On-Policy Prediction with Approximation

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

Summary

Introduction

Introduction to the second part of the course

- Second part of the course: **How can we extend tabular RL methods to apply them to problems with arbitrarily large state spaces?**
- E.g., # possible camera images > # atoms in the universe
- Almost all **states** encountered have **never been seen before** → **Generalization from previously encountered (similar) states**
- Optimal policies → Good approximate solutions
- **Combine RL with function approximation (supervised learning)**
- The **RL setting** introduces **new issues** to **supervised learning**: e.g., nonstationarity, bootstrapping, delayed targets

Introduction to this lecture

- **Goal of this lecture**: substitute **tabular representations** of **state-value function** v_π with **function approximations**

- Approximated v_π are **estimated** from **on-policy** data, i.e., from experience generated using the known policy π.

- Approximations are based on **parametrized functions** $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$ where $\mathbf{w}$ is the vector of parameters

- Example: $\hat{v}$ might be a **linear function** in features of the state with $\mathbf{w}$ vector of feature weights, or a multilayer **artificial neural network** with $\mathbf{w}$ vector of connection weights, or a **decision tree** with $\mathbf{w}$ split points and leaf values of the tree

- **Adjusting the weights** several functions can be implemented

Introduction to this lecture

- Typically, the number of weights is much less than the number of states, i.e., $d \ll |\mathcal{S}|$
- Changing **one weight** changes the estimated value of **many states** (**generalization**)
- **Generalization** makes **reinforcement learning** more **powerful** but also more **difficult** to manage and understand
- Extending RL to function approximation makes it applicable to **partially observable problems** (i.e., full state not available)

Value-function approximation

Value-function approximation

- All **prediction metods** seen so far are based on **updates** of an **estimated value function**

- MC update: $S_t \mapsto G_t$

  State updated

  Update target

- TD update: $S_t \mapsto \underbrace{R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)}$

  State updated

  Update target

- DP update: $s \mapsto \underbrace{\mathbb{E}_\pi[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) \mid S_t = s]}$

  State updated

  Update target

Value-function approximation

- Can we interpret each **update** as specifying an **example** of the **desired input-output behaviour of the value function? Yes!**

$$\begin{tikzcd} s \arrow[r] & \boxed{v_\pi} \arrow[r] & v_\pi(s) \end{tikzcd}$$

- **Update in tabular representations** of the value function: the table entry for the **estimated value of state *s*** is **shifted** a fraction of the way **towards** the **targer *u*** (**estimated values of other states are unchanged**)

- **Update in function approximations** of the value function: **arbitrary complex parameter updates** are available. **Updating at state *s* can change value estimations of other states**

- **Supervised learning** can be used to compute weights $\mathbf{w}$ using **function approximation** methods

Value-function approximation

- **Problem: not all** function approximation methods are **equally well suited for RL**

- In RL, learning must be performed **online**, while the agent interacts with the environment.

- We need **learning methods** that
  - learn efficiently from **incrementally acquired data**
  - handle **nonstationary target functions**

- Example: in GPI we seek to learn q_π as π changes.

- Methods that cannot deal with such nonstationarity are less suitable for RL

The Prediction Objective

The Prediction Objective

- Which **objective** do we use to **evaluate** the **approximated function**?
- In **tabular case** a continuous measure of prediction quality was not necessary because
  - the learned value function could become equal to the true one
  - updates affect only single states
- With **function approximation** these two assumptions are not guaranteed
- We define **which states we care most about** defining a **state distribution** $\mu(s) \geq 0, \sum_s \mu(s) = 1$
- Then, a natural **objective function** is the **Mean Squared Value Error**

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

The Prediction Objective

- The square root of $\overline{\mathrm{VE}}$ provides a **measure** of how much the **approximate** values differ form the **true** values

- Often $\mu(s)$ is set to the fraction of time spent in state *s* (***on-policy distribution***)

- In episodic tasks: let *h(s)* the probability an episode starts in state *s*, then the **number of time steps spent, on average, in state s in a single episode** is

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

  and the on-policy distribution is then

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

The Prediction Objective

- In **continuing tasks** the on-policy distribution is the stationary distribution under π. In **episodic tasks** also depends on state initial probability

- The **formal analysis** of the **continuing** and **episodic** cases must be treated separately with value function approximation

The Prediction Objective

- The **goal** of $\overline{\mathrm{VE}}$ is to find a **global optimum**, namely, a **weight vector** $\mathbf{w}^*$ for which

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

  for all possible $\mathbf{w}$

- This is **possible** for **simple function approximators** (e.g., linear models) rarely for **complex approximators** (e.g., ANNs and decision trees) in which learning usually converges to **local optima**, i.e., $\mathbf{w}^*$ for which

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

  for all $\mathbf{w}$ **in some neighbourhood of** $\mathbf{w}^*$

- This is the **best** that can be done and it is usually **enough** although in many cases there is **no guarantees of convergence to the optimum**

The Prediction Objective

In summary, so far we have described:

- A **framework** for **combining** RL methods for **value prediction** with **function approximation** methods (using RL updates as training examples)
- A $\overline{\mathrm{VE}}$ performance measure that these methods may aspire to minimize

In the rest of the lecture we will consider **function approximation methods** based on **gradient-descent** since they are particularly **promising** and reveal **key theoretical properties**

Stochastic-Gradient and Semi-Gradient Methods

Stochastic-gradient Methods (SDG)

- **Class of learning methods** for function approximation in value prediction: **Stochastic Gradient Descent (SGD)**
- Among the most **widely used** of all function approximation methods
- Well suited to **online RL**

Let:

- $\mathbf{w} \doteq (w_1, w_2, \ldots, w_d)^\top$ a weight vector

- $\hat{v}(s,\mathbf{w})$ is a **differentiable** function of $\mathbf{w}$ for all states *s*

At each time step $t = 0, 1, 2, 3, \ldots,$ we observe a new example $S_t \mapsto v_\pi(S_t)$ and update $\mathbf{w}_t$

States S_t can be **randomly selected** or they can be **successive** states of an interaction with the environment

Stochastic-gradient Methods (SGD)

- Values $v_\pi(S_t)$ are **unknown** but even though we could observe their **true values**, learning the approximate function would be difficult
- The approximator has **limited "resolution"**. There is no $\mathbf{w}$ that gets all the states exactly correct
- **Goal of SGD: to minimize error on the observed examples**
- **Strategy of SGD: adjust $\mathbf{w}$ after each example by a small amount in the direction that would most reduce the error on that example**

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t),\end{aligned}$$

where α >0 and $\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top$ **gradient** of *f*

Stochastic-gradient Methods (SGD)

- The **negative gradient of the example's squared error** is the **direction in which the error falls most rapidly**

- SDG is called **"stochastic"** when the **update is done on only a single sample**

- Over many examples, making small steps, the effect is to minimize $\overline{\mathrm{VE}}$

- Why performing only **"small" steps**? If we **completely corrected each example in one step** then we would **not balance the error** (which cannot be completely removed) **on all samples**

- Convergence results on SGD assume that α decreases over time (according to standard stochastic approximation conditions – Lec. 2)

Stochastic-gradient Methods (SGD)

- In practice the **target output observed** at time *t*, $U_t \in \mathbb{R}$, is **not the true value** $v_\pi(S_t)$, but some **random approximation** of it (e.g., noisy corrupted value of $v_\pi(S_t)$ or a bootstrapping target)

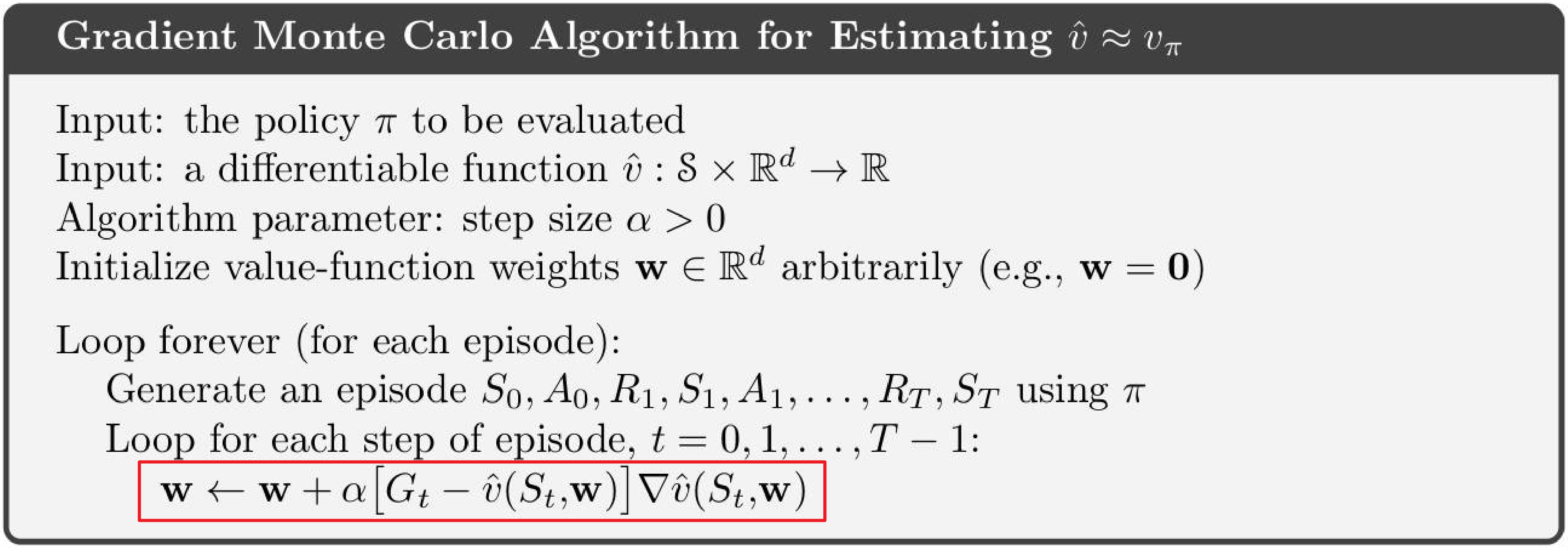
- We perform an **approximate update** using $U_t \in \mathbb{R}$:

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

- If U_t is an **unbiassed estimate of the value**, i.e., $\mathbb{E}[U_t | S_t = s] = v_\pi(S_t)$ then $\mathbf{w}_t$ is guaranteed to converge to a local optimum

Gradient MC algorithm for estimating the value function

- The **Monte Carlo target** $U_t \doteq G_t$ is an unbiased estimate of $v_\pi(S_t)$, hence the **SGD version of MC state-value prediction converges**



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

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

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \dots, T-1$:
        $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[G_t - \hat{v}(S_t,\mathbf{w})\big] \nabla \hat{v}(S_t,\mathbf{w})}$
```

- Notice: MC provides a **non-bootstrapping estimate of** $v_\pi(S_t)$

Stochastic-gradient Methods (SGD)

- If a **bootstrapping estimate** of $v_\pi(S_t)$ is used as the target U_t (e.g., in TD and DP), then **convergence is not guaranteed**

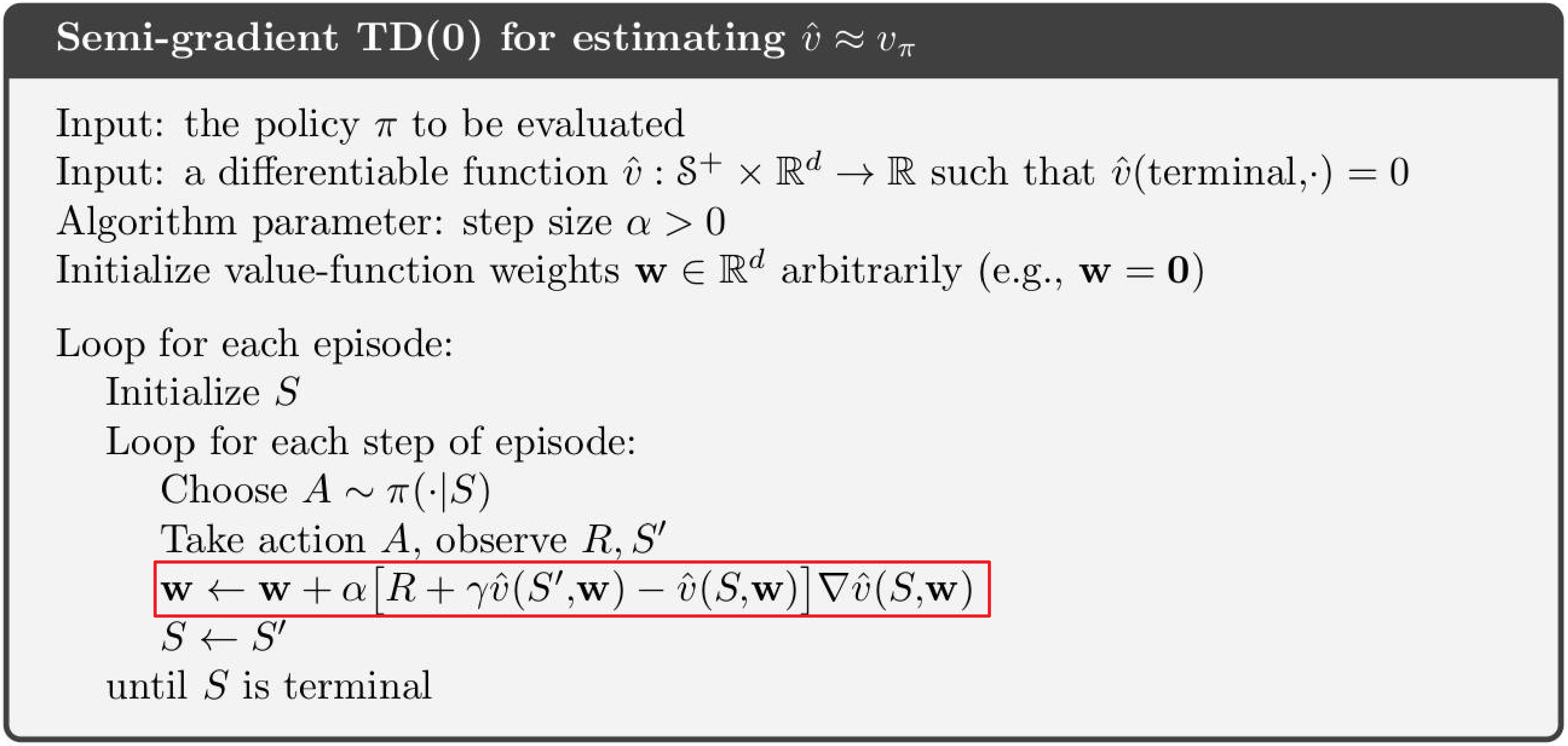
- This is because the **target must be independent of** $\mathbf{w}_t$

- These methods are called **semi-gradient (bootstrapping) methods**

- They **do not converge as robustly as gradient methods** but they **converge reliably in important cases (e.g., linear case)**

- **Advantage of semi-gradient methods:**
  - They enable **faster learning**
  - They enable **learning continual and online**, **without waiting for the end of the episode**

Semi-Gradient TD algorithm for estimating the value function



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

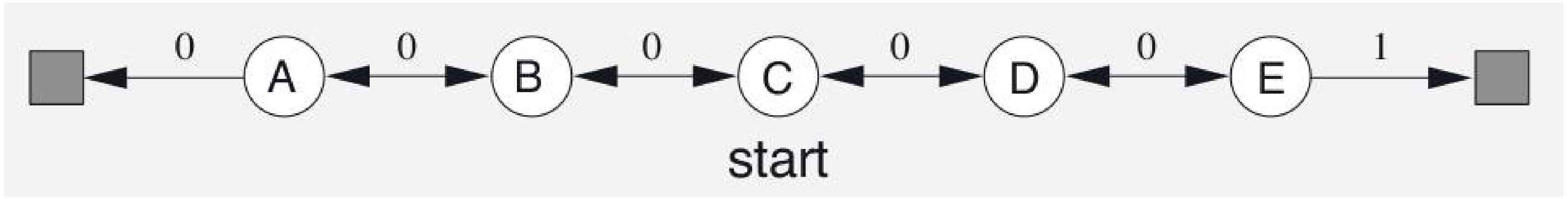
```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big] \nabla \hat{v}(S,\mathbf{w})}$
        $S \leftarrow S'$
    until $S$ is terminal
```

Example: state aggregation on the 1000-state random walk

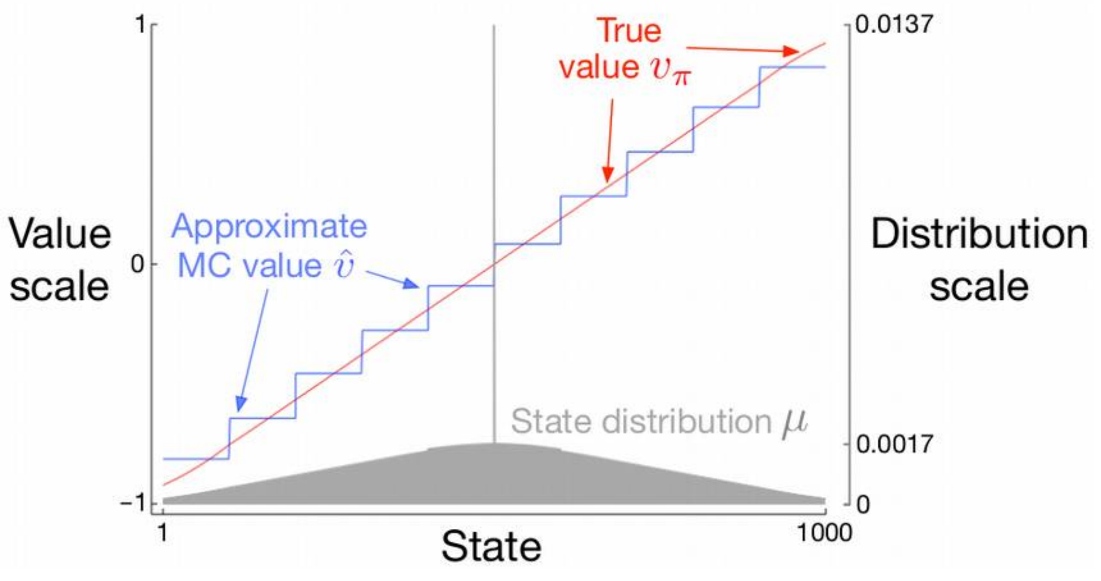
State aggregation: simple generalizing function approximation

- States are grouped together with one estimated value (constant)
- Each component of $\mathbf{w}$ is the estimation for a group of states
- The gradient $\nabla\hat{v}(S_t,\mathbf{w}_t)$ is 1 for the components of the group of S_t and 0 for the other components

- Consider a 1000-state version of of the random walk task



- **Function approximation by state aggregation using the gradient MC algorithm**
  - 100.000 episodes,
  - alpha=2x10^-5
  - 10 groups



Linear Value Function Approximation

Linear Methods

- Approximate function $\hat{v}(\cdot,\mathbf{w})$ with **linear function of the weight vector** $\mathbf{w}$

- For each state *s* there is a real-valued **feature vector**

$$\mathbf{x}(s) \doteq (x_1(s), x_2(s), \ldots, x_d(s))^\top$$

  with the same number of components (**features**) as $\mathbf{w}$ (i.e., *d*). The value of each feature is a **function of the state** $x_i : \mathcal{S} \to \mathbb{R}$

- **Linear method approximations of the state-value function** implement the **inner-product** between $\mathbf{w}$ and $\mathbf{x}(s)$:

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

- The **approximate value-function** is said to be **linear in the weights**

Linear Methods

- It is natural to use **SGD updates** with **linear function approximations**

- The **gradient** of the approximate value function w.r.t. **w** in this case is

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

- Hence, the **SDG update** becomes:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{v}(S_t,\mathbf{w}_t)\Big]\boxed{\mathbf{x}(S_t)}$$

- Simple form → Good for mathematical analysis (e.g., convergence)

- Only one optimum → local optimum = global optimum

Linear Methods

- The **Gradient Monte Carlo algorithm converges** to the global optimum of the $\overline{\mathrm{VE}}$ under linear function approximation

- The **semi-gradient TD(0) algorithm** also **converges** under linear function approximation. This result requires a separate theorem (the weight vector converges to a point near the local optimum)

  - The **update** at each time *t* is

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma\mathbf{w}_t^\top\mathbf{x}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t\Big)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\big)^\top\mathbf{w}_t\Big)\end{aligned}$$

  - At steady state the expected next weight vector is

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$$

with $\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d$ and $\mathbf{A} \doteq \mathbb{E}\Big[\mathbf{x}_t\big(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\big)^\top\Big] \in \mathbb{R}^d \times \mathbb{R}^d$

Linear Methods

- The **TD fixed point for linear semi-gradient TD(0)** can be computed as:

$$\begin{aligned} & \mathbf{b} - \mathbf{A}\mathbf{w}_{\text{TD}} = \mathbf{0} \\ \Rightarrow \quad & \mathbf{b} = \mathbf{A}\mathbf{w}_{\text{TD}} \\ \Rightarrow \quad & \mathbf{w}_{\text{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}. \end{aligned}$$

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

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

- Namely, the **asimptotic error of the TD method** is no more than $\frac{1}{1-\gamma}$ times the smallest possible error, i.e., the error reached in the limit by the Monte Carlo method

- γ is usually close to 1 → **Substantial potential loss but TD methods have reduced variance and are faster than MC methods in practice**

Feature Constructions for Linear Methods

Feature Construction for Linear Methods

Advantages of linear approximation:

- convergence guarantees
- data efficiency
- computational efficiency

- These advantages **depend a lot on how the states are represented** in terms of **features**

- Appropriate features → Prior domain knowledge

- **Features** should correspond to the **aspects of the state space** along which **generalization** may be appropriate
  - E.g., states of geometric objects: features for each possible shape, color, size, etc.
  - E.g., states of mobile robot: features for location, remaining battery, etc.

Feature Construction for Linear Methods

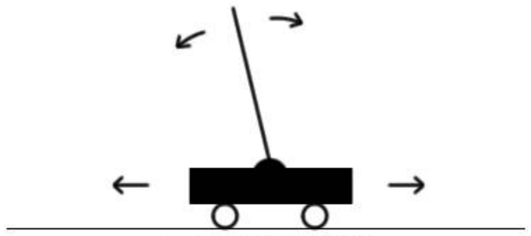
There exist **several ways to construct meaningful features** (e.g., polynomials, Fourier basis, etc.). This is beyond the scope of the course (see **Sec. 9.5** of the Sutton and Barto book for details)

Limitation of linear approximation: it cannot consider **interactions** between features

- E.g., in the pole-balancing task,

  high angular velocity can be either good

  or bed depending on the angle



Single Pole Balancing

- A linear value function cannot represent this if this features are coded separately for the angle and the angular velocity

Nonlinear Value Function Approximation

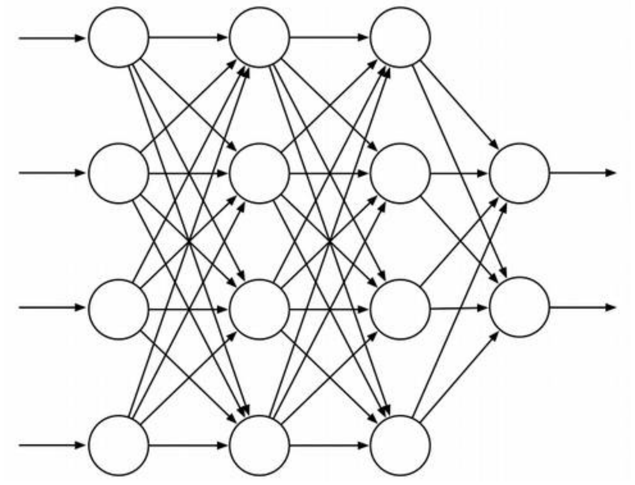
Nonlinear Value Function Approximation

There exist several **non-linear methods for approximating the value function**, such as,

- Artificial Neural networks (ANNs)
- Memory-based (nonparametric) functions
- Kernel-based functions

ANNs: have recently become the most popular approximation functions

- They are **universal function approximators**
- In **deep architectures** they can generate **hierarchical representations of features** automatically (vs hand-crafted features)



- They typically learn by **stochastic gradient** methods
- They can learn value functions (see **Deep Q Networks** in next slides)

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 9