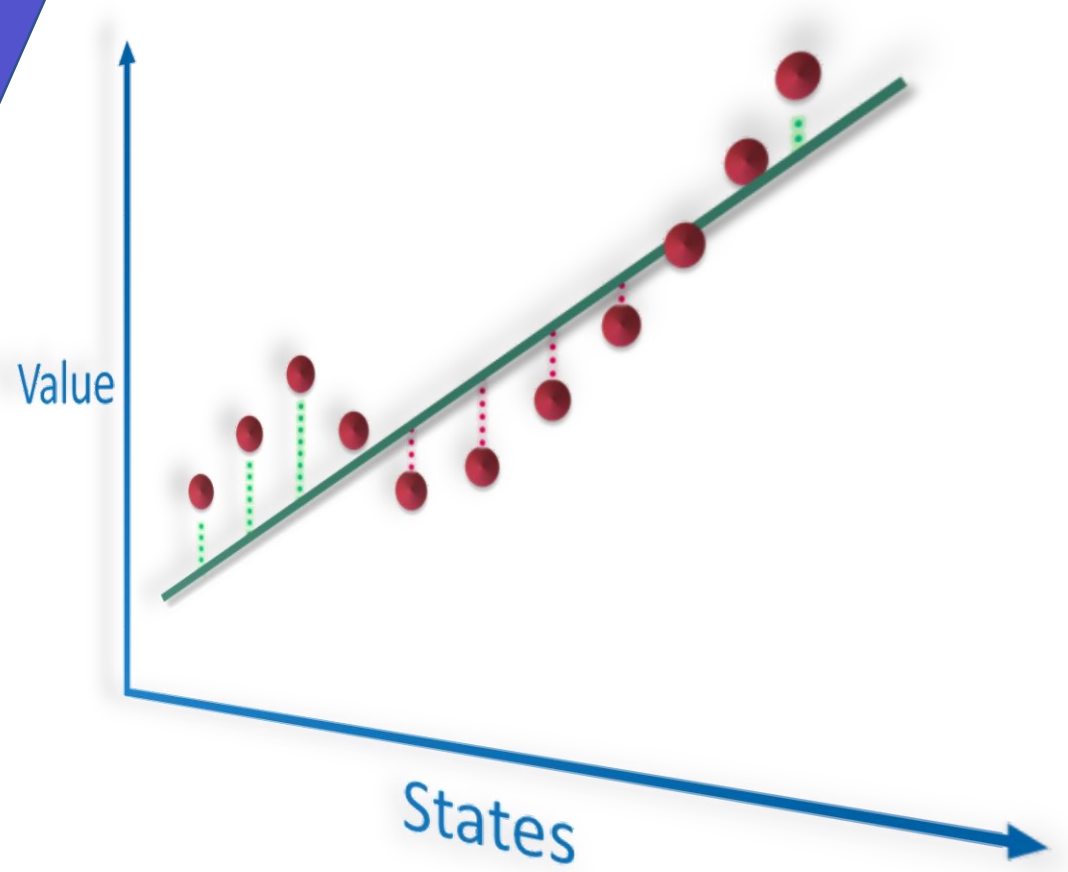
Adaptable to online learning

Capable of handling nonstationary target functions

Compatible with bootstrapping nature of some RL methods

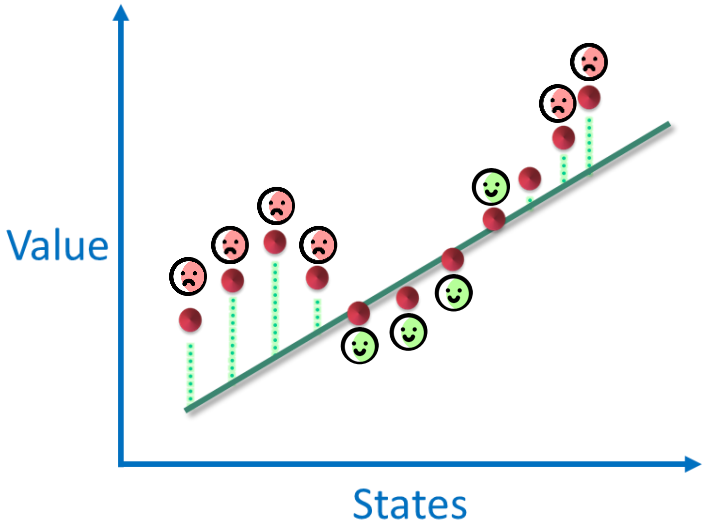
Value Error Objective

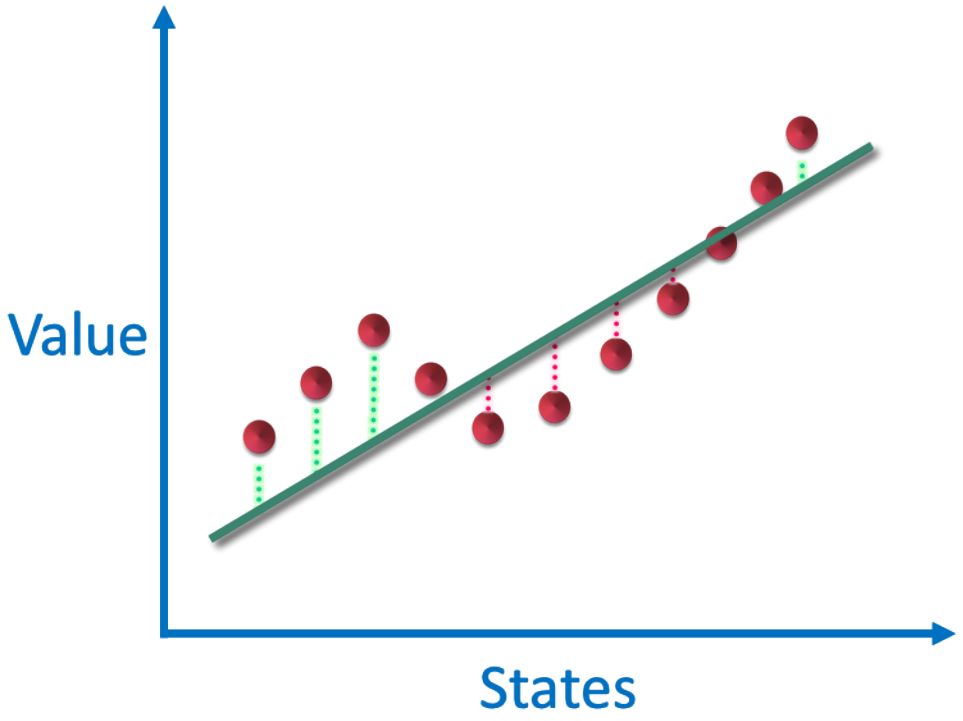
$$(\overline{VE})$$

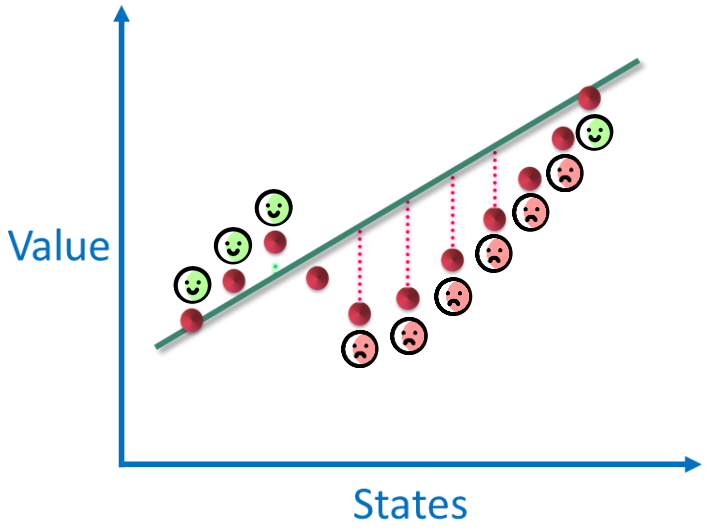


Value Error Objective

Graphical representation

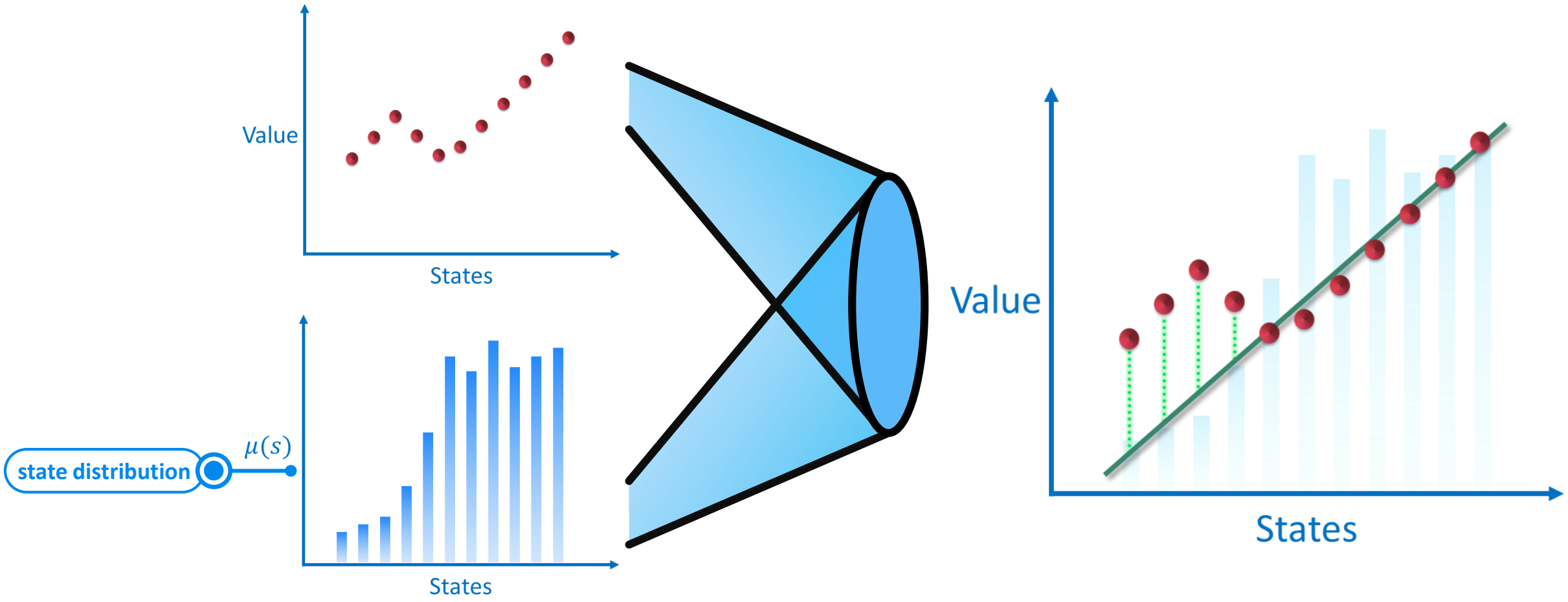






Value Error Objective

Weighting approximation function



Value Error Objective

Mean Squared Value Error

Definition

Mean Squared Value Error, denoted $\overline{VE}$:

$$\overline{VE}(w) \doteq \sum_{s \epsilon S} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$$

true value

approximate value

State dist.

- $\mu(s)$ representing how much we care about the error in each state s
  - $\mu(s) \geq 0$
  - $\sum_s \mu(s) = 1$

Value Error Objective

The on-policy distribution

Definition

- **On-policy distribution** represents how much we care about the error in each state s, and often is chosen to be the ***fraction of time spent in s***.
- In episodic tasks, *on-policy distribution* is defined as:

normalized to sum to 1

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \qquad \text{for all } s \in \mathcal{S}$$

  - $\eta(s)$ denote the number of time steps spent, on average, in state s in a single episode.

Prob. of starting from s | Transitions to s form $\bar{s}$

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_{a} \pi(a|\bar{s}) p(s|\bar{s}, a), \qquad \text{for all } s \in \mathcal{S}$$

  - $h(s)$ denote the probability that an episode begins in each state s

Value Error Objective

Is $\overline{VE}$ the right performance objective for RL?

$$\overline{VE}(w) \doteq \sum_{s \epsilon \mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$$

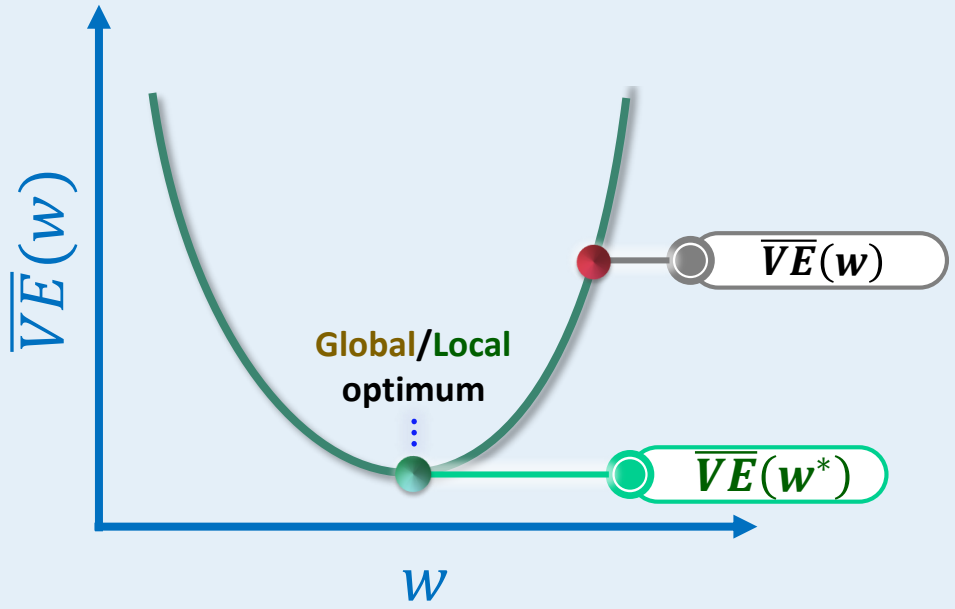
- **Ultimate purpose—the reason we are learning a value function—is to find a better policy.**
  - The best value function for this purpose is not necessarily the best for minimizing $\overline{\mathbf{VE}}$
  - Nevertheless, it is not yet clear what a more useful alternative goal for value prediction might be

- **BTW, lets focus on other apsects of $\overline{VE}$** 

Value Error Objective

Finding the global optimum

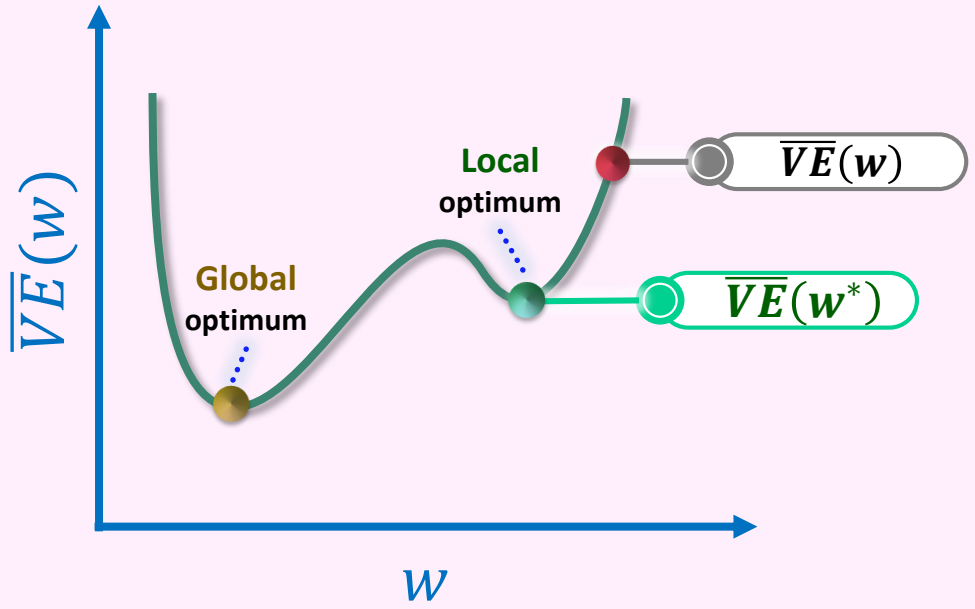
Simple function approximators



$$\overline{VE}(w^*) \leq \overline{VE}(w)$$

for all possible w

Complex function approximators

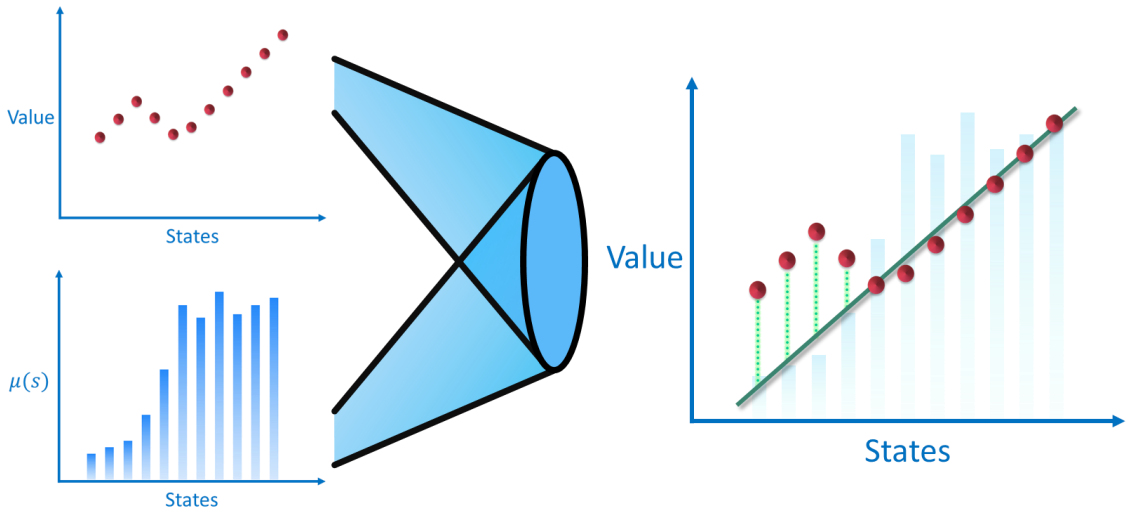


$$\overline{VE}(w^*) \leq \overline{VE}(w)$$

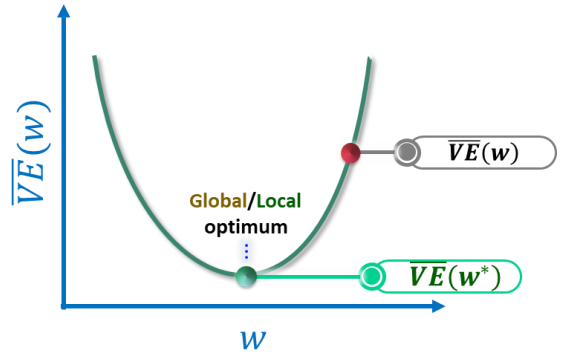
for all w in some neighborhood of w^*

Value Error Objective

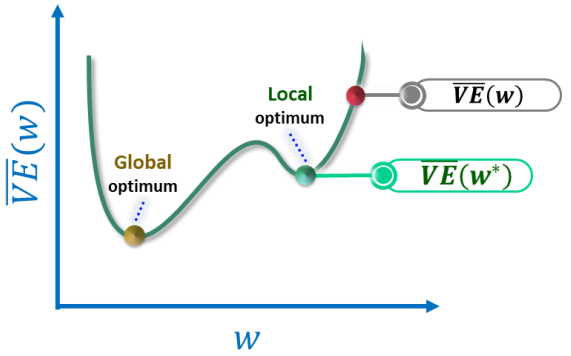
Summary



$$\overline{VE}(w) \doteq \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$$

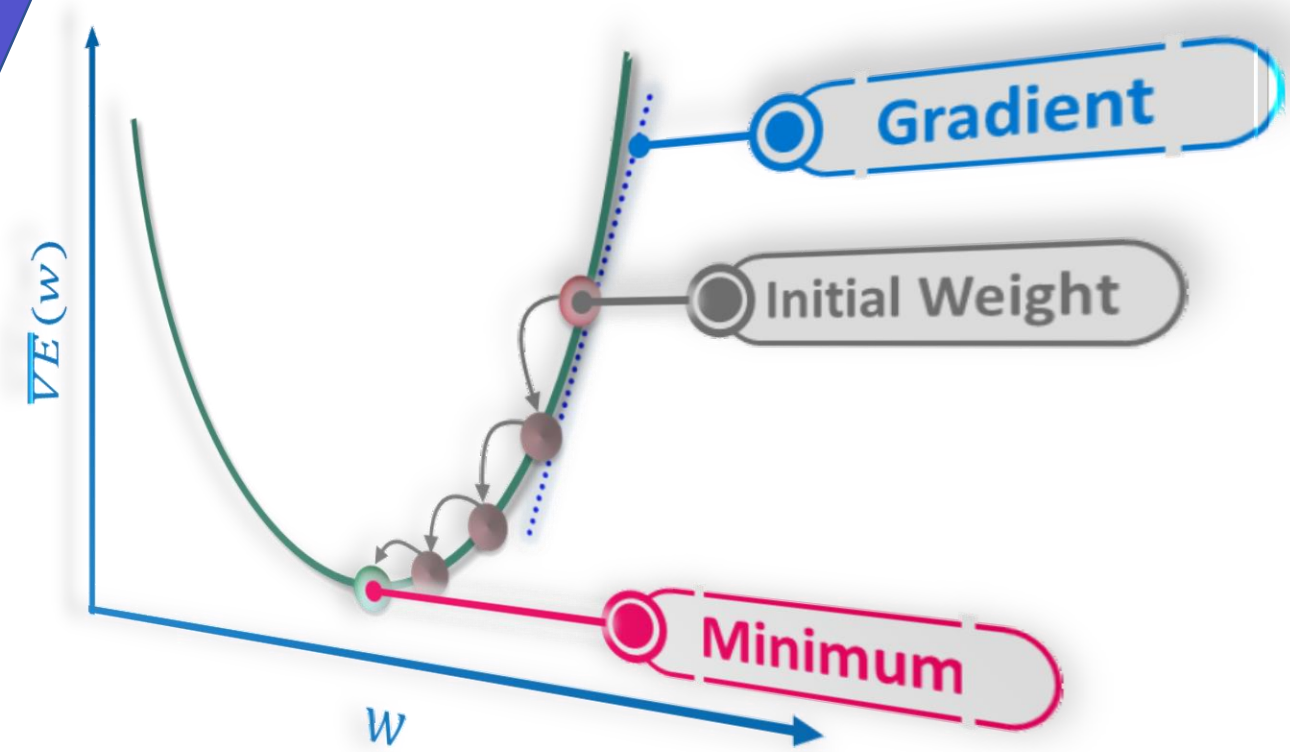


$\overline{VE}(w^*) \leq \overline{VE}(w)$
for all possible w



$\overline{VE}(w^*) \leq \overline{VE}(w)$
for all w in some neighborhood of w^*

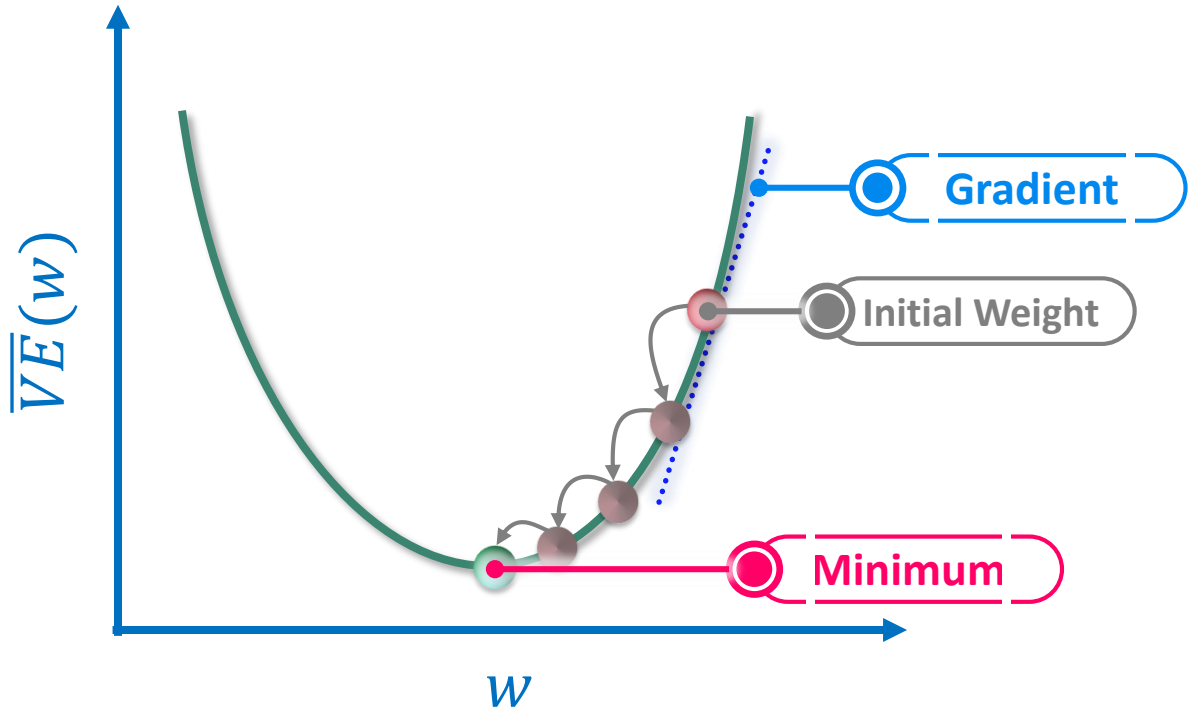
Gradient Descent (GD)



Gradient Descent

Graphical representation

- **Stochastic gradient descent (SGD) is one of the most popular and used optimizers in Data Science.**



- **SGD methods are among the most widely used of all function approximation methods**
- **SGDs are particularly well suited to online reinforcement learning.**

Gradient Descent

Mathematical representation

$$\overline{VE}(w) \doteq \sum_{s \epsilon \mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$$

$$w \doteq (w_1, w_2, \ldots, w_d)^T$$

- **How to apply gradient decent**

$$w \doteq (w_1, w_2, \ldots, w_d)^T$$

$$\downarrow$$

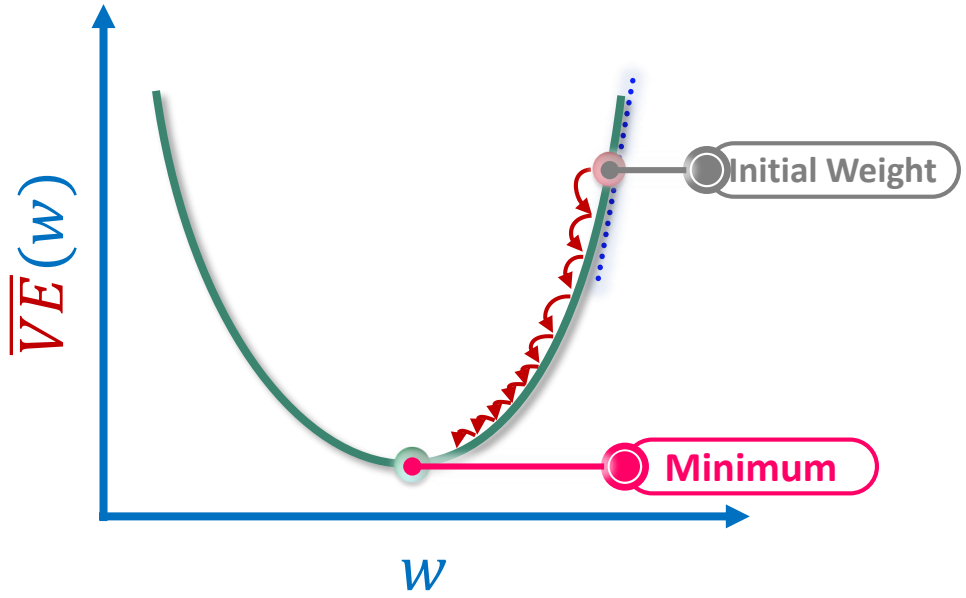
$$\nabla f(w) \doteq \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_d}\right)^T$$

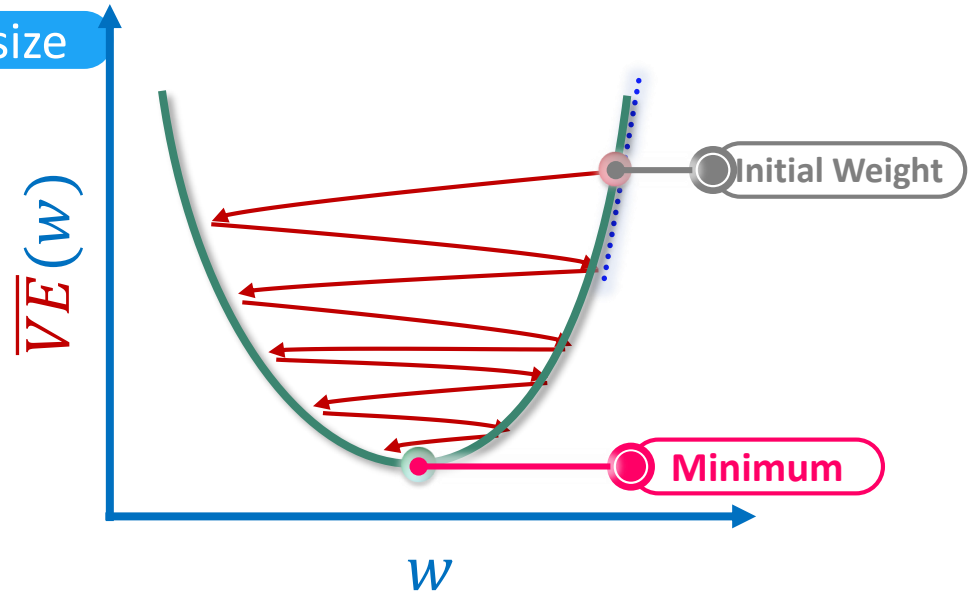
Gradient Descent

Importance of step size

$$w_{t+1} \doteq w_t + \alpha \nabla \overline{VE}(w_t)$$

Step size





Small step size:

- gradient decent can be slow

Large step size:

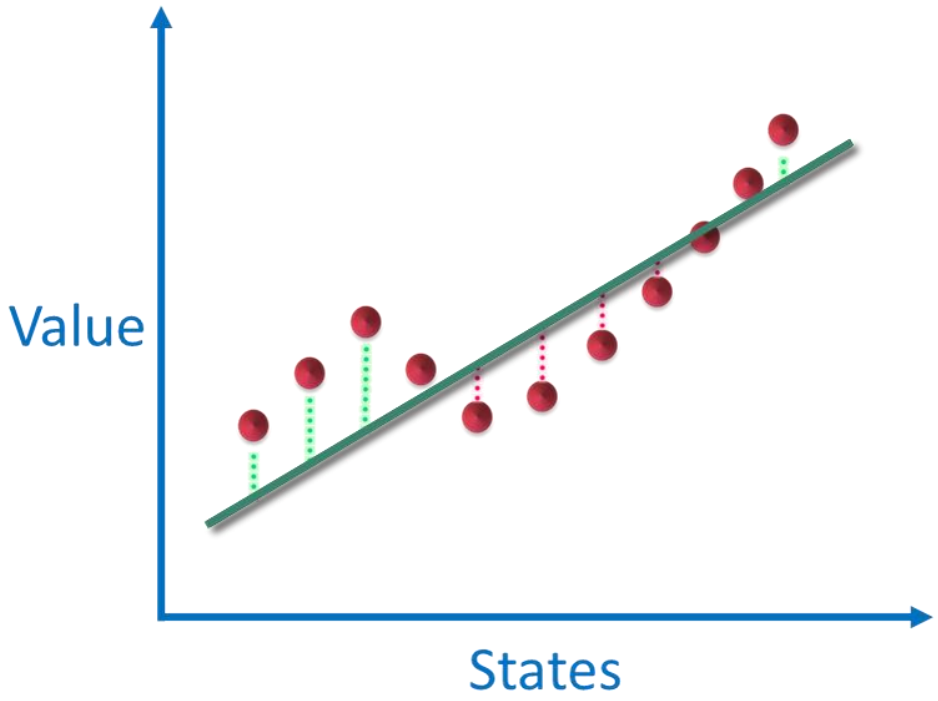
- gradient decent can overshoot the minimum

1. $\sum_{n=1}^{\infty} \alpha_n(a) = \infty$ and 2. $\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$

Gradient Descent

Linear function approximation

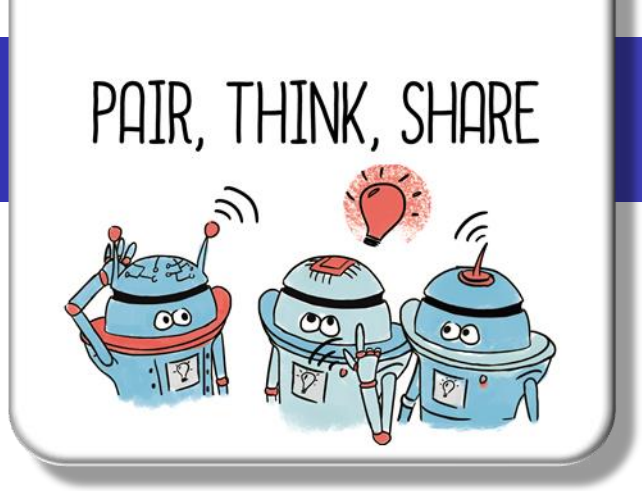
- **Linear value approximation is just an inner product of the weights with the feature vector for the state.**



$$\hat{v}(s_i, w) \doteq \sum w_i x_i(s)$$

$$\frac{\partial \hat{v}(s_i, w)}{\partial w_i} = x_i$$

$$\Delta \hat{v}(s_i, w) = \boldsymbol{x}(\boldsymbol{s})$$



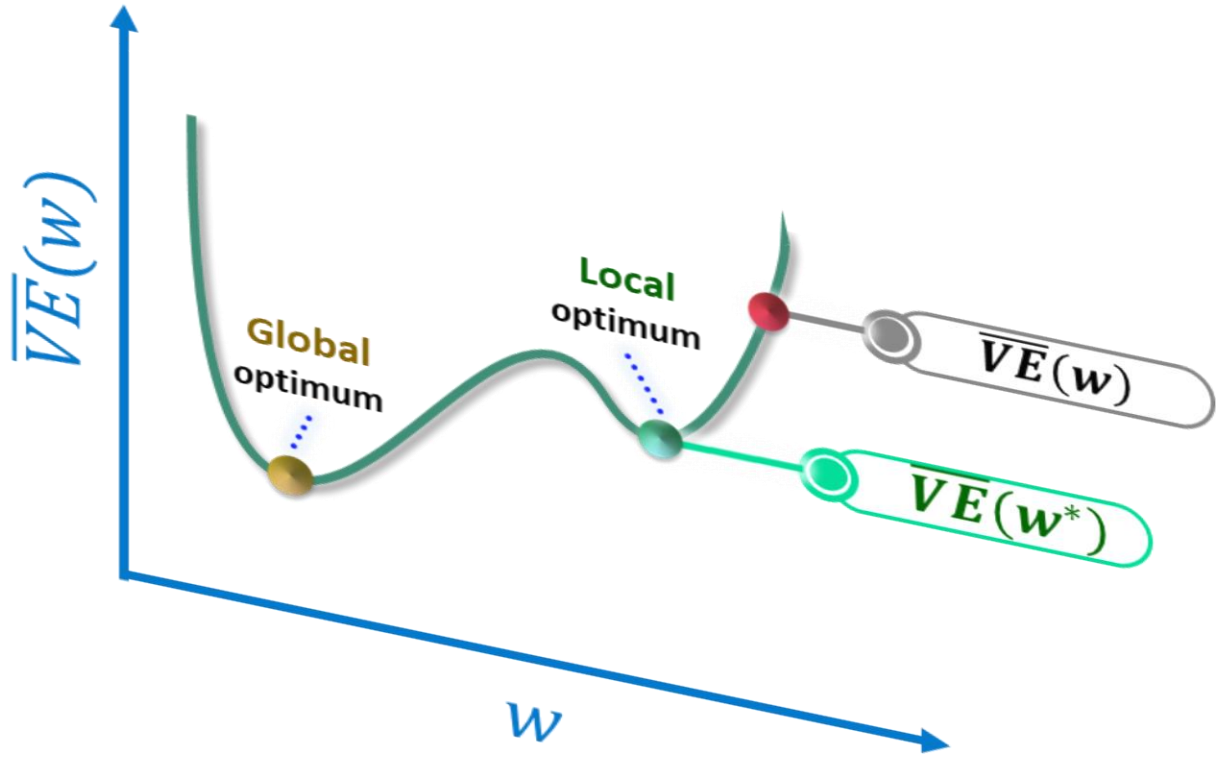
Global Minimum

- In some cases, gradient descent is guaranteed to converge to the global minimum, which is the best possible setting of the weights for the objective.
  - i.e. the mean squared value error with linear function approximation
- For more complex function approximate errors such as a neural network, a stationary point may not be a global minimum.

Question

- In those cases, can we still use gradient descent as an optimization approach? Why?

Stochastic Gradient Descent for $(\overline{VE})$



Stochastic Gradient Descent

Mathematical Calculation: Gradient Descent of $\overline{VE}$

- let us assume that states appear in examples with the same distribution, $\mu(s)$

$$\overline{VE}(\boldsymbol{w}) \doteq \sum_{s \in \mathcal{S}} [v_\pi(s) - \hat{v}(s, \boldsymbol{w})]^2$$

- Gradient of $\overline{VE}$

$$\nabla \overline{VE}(w) \doteq \nabla \sum_{s \in \mathcal{S}} [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

$$= \sum_{s \in \mathcal{S}} \nabla [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

$$= -\sum_{s \in \mathcal{S}} 2[v_\pi(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w})$$

Estimation *error* | Direction

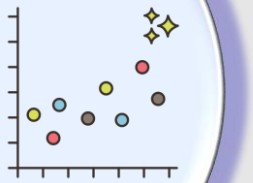
Estimation *error*	Direction
$v_\pi(s) - \hat{v}(s, \mathbf{w}) > 0$	↑
$v_\pi(s) - \hat{v}(s, \mathbf{w}) < 0$	↓

The gradient of the value function approximation depend on the particular parameterized function we are using

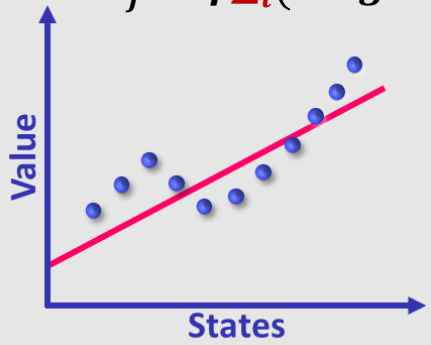
Stochastic Gradient Descent

Gradient Descent vs Stochastic Gradient Descent

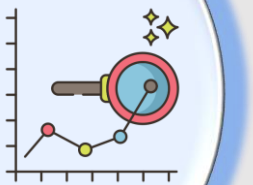
Gradient Descent



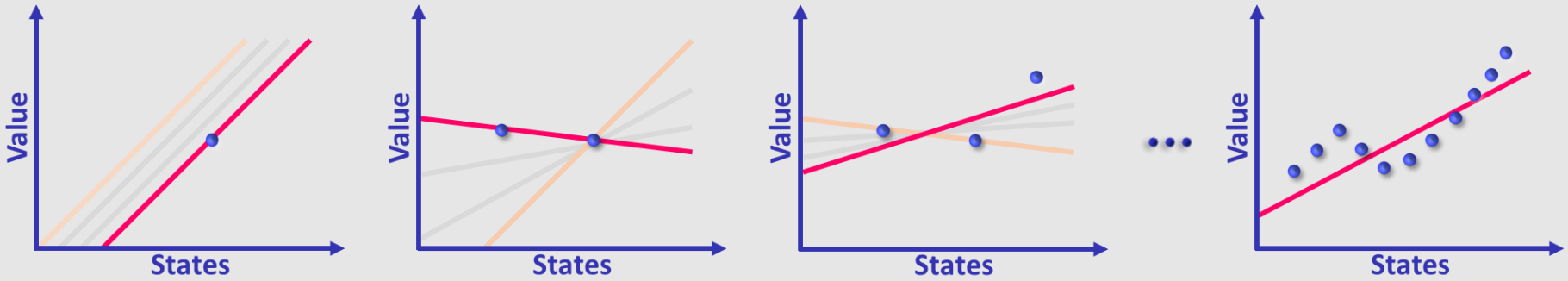
$$w_i \doteq w + \Delta w_j, \text{where: } \Delta w_j = \eta \sum_i \left(target^{(i)} - output^{(i)}\right) x_j^{(i)}$$



Stochastic Gradient Descent



$$w_i \doteq w + \Delta w_j, \text{where: } \Delta w_j = \eta \left(target^{(i)} - output^{(i)}\right) x_j^{(i)}$$



Stochastic Gradient Descent

Why Stochastic Gradient Descent? (1)

- **Stochastic gradient descent (SGD) is useful when we work with large datasets.**
  - In RL, Value Error objective requires summing over all states, which generally is not feasible

$$\sum_{s \epsilon \mathcal{S}} \mu(s)\, 2[v_\pi(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w})$$

  - Empirically measuring the higher order partial derivatives is computationally too expensive. Therefore, in case of a large size vector $\boldsymbol{w}$, gradient decent is not practical. Instead, we use SGD.
  - $\mu(s)$ is not available
  - There is generally no w that gets all the states, or even all the examples, exactly correct.

Stochastic Gradient Descent

Why Stochastic Gradient Descent? (2)

- **In SGD, we update the weights after each training sample**
  - In RL:
    - states are selected stochastically (i,e, using a policy)
    - leaning is online, as we observe a new state, we can update weights using SGD
    - also, we must generalize to all the other states that have not appeared in examples.

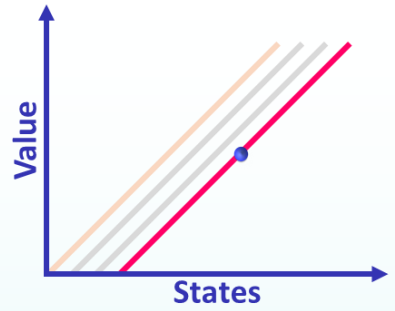
- **Also, SGD is efficient when there are redundancy in the data**
  - In RL many states share similar characteristics, and therefore, can be aggregated together.

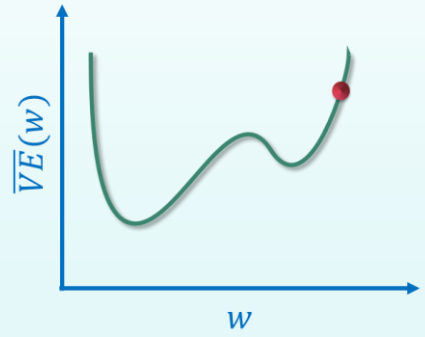
Stochastic Gradient Descent

Graphical representation: SGD for $(\overline{VE})$

$S_1, v_\pi(S_1)$

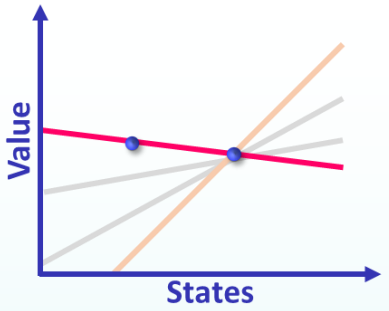
$$w_2 \doteq w_1 + \alpha[v_\pi(S_1) - \hat{v}(S_1, w_1)]\nabla\hat{v}(S_1, w_1)$$

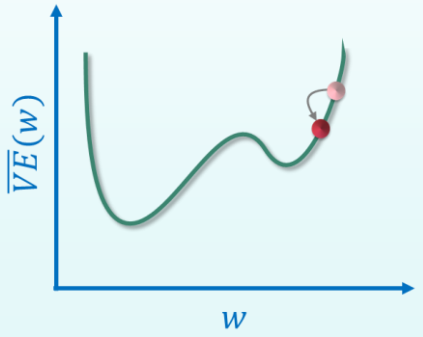




$S_2, v_\pi(S_2)$

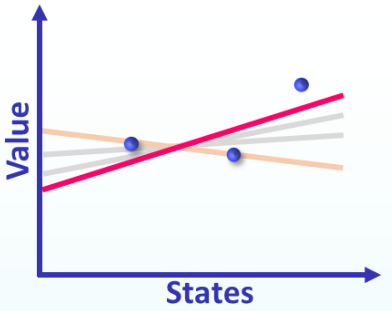
$$w_3 \doteq w_2 + \alpha[v_\pi(S_2) - \hat{v}(S_2, w_2)]\nabla\hat{v}(S_2, w_2)$$

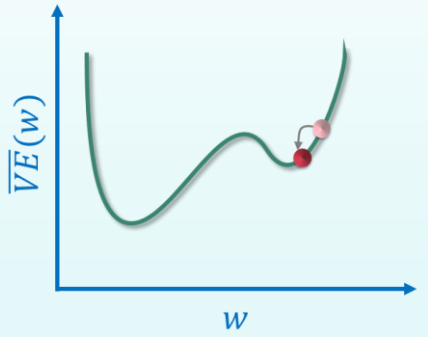


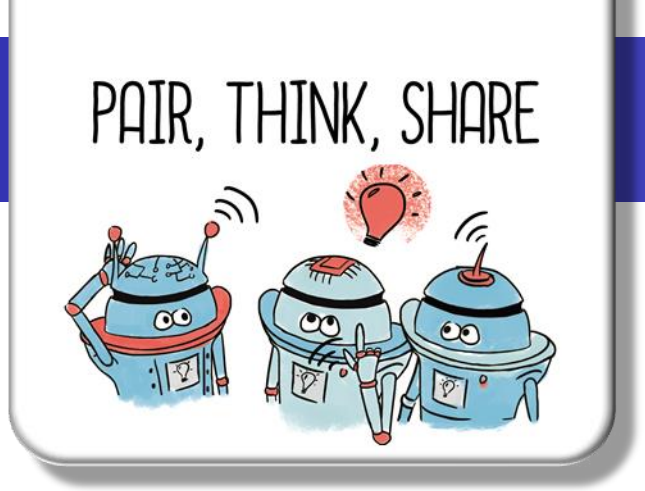


$S_3, v_\pi(S_3)$

$$w_4 \doteq w_3 + \alpha[v_\pi(S_3) - \hat{v}(S_3, w_3)]\nabla\hat{v}(S_3, w_3)$$







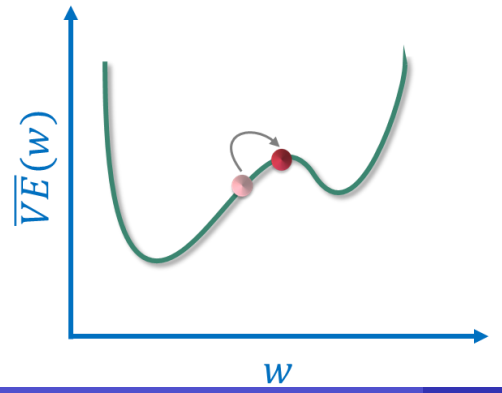
$$w_{t+1} \doteq w_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Estimation *error* | Direction

Stochastic gradient descent

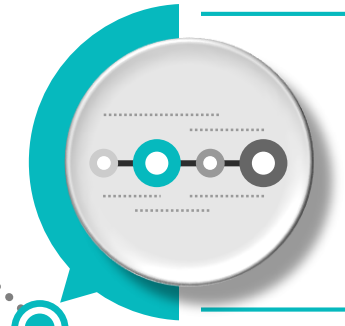
- Gradient descent methods are called "stochastic" when the update is done on only a single example, which might have been selected stochastically. Over many examples, making small steps, the overall effect is to minimize an average performance measure such as the VE.
- It may not be immediately apparent why SGD takes only a small step in the direction of the gradient.

1. **Could we not move all the way in this direction and completely eliminate the error on the example?**
2. **In SGD, do we necessarily reduce the error in every single example?**



Stochastic Gradient Descent

Update rule



$$w_{t+1} \doteq w_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

correct values

Update rule

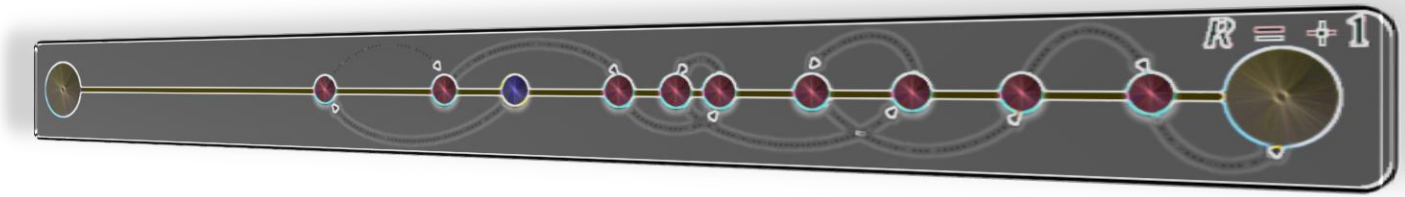
MC target $U_t \doteq G_t$

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

$$w_{t+1} \doteq w_t + \alpha[U_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Estimated value

Gradient Monte Carlo and TD



Gradient Monte Carlo and TD

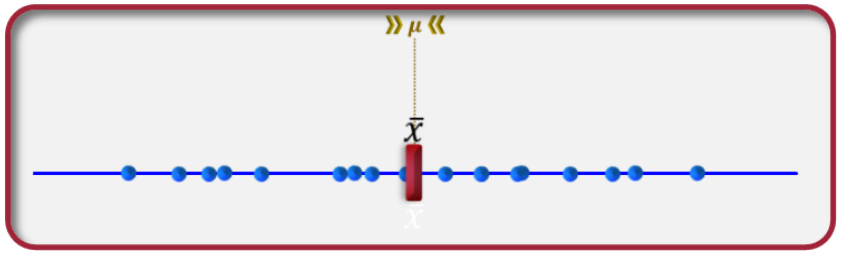
Update Rule with MC estimation

$$v_\pi(S_t) \qquad U_t$$

- **Recall from MCM**

Monte Carlo Methods

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



- **State-value prediction with MCM**

$$w_{t+1} \doteq w_t + \alpha[G_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Gradient Monte Carlo and TD

Pseudocode: Gradient Monte Carlo Algorithm

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

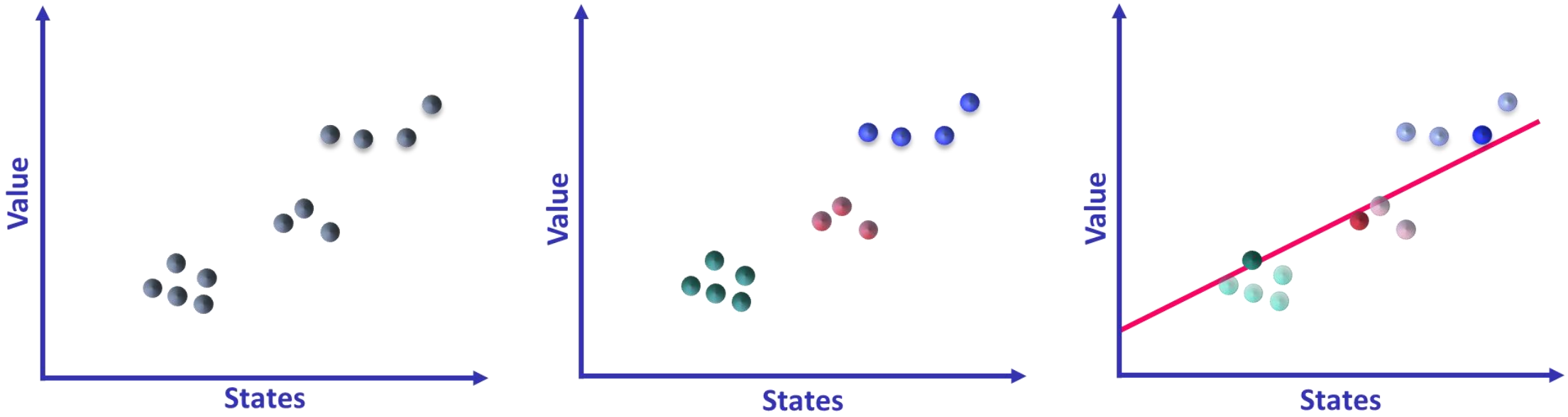
1. Input: the policy π to be evaluated
   Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
   Algorithm parameter: step size $\alpha > 0$
2. Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
3. Loop forever (for each episode):
   Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
4. Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
   $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G_t - \hat{v}(S_t, \mathbf{w}) \right] \nabla \hat{v}(S_t, \mathbf{w})$

1. **Initialization**
2. **Initialize weights**
3. **Monte Carlo loop**
   Selecting stochastically
4. **SGD**
   Update weights

Gradient Monte Carlo and TD

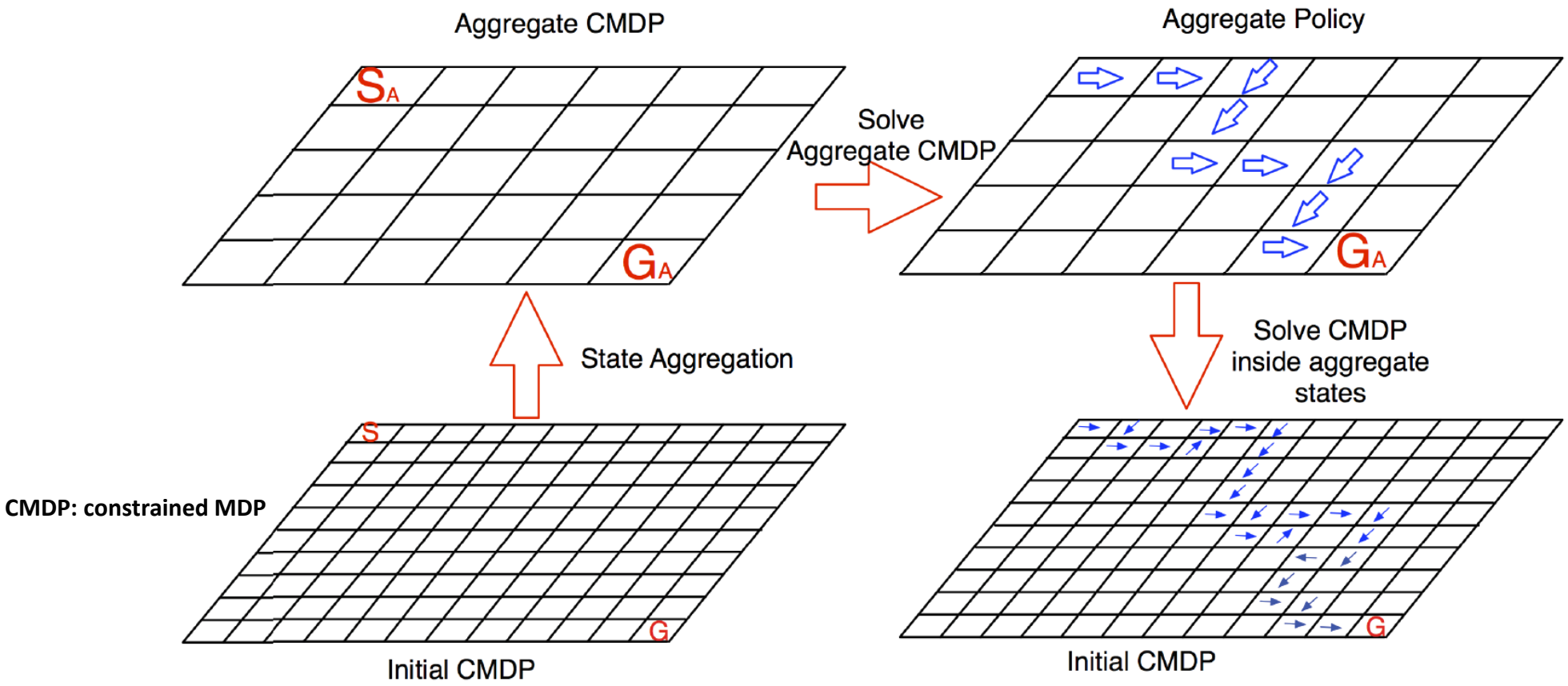
State aggregation

- **State aggregation is a simple form of generalizing function approximation in which states are grouped together**
  - The value of a group (cluster) generalizes to all states
  - Therefore, there is one estimated value (one component of the weight vector w) for each group.



Gradient Monte Carlo and TD

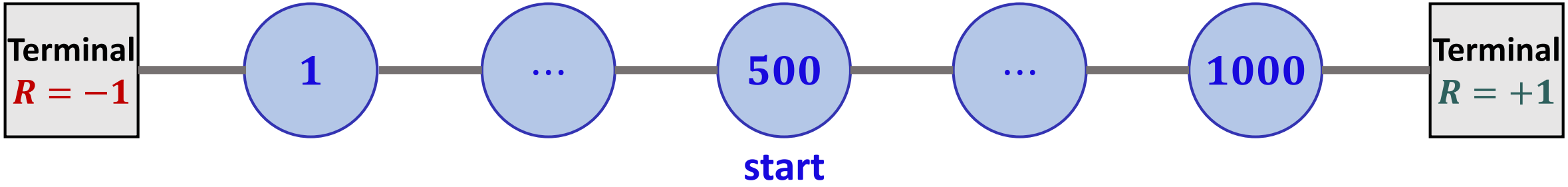
State Aggregation in a GridWorld setting

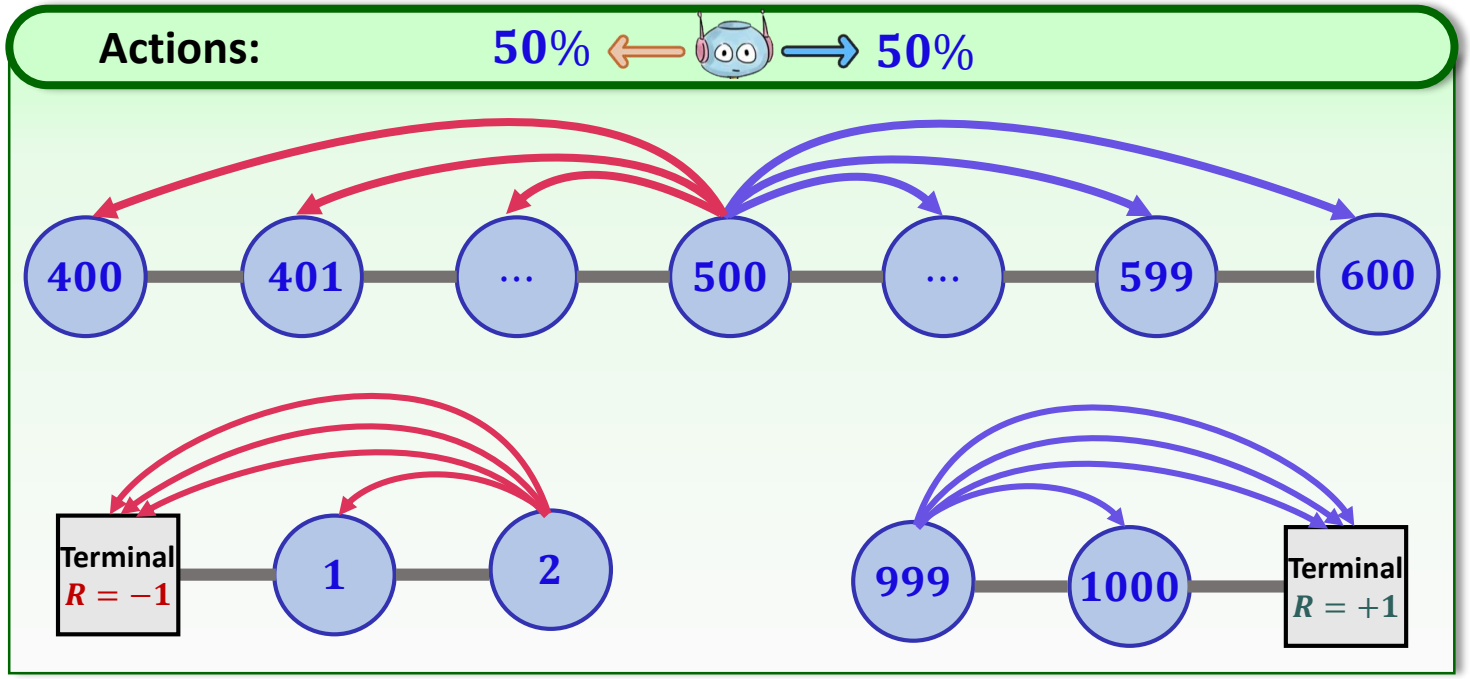


Feyzabadi, S., & Carpin, S. (2014, August). Risk-aware path planning using hirerachical constrained markov decision processes. In 2014 IEEE International Conference on Automation Science and Engineering (CASE) (pp. 297-303). IEEE.

Gradient Monte Carlo and TD

Example: 1000-state Random Walk





Settings:

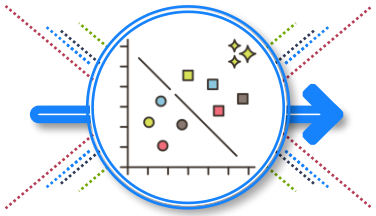
$$\gamma = 1$$
$$\alpha = 2 \times 10^{-5}$$

Gradient Monte Carlo and TD

State aggregation

Original states

State	Value
S_1	0
S_2	0
S_3	0
S_4	0
S_5	0
S_6	0
S_7	0
S_8	0



Aggregated states

State	Group	Value
S_1		
S_2		
S_3	Group 1	0
S_4		
S_5		
S_6		
S_7	Group 2	0
S_8		

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

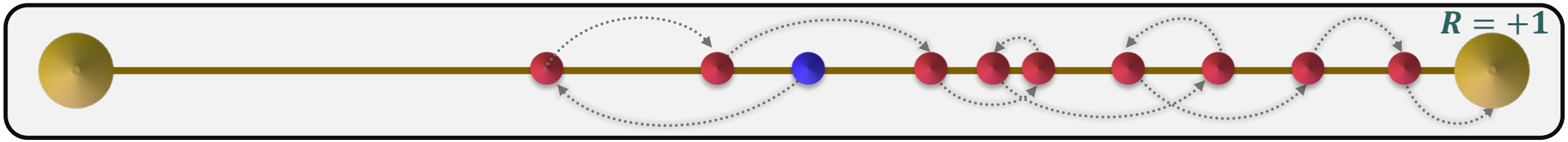
$$\boldsymbol{x}(s) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \hat{v}(s, w) = w_2$$

Gradient Monte Carlo and TD

State aggregation for 1000-state Random Walk

- **1000 states aggregation**

- **Monte Carlo trajectory**



Visited states: 500, 405, 476, … , 888, 957

Return: 1 , 1 , 1 , …, 1 , 1

Update group values: w_5 , w_5 , w_5 , …, w_9, w_{10} (Terminal state)

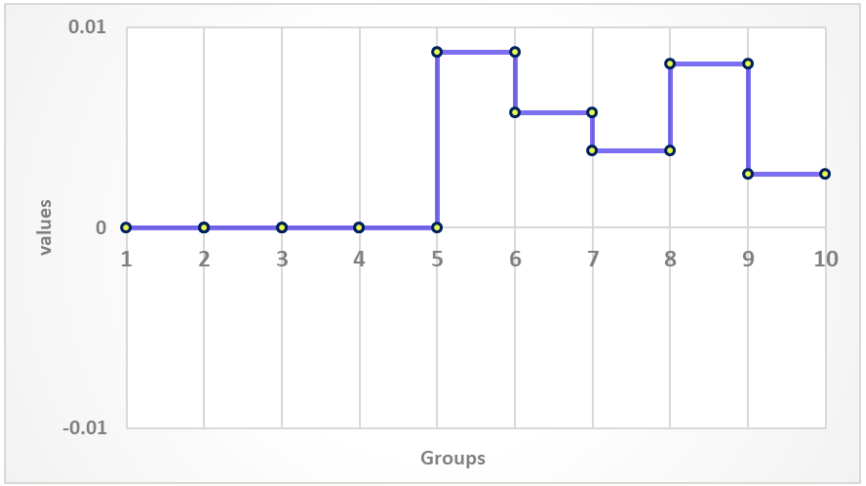
Gradient Monte Carlo and TD

State aggregation for 1000-state Random Walk

Visited states 500, 405, 476, … , 888, 957

Return 1 , 1 , 1 , …, 1 , 1

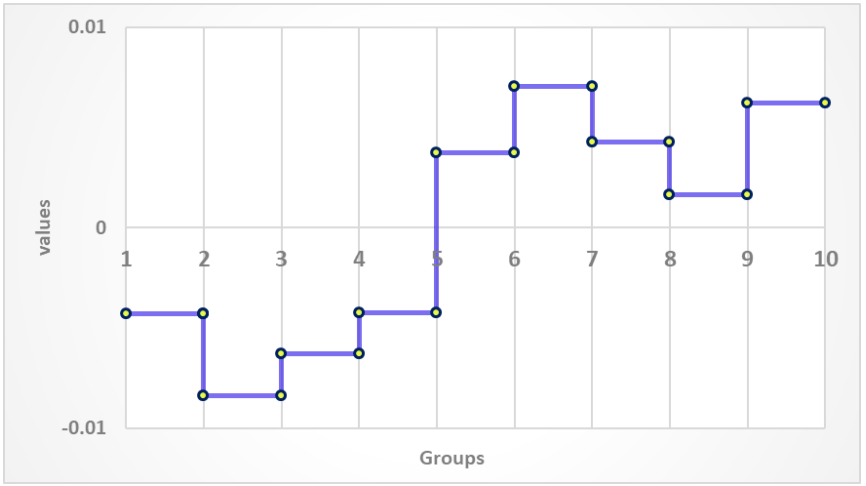
Update **group** values w_5 , w_5 , w_5 , …, w_9, w_{10}



Visited states 500, 555, 499, … , 220, 77

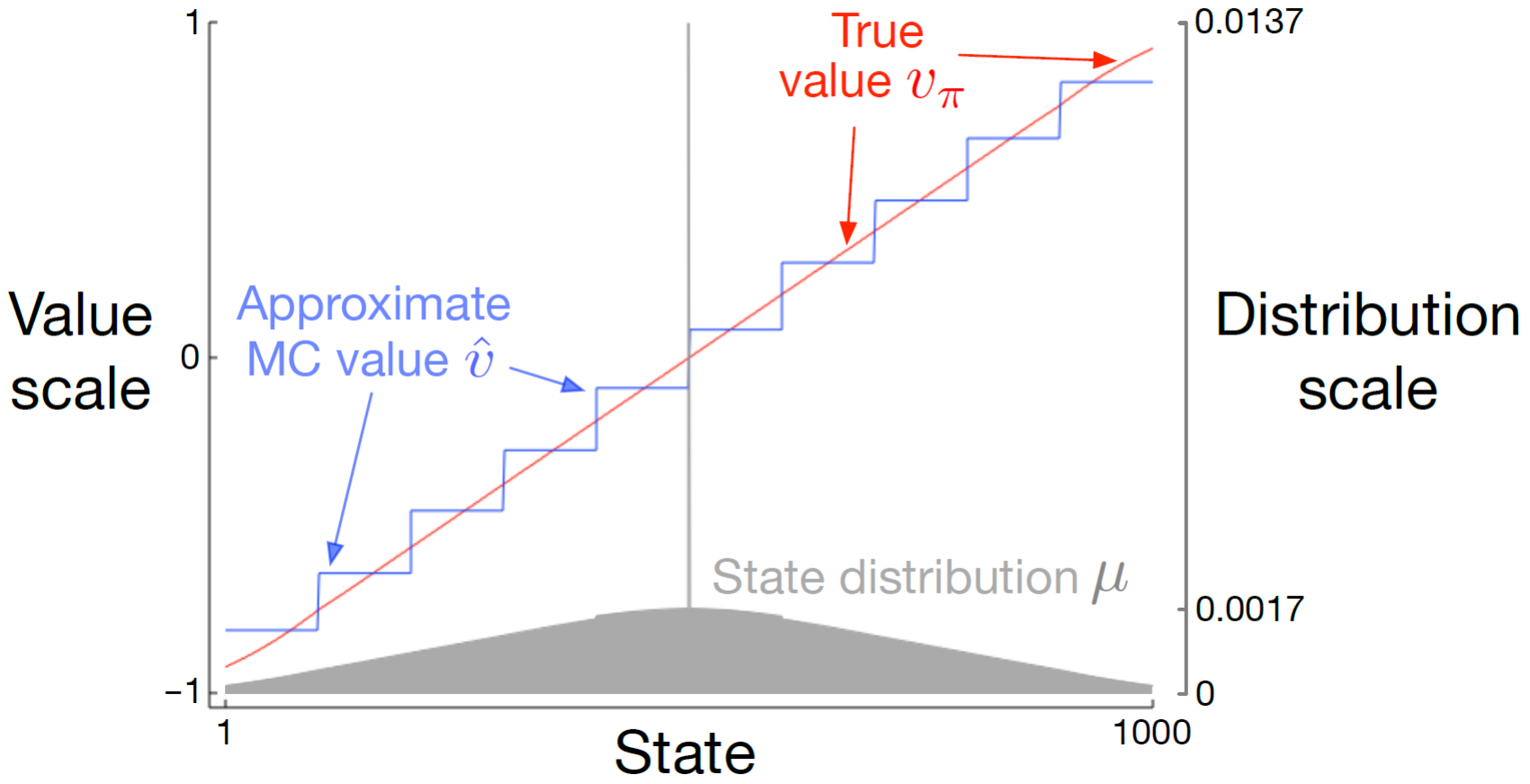
Return -1, -1 , -1 , … , -1 , -1

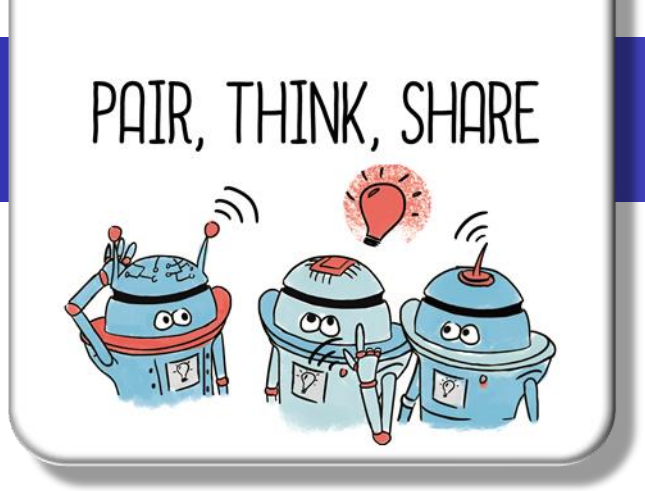
Update **group** values w_5 , w_6 , w_5 , …, w_3, w_1

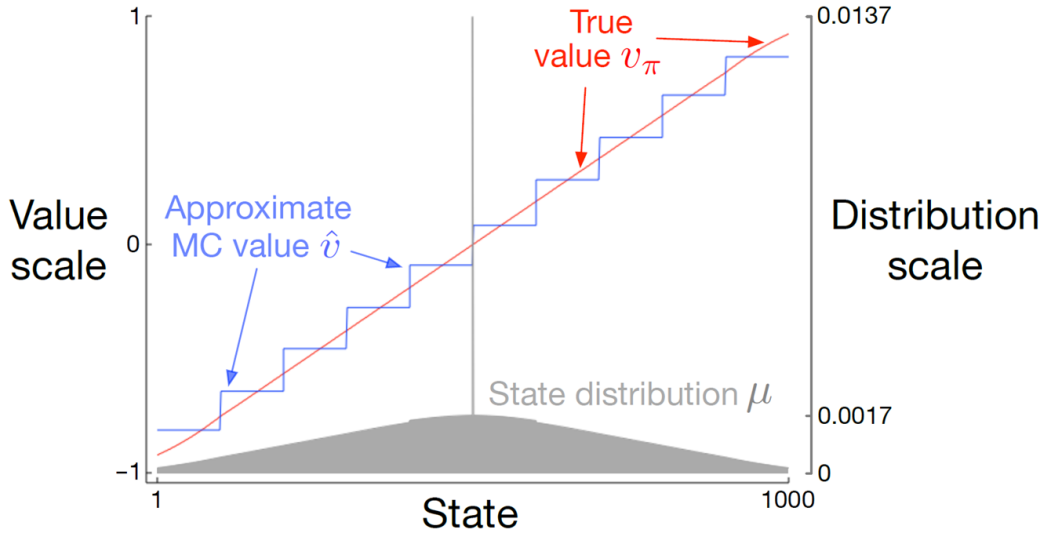


Gradient Monte Carlo and TD

Results: 1000-state Random Walk



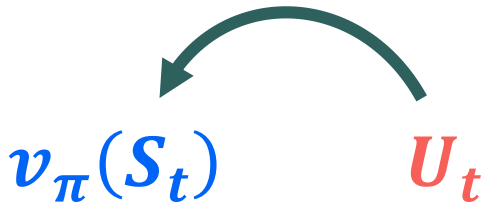




1000-state Random Walk

- Each step in the plot corresponds to our group of states that share the same approximate value.
- Some of the details of the approximate values are best appreciated by reference to the state distribution μ for this task, shown in the lower portion of the figure with a right-side scale.

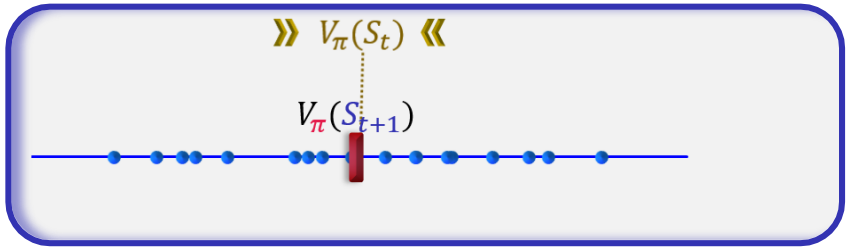
Explain why the red line does not pass directly through the center of all the blue steps?



- **Recall from TD**

Temporal Difference

$$V(S_t) \doteq V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$



- **State-value prediction with TD**

$$w_{t+1} \doteq w_t + \alpha[R_{t+1} + \gamma\, \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)]\nabla \hat{v}(S_t, w_t)$$

Stochastic Gradient Descent

Pseudocode: Semi-gradient TD(0)

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

Input: the policy π to be evaluated

1 Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Algorithm parameter: step size $\alpha > 0$

2 Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

3 Loop for each episode:

Initialize S

Loop for each step of episode:

Choose $A \sim \pi(\cdot|S)$

Take action A, observe R, S'

4 $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$

$S \leftarrow S'$

until S is terminal

1 Initialization

2 Initialize weights

3 Initialize weights

4 Semi-gradient TD update

Stochastic Gradient Descent

Convergence of function approximation

$$w_{t+1} \doteq w_t + \alpha[U_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Convergence of stochastic approximation

- If U_t is an unbiased estimate, that is, if $\mathbb{E}[U_t|S_t = s] = v_\pi(s_t)$ for each t, then w_t is guaranteed to converge to a local optimum under the usual stochastic approximation conditions for decreasing α.

Monte Carlo Method

$$w_{t+1} \doteq w_t + \alpha[G_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

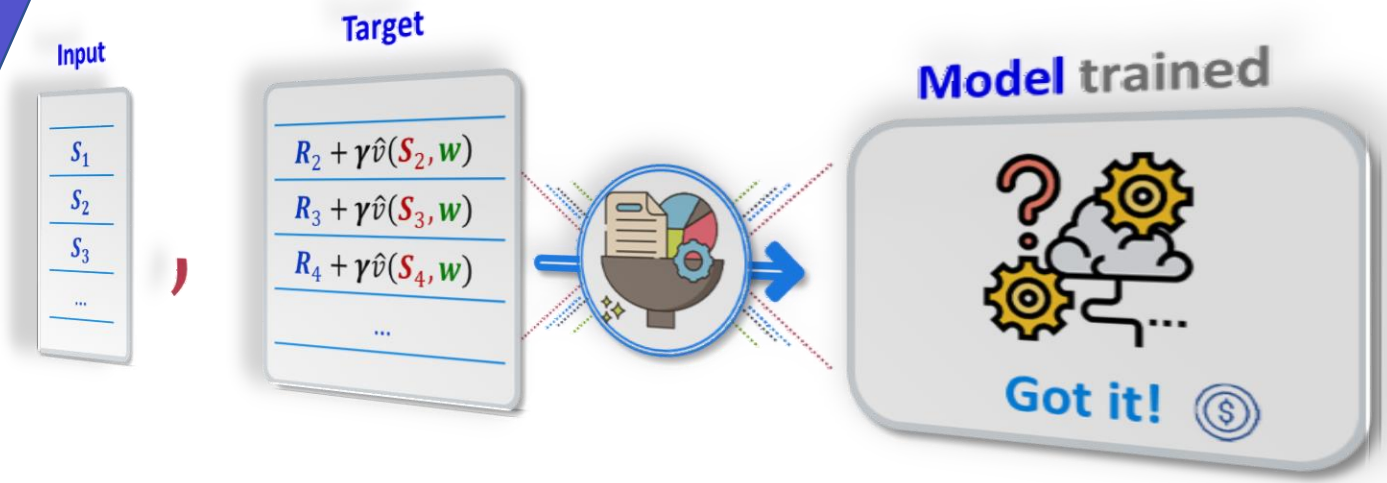
- **Unbiased estimate:** the general SGD method converges to a locally optimal approximation to $v_\pi(s_t)$
- Therefore, it is guaranteed to find a locally optimal solution

Temporal Difference

$$w_{t+1} \doteq w_t + \alpha[R_{t+1} + \gamma\,\hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

- **Biased estimate:** in bootstrapping methods like DP and TD, ***target value*** depends on the current value of the weight vector w_t
- Bootstrapping methods are not in fact instances of true gradient descent ***(semi-gradient methods)***

Linear Methods



Linear Methods

Linear methods approximate

- **Linear methods approximate the state-value function by the inner product between w and $x(s)$:**

$$\boldsymbol{x}(s) \doteq \begin{bmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_d(s) \end{bmatrix}, \quad \boldsymbol{w} \doteq \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Feature vector

Weights

- **$\hat{v}(s, w)$, is a linear function of the weight vector, w.**

$$\hat{v}(s, \boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s)$$

  - In this case the approximate value function is said to be ***linear in the weights***, or simply ***linear***.
  - For linear methods, features are basis functions because they form a linear basis for the set of approximate functions.

Linear Methods

Tabular TD as a special case of Linear TD

Tabular representation

State	Values
S_1	v_1
S_2	v_2
S_3	v_3
⋮	⋮
S_d	v_d

Function Approx. representation

State	Values
S_1	$\hat{v}(S_1, w)$
S_2	$\hat{v}(S_2, w)$
⋮	⋮
S_{10}	$\hat{v}(S_{10}, w)$
⋮	
S_d	$\hat{v}(S_d, w)$

$$\begin{bmatrix} w_1 & w_2 & \cdots & w_{10} & \cdots & w_d \end{bmatrix} \cdot \begin{bmatrix} x_1 = 0 \\ x_2 = 0 \\ \vdots \\ x_{10} = 1 \\ \vdots \\ x_d = 0 \end{bmatrix} = w_{10}$$

$$\hat{v}(s_i, w) \doteq \langle \boldsymbol{w}, x(s_i) \rangle = w_i$$

SGD updates and its convergence

- **The gradient of the approximate value function with respect to w in this case is:**

$$\nabla \hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$$

  - Thus, in the linear case the general ***SGD update rule*** is:

$$\boldsymbol{w_{t+1}} \doteq \boldsymbol{w_t} + \boldsymbol{\alpha}[\boldsymbol{U_t} - \hat{\boldsymbol{v}}(\boldsymbol{S_t}, \boldsymbol{w_t})]\, \boldsymbol{x}(S_t)$$

  - Because it is so simple, the linear SGD case is one of the most favorable for mathematical analysis.

- **Convergence results**
  - In particular, in the linear case there is only ***one optimum*** (or, in degenerate cases, one ***set of equally good optima***)
  - Thus, any method that is guaranteed to converge to or near a local optimum is automatically guaranteed to converge to or near the global optimum.
    - i.e. gradient Monte Carlo algorithm

The True Objective for TD: TD fixed point (1)

- **The semi-gradient TD(0) algorithm converges under linear function approximation but this does not follow from general results on SGD.**

  - In TD(0), the update at each time t is:

shorthand *notation*

$$\boldsymbol{x_t} : \boldsymbol{x}(S_t)$$

$$\begin{aligned} w_{t+1} &\doteq w_t + \alpha\left[R_{t+1} - \gamma w_t^T x_{t+1} - w_t^T x_t\right]\boldsymbol{x_t} \\ &= w_t + \alpha[R_{t+1}\boldsymbol{x_t} - \boldsymbol{x_t}(x_t - \gamma x_{t+1})^T w_t\ ] \end{aligned}$$

  - The update rule can be written as:

$$\Delta \boldsymbol{w} = \alpha[\underbrace{R_{t+1}\boldsymbol{x_t}}_{\text{noise}} - \underbrace{\boldsymbol{x_t}(x_t - \gamma x_{t+1})^T w_t}_{\text{expected update}}\ ]$$

The True Objective for TD: TD fixed point (2)

$$\Delta w = \alpha[R_{t+1}\boldsymbol{x_t} - \boldsymbol{x_t}(x_t - \gamma x_{t+1})^T w_t]$$

- The expected weight update vector can be written

$$\mathbb{E}[\Delta w] = \alpha(b - \boldsymbol{A} w_t), \text{ where:}$$

noise $\boldsymbol{b} \doteq \mathbb{E}[\boldsymbol{R_{t+1}}\boldsymbol{x_t}] \in \mathbb{R}^{\mathrm{d}}$

expected update $\boldsymbol{A} \doteq \mathbb{E}\left[\boldsymbol{x_t}(\boldsymbol{x_t} - \boldsymbol{\gamma}\boldsymbol{x_{t+1}})^{\boldsymbol{T}}\right] \in \mathbb{R}^{\mathrm{d}} \times \mathbb{R}^{\mathrm{d}}$

$$\mathbb{E}[\Delta w_{TD}] = \alpha(b - \boldsymbol{A} w_{TD})$$

- By setting $\mathbb{E}[\Delta w_{TD}] = 0$

$$\boldsymbol{w_{TD}} = \boldsymbol{A^{-1}b}$$

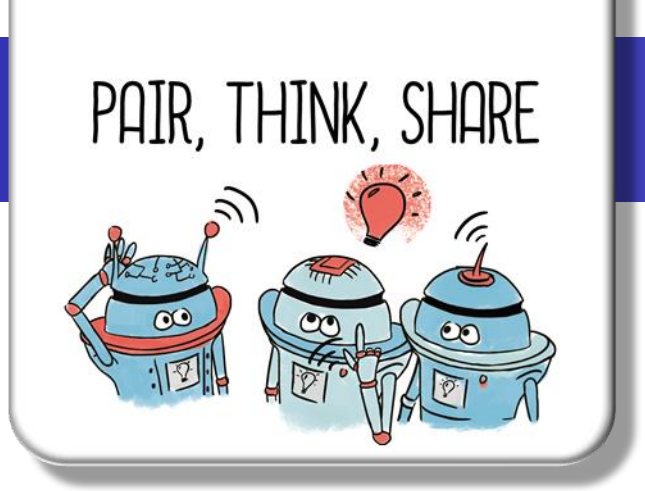
TD fixed point

Value Error of TD fixed point

- **Review Linear TD**
  - Biased estimation
  - Convergence to TD fixed point (check SB book for proof)
  - Linear TD approximates values based on Bellman equation and minimize an objective function called ***Projected Bellman error***.

- **At the TD fixed point, it has also been proven (in the continuing case) that the $\overline{VE}$ is within a bounded expansion of the lowest possible error:**

$$\overline{VE}(w_{TD}) \leq \frac{1}{1-\gamma} \min_{w} \overline{VE}(w)$$

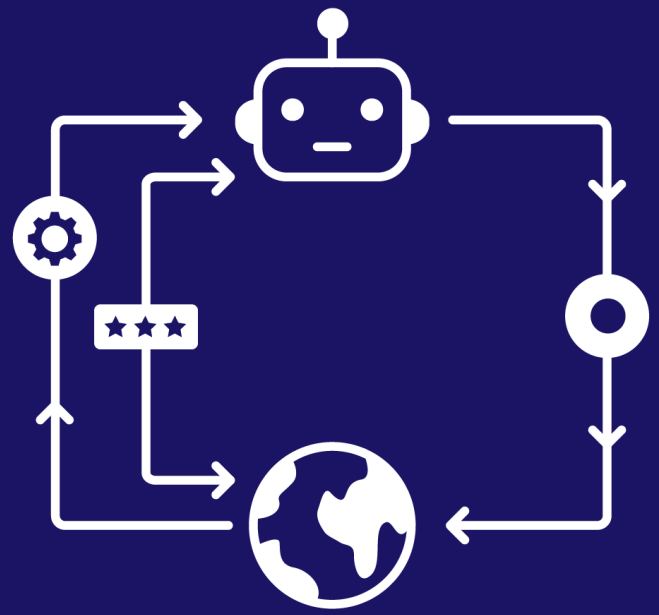


$$\overline{VE}(w_{TD}) \leq \frac{1}{1-\gamma} \min_{w} \overline{VE}(w)$$

TD fixed point and the Minimum of $\overline{VE}$

- Linear TD does not minimize the MSE objective, but in fact minimizes ***Projected Bellman error***
- *However, TD fixed point is very close to the minimum value error solution*

1. **What is the impact of Gamma (γ) on the difference between the TD fixed point and the minimum value error solution**
2. **Does this bound depend on the quality of the features? What happened if we can perfectly represent the value function**
3. **In general, why isn't the TD fixed point equal to the minimum value error solution?**



Summary