Reinforcement Learning

RL in continuous MDPs

Alberto Maria Metelli

12nd February 2024

Alberto Maria Metelli
Assistant Professor

- Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
  - bulding 21
  - first floor
  - office 19
- Email: `albertomaria.metelli@polimi.it`
- Voice: (+39 02 2399) 9679

Book References

Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction (second edition)
Chapter 9, 10, 11, 12

Csaba Szepesvári
Algorithms for Reinforcement Learning
Section 3.3, 4.3.2

Outline

Outline

Challenges in Real-World Reinforcement Learning

- Large/Continuous **state** space $\mathcal{S}$
  - Tabular representation cannot be employed
- Large/Continuous **action** space $\mathcal{A}$
  - Computation of the $\max$ over the action space is problematic
- Continuous **time**
  - The advantage of individual actions is too small

Examples

- Can Reinforcement Learning (RL) be used to solve **large** problems?
  - Backgammon: 10^{20} states
  - Go: 10^{170} states
  - Autonomous driving, industrial robot control: continuous state-action space (maybe continuous time)
  - ...
- How can we make RL **prediction** and **control** methods scale up?

Value Function Approximation

- So far we have represented value function by a **lookup table**
  - Every state $s \in \mathcal{S}$ has an entry $v(s)$
  - Every state–action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ has an entry $q(s, a)$
- Large/Infinite MDPs:
  - Too many states and/or actions to store in **memory**
  - Too **slow** to learn the value of each state/state-action pair individually

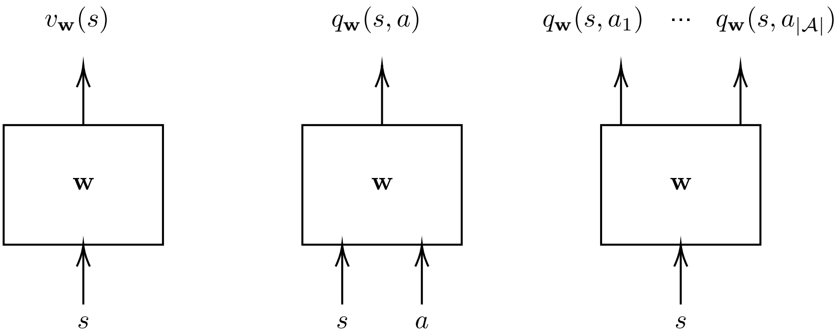
Value Function Approximation

- Solution for large/infinite MDPs
  - Estimate value function with **function approximation**

$$
\begin{aligned}
v_{\mathbf{w}}(s) &\approx v_\pi(s) \text{ or } v_*(s) \\
q_{\mathbf{w}}(s,a) &\approx q_\pi(s,a) \text{ or } q_*(s,a)
\end{aligned}
$$

    where $\mathbf{w} \in \mathbb{R}^n$ is a vector of real parameters
  - **Generalize** from seen states/state-action pairs to unseen states/state-action pairs
  - **Update** parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximators



$v_{\mathbf{w}} : \mathcal{S} \to \mathbb{R}$ $\quad$ $q_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ $\quad$ $q_{\mathbf{w}} : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$

Which Function Approximation?

- There are **many** function approximators, e.g.
  - Linear models
  - (Deep) Neural networks
  - Decision trees
  - Nearest neighbors
  - Fourier/wavelet bases
  - Coarse coding, tile coding
  - ...
- Some are **differentiable** in $\mathbf{w}$ (e.g., neural networks)
- **Linear** vs **non-linear** function approximation

Training Function Approximators

- Training a function approximator is **more challenging** than **supervised learning** (regression)
- Experience is **not i.i.d.**
  - Successive timesteps are **statistically dependent**
  - Agent's action **affect** the subsequent data it receives
- Regression targets can be **non-stationary**
  - In TD evaluation and TD control, the regression target **contains** the function approximator $v_{\mathbf{w}}(s)$
  - In control, the policy π **changes** during learning (and so $v_\pi(s)$)

Outline

Gradient Descent

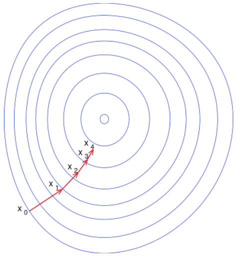
- Let $L : \mathbb{R}^n \to \mathbb{R}$ be a **differentiable** function of parameter vector $\mathbf{w} \in \mathbb{R}^n$
- The **gradient** of $L(\mathbf{w})$ is defined as

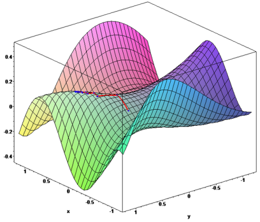
$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \begin{pmatrix} \frac{\partial L(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial L(\mathbf{w})}{\partial w_n} \end{pmatrix}$$

- To find a **local minimum** of $L(\mathbf{w})$
- Adjust the parameter $\mathbf{w}$ in the direction of **negative gradient**

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \qquad \Delta\mathbf{w} = -\alpha \nabla_{\mathbf{w}} L(\mathbf{w})$$

where $\alpha > 0$ is a **stepsize** (or **learning rate**) parameter





Value Function Approximation by Stochastic Gradient Descent

- **Goal**: find parameter vector $\mathbf{w} \in \mathbb{R}^n$ **minimizing** mean square error between approximate value function $v_{\mathbf{w}}(s)$ and true value function $v_\pi(s)$

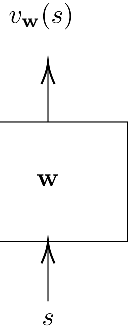
$$L(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{S\sim d_\pi}[(v_\pi(S) - v_{\mathbf{w}}(S))^2],$$

  where d_π is a distribution over states induced by policy π
- Gradient descent finds a **local** minimum

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -\mathbb{E}_{S\sim d_\pi}[(v_\pi(S) - v_{\mathbf{w}}(S))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S)]$$

- Stochastic gradient descent **samples** the gradient

$$\Delta\mathbf{w} = \alpha(v_\pi(s) - v_{\mathbf{w}}(s))\nabla_{\mathbf{w}} v_{\mathbf{w}}(s)$$



Outline

Linear Value Function Approximation

- Represent value function by a **linear combination** of features

$$v_{\mathbf{w}}(s) = \mathbf{x}(s)^{\intercal}\mathbf{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- Objective function is **quadratic** in $\mathbf{w}$

$$L(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{S\sim d_\pi}\left[(v_\pi(S) - \mathbf{x}(S)^{\intercal}\mathbf{w})^2\right]$$

- Stochastic gradient descent converges to **global** optimum
- **Update rule** is particularly simple

$$\begin{aligned} \nabla_{\mathbf{w}} v_{\mathbf{w}}(s) &= \mathbf{x}(s) \\ \Delta\mathbf{w} &= \alpha(v_\pi(s) - v_{\mathbf{w}}(s))\mathbf{x}(s) \end{aligned}$$

- Update $=$ stepsize $\times$ prediction error $\times$ feature value

Table Lookup Features

- Table lookup is a **special case** of linear value function approximation
- Using table lookup features

$$\mathbf{x}^{\text{table}}(s) = \begin{pmatrix} \mathbf{1}\{s = s_1\} \\ \vdots \\ \mathbf{1}\{s = s_{|\mathcal{S}|}\} \end{pmatrix}$$

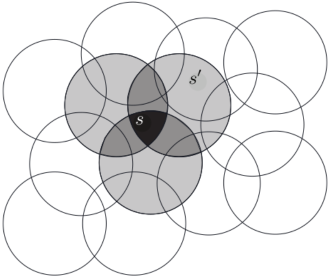
- Parameter vector $\mathbf{w}$ gives value of **each individual state**

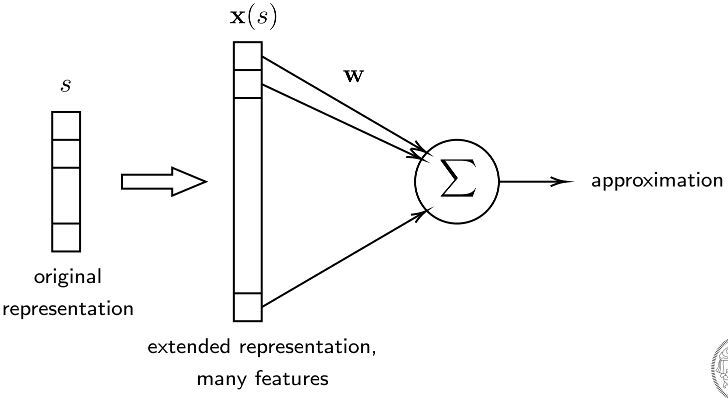
$$v_{\mathbf{w}}(s) = \mathbf{x}^{\text{table}}(s)^{\mathtt{T}}\mathbf{w} = \begin{pmatrix} \mathbf{1}\{s = s_1\} \\ \vdots \\ \mathbf{1}\{s = s_{|\mathcal{S}|}\} \end{pmatrix}^{\mathtt{T}} \begin{pmatrix} w_1 \\ \vdots \\ w_{|\mathcal{S}|} \end{pmatrix}$$

Coarse Coding

Example of linear value function approximation (Hinton, 1984):

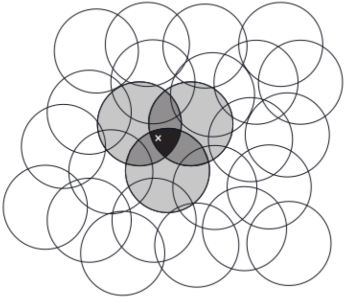
- Coarse coding provides **large** feature vector $\mathbf{x}(s)$
- Parameter vector $\mathbf{w}$ gives a value to **each feature**



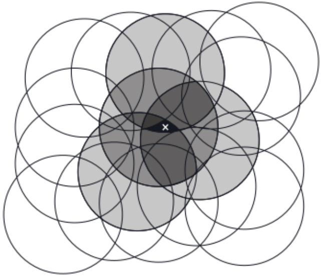


Pictures from (Sutton and Barto, 2018)

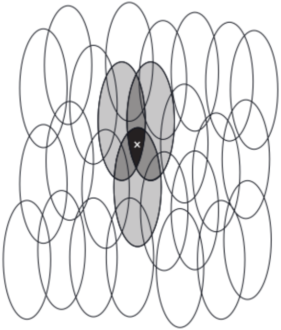
Generalization in Coarse Coding



Narrow generalization



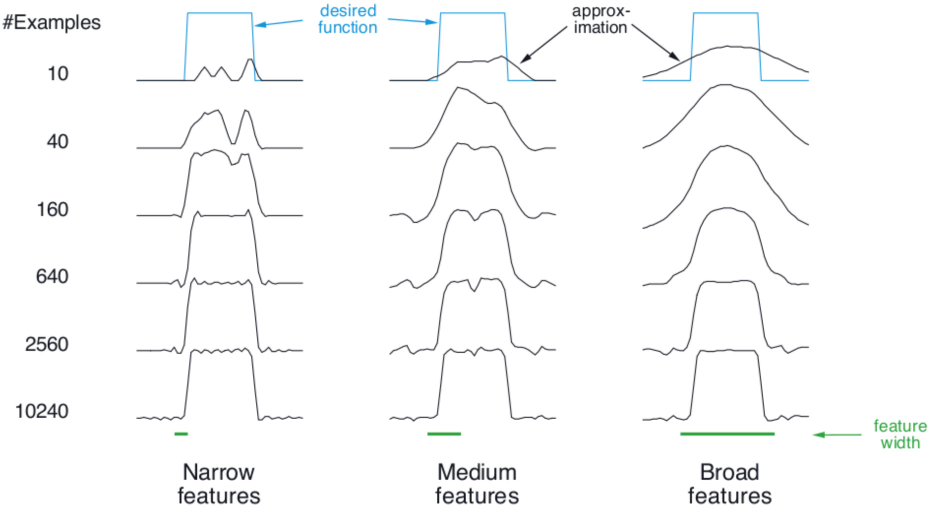
Broad generalization



Asymmetric generalization

Pictures from (Sutton and Barto, 2018)

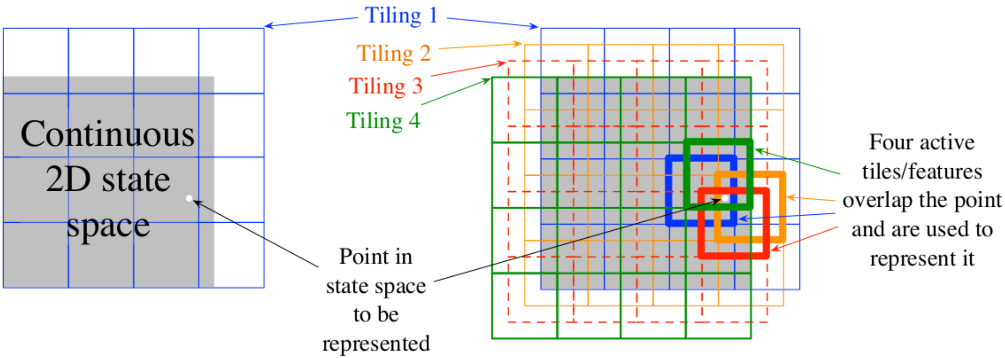
Stochastic Gradient Descent with Coarse Coding



Pictures from (Sutton and Barto, 2018)

Tile Coding

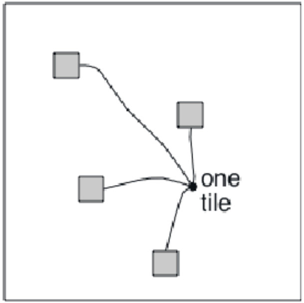
- **Binary feature** for each tile (Albus, 1971, 1981)
- Number of features active at any one time is **constant**
- Binary features means weighted sum **easy to compute**
- Easy to compute **indices** of the active features



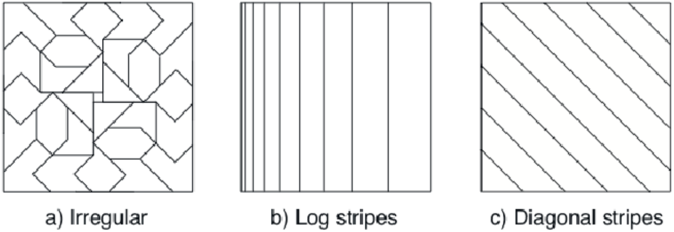
Pictures from (Sutton and Barto, 2018)

Tile Coding

- Hashing



- Irregular tilings

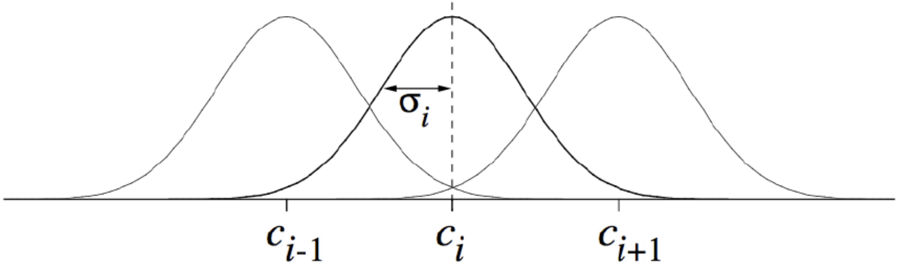


Pictures from (Sutton and Barto, 2018)

Radial Basis Functions (RBFs)

e.g., Gaussians $s \in \mathbb{R}^d$, centers $c_i \in \mathbb{R}^d$, bandwidths $\sigma_i > 0$ (Poggio and Girosi, 1989, 1990)

$$x_i(s) = \exp\left(-\frac{\|s - c_i\|_2^2}{2\sigma_i^2}\right)$$



Pictures from (Sutton and Barto, 2018)

Outline

Definitions: MC and n-step Returns

- Consider a **trajectory** of length T: $(S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$
- The **complete/MC return** is defined as:

$$G_t = \sum_{l=t}^{T-1} \gamma^{l-t} R_{l+1}$$

- Given an estimate of the value function $v_{\mathbf{w}}(s)$, the n-**step return** is defined as

$$G_{t:t+n} = \sum_{l=t}^{t+n-1} \gamma^{l-t} R_{l+1} + \gamma^n v_{\mathbf{w}}(S_{t+n})$$

  - If $n = 1$, we have the **TD(0) target**:

$$G_{t:t+1} = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$$

Definitions: λ-returns

- Consider a **trajectory** of length T: $(S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$
- Given an estimate of the value function $v_{\mathbf{w}}(s)$, the λ**-return** is defined as

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

  - If $\lambda = 0$, we have the **TD(0) target**:

$$G_t^0 = G_{t:t+1} = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$$

  - If $\lambda = 1$, we have the **MC return**:

$$G_t^1 = G_t$$

Incremental Prediction Algorithm

- So far, we have assumed true value function $v_\pi(s)$ given by **supervisor**

$$\Delta \mathbf{w} = \alpha(v_\pi(S_t) - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

- But in RL there is **no supervisor**, only rewards
- In practice, we substitute a **target** for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

  - For TD(λ), the target is the λ–return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

Gradient Monte Carlo Prediction

Monte–Carlo with Value Function Approximation

- The return G_t is an **unbiased**, noisy sample of true value $v_\pi(S_t)$

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

- Can therefore apply **supervised learning** to "training data":

$$\{(S_1, G_1), (S_2, G_2), \dots, (S_{T-1}, G_{T-1})\}$$

- For example, using **linear** Monte–Carlo **policy evaluation**

$$\begin{aligned} \Delta \mathbf{w} &= \alpha({\color{red}G_t} - v_{\mathbf{w}}(S_t)) \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) \\ &= \alpha(G_t - v_{\mathbf{w}}(S_t)) \mathbf{x}(S_t) \end{aligned}$$

Semi-Gradient TD(0) Prediction

TD Learning with Value Function Approximation

- The TD–target $R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$ is a **biased** sample of true value $v_\pi(S_t)$ (Sutton, 1984, 1988)

$$\mathbb{E}[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})|\mathbf{w}, S_t] \neq v_\pi(S_t)$$

- Can still apply supervised learning to "training data":

$$\{(S_1, R_2 + \gamma v_{\mathbf{w}}(S_2)), (S_2, R_3 + \gamma v_{\mathbf{w}}(S_3)), \dots, (S_{T-1}, R_T)\}$$

- For example, using **linear TD(0)**

$$\Delta \mathbf{w} \quad = \quad \alpha \underbrace{(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t))}_{\delta_t \text{ - TD Error}} \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) = \alpha \delta_t \mathbf{x}(S_t)$$

- Targets $R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$
  - are **non-stationary** as $\mathbf{w}$ changes during training
  - depend **explicitly** on the optimization variable $\mathbf{w}$
- This is **not** the gradient of **any objective function** $\rightarrow$ **semi-gradient**

Semi-Gradient TD(λ) Prediction

TD(λ) with Value Function Approximation

- The λ–return G_t^λ is also a **biased** sample of true value $v_\pi(s)$ (Sutton, 1984, 1988)
- Can again apply supervised learning to "training data":

$$\{(S_1, G_1^\lambda), (S_2, G_2^\lambda), \ldots, (S_{T-1}, G_{T-1}^\lambda)\}$$

- **Forward view** linear TD(λ)

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(G_t^\lambda - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) \\ &= \alpha(G_t^\lambda - v_{\mathbf{w}}(S_t))\mathbf{x}(s_t) \end{aligned}$$

- **Backward view** linear TD(λ)

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t) \\ \boldsymbol{z}_t &= \gamma\lambda \boldsymbol{z}_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t \boldsymbol{z}_t \end{aligned}$$

- Forward view and backward view linear TD(λ) are **equivalent**

Outline

Off-Policy Prediction with Function Approximation

- So far, we wanted to estimate $v_\pi(s)$ having samples collected with the **same policy** π
- What if we have samples collected with a **behavior** policy b and we want to estimate $v_\pi(s)$ for a **target** policy π?
- For **prediction**, we can use **importance sampling** (Owen, 2013)

Importance Sampling

- Our **goal** is to
  - **estimate** the expectation of a **function** $f : \mathcal{X} \to \mathbb{R}$
  - under a **target** distribution P, i.e., $\mathbb{E}_{x \sim P}[f(x)]$
  - having i.i.d. samples $\{X_1, X_2, \dots, X_T\}$ collected with a **behavior** distribution Q
- **Importance sampling** reweights every sample with the **likelihood ratio** $\rho(x) = \frac{P(x)}{Q(x)}$

$$\widehat{\mu}_{\text{IS}} = \frac{1}{T} \sum_{i=1}^{n} \rho(X_i) f(X_i) = \frac{1}{T} \sum_{i=1}^{n} \frac{P(X_i)}{Q(X_i)} f(X_i)$$

- $\widehat{\mu}_{\text{IS}}$ is **unbiased**

$$\mathbb{E}_{x_i \sim Q}[\widehat{\mu}_{\text{IS}}] = \mathbb{E}_{X \sim Q}\left[\frac{P(X)}{Q(X)} f(X)\right] = \int_{\mathcal{X}} Q(x) \frac{P(x)}{Q(x)} f(x) \mathrm{d}x = \int_{\mathcal{X}} P(x) f(x) \mathrm{d}x = \mathbb{E}_{X \sim P}[f(X)]$$

- but it might suffer from **large variance** (sometimes infinite) (Metelli et al., 2018)

Incremental Off-Policy Prediction Algorithm

- π **target** policy and b **behavior** policy (Precup et al., 2001)
- Define the **importance weight**

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

- In practice, we substitute a **target** for $v_\pi(s)$ and perform **importance weighting**
  - For MC, the target is the return G_t multiplied by the product of ratios

$$\Delta \mathbf{w} = \alpha \left(\prod_{l=t}^{T-1} \rho_l \right) (G_t - v_{\mathbf{w}}(S_t)) \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$ multiplied by the ratio ρ_t

$$\Delta \mathbf{w} = \alpha \rho_t (R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)) \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

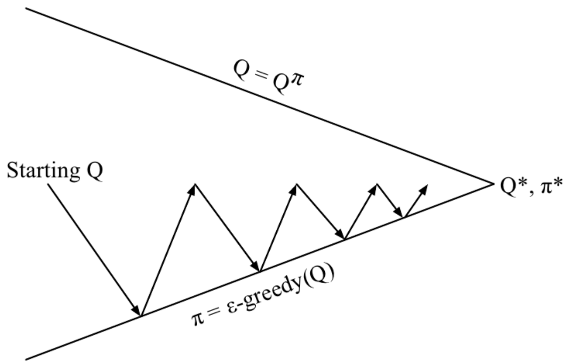
  - For forward-view TD(λ), the target is the λ–return G_t^λ multiplied by the product of ratios

$$\Delta \mathbf{w} = \alpha \left(\prod_{l=t}^{T-1} \rho_l \right) \left(G_t^\lambda - v_{\mathbf{w}}(s) \right) \nabla_{\mathbf{w}} V_{\mathbf{w}}(s)$$

- A lot of variations: **per-decision** importance sampling, self-normalized, control variate (Precup, 2000)

Outline

Control with Value Function Approximation



- **Policy Evaluation**: Approximate policy evaluation, $q_{\mathbf{w}} \approx q_\pi$
- **Policy Improvement**: ϵ–greedy policy improvement

Pictures from (Sutton and Barto, 2018)

Action–Value Function Approximation

- **Approximate** the action-value function

$$q_{\mathbf{w}}(s,a) \approx q_\pi(s,a) \text{ or } q_*(s,a)$$

- **Minimize** the mean square error between approximate action–value function $q_{\mathbf{w}}(s,a)$ and true action–value function $q_\pi(s,a)$

$$L(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi(\cdot|S)}}\left[(q_\pi(S,A) - q_{\mathbf{w}}(S,A))^2\right]$$

- The **full gradient** is given by

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi(\cdot|S)}}\left[(q_\pi(S,A) - q_{\mathbf{w}}(S,A))\nabla_{\mathbf{w}} q_{\mathbf{w}}(S,A)\right]$$

- Use **stochastic gradient descent** to find local minimum

$$\Delta\mathbf{w} = \alpha(q_\pi(s,a) - q_{\mathbf{w}}(s,a))\nabla_{\mathbf{w}} q_{\mathbf{w}}(s,a)$$

Linear Action–Value Function Approximation

First Possibility

- Represent state and action by a **feature vector**

$$\mathbf{x}(s,a) = \begin{pmatrix} x_1(s,a) \\ \vdots \\ x_n(s,a) \end{pmatrix} \in \mathbb{R}^n$$

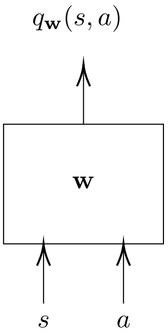
- Represent action–value function by **linear combination of features**

$$q_{\mathbf{w}}(s,a) = \mathbf{x}(s,a)^{\mathsf{T}}\mathbf{w} = \sum_{j=1}^{n} x_j(s,a) w_j, \qquad \mathbf{w} \in \mathbb{R}^n$$

- Stochastic gradient descent **update**

$$\begin{aligned} \nabla_{\mathbf{w}} q_{\mathbf{w}}(s,a) &= \mathbf{x}(s,a) \\ \Delta\mathbf{w} &= \alpha(q_\pi(s,a) - q_{\mathbf{w}}(s,a))\mathbf{x}(s,a) \end{aligned}$$

- Works even with **infinite** actions



Linear Action–Value Function Approximation

Second Possibility

- If number of actions is **finite** $|\mathcal{A}| < +\infty$
- Represent state by a **feature vector**

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix} \in \mathbb{R}^n$$

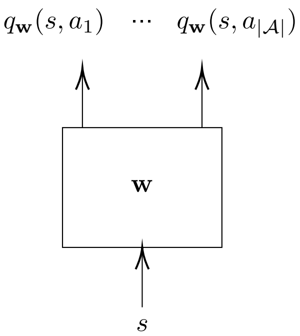
- Represent action–value function by **linear combination of features**

$$q_{\mathbf{w}}(s, \cdot) = \mathbf{W}\mathbf{x}(s) = \sum_{j=1}^{n} x_j(s)\mathbf{W}_{\cdot,j}, \qquad \mathbf{W} \in \mathbb{R}^{|\mathcal{A}| \times n}$$

- Stochastic gradient descent **update**

$$\begin{aligned} \nabla_{\mathbf{W}_{i,\cdot}} q_{\mathbf{W}}(s, a_k) &= \mathbf{1}\{i = k\}\mathbf{x}(s) \\ \Delta\mathbf{W}_{i,\cdot} &= \alpha(q_\pi(s, a_k) - q_{\mathbf{W}}(s, a_k))\mathbf{1}\{i = k\}\mathbf{x}(s) \end{aligned}$$

- Can be represented with $\mathbf{x}(s, a_i) = (\mathbf{x}(s)^\mathsf{T}\mathbf{1}\{i = 1\}| \dots |\mathbf{x}(s)^\mathsf{T}\mathbf{1}\{i = |\mathcal{A}|\})^\mathsf{T}$



Incremental Control Algorithms

- Like prediction, we must substitute a **target** for $q_\pi(s, a)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - q_{\mathbf{w}}(S_t, A_t))\mathbf{x}(S_t, A_t)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma q_{\mathbf{w}}(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - q_{\mathbf{w}}(S_t, A_t))\mathbf{x}(S_t, A_t)$$

  - For forward–view TD(λ), target is λ–return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - q_{\mathbf{w}}(S_t, A_t))\mathbf{x}(S_t, A_t)$$

  - For backward–view TD(λ), equivalent update is

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - q_{\mathbf{w}}(S_t, A_t) \\ \mathbf{z}_t &= \gamma\lambda \mathbf{z}_{t-1} + \mathbf{x}(S_t, A_t) \\ \Delta \mathbf{w} &= \alpha \delta_t \mathbf{z}_t \end{aligned}$$

Semi-Gradient SARSA(0) for Control

- We need an **exploration policy** that selects the action based on $q_{\mathbf{w}}$ (e.g., ϵ-greedy, Boltzmann, ...) (Rummery and Niranjan, 1994)

Initialize $\mathbf{w}$ arbitrarily
loop for each episode
  $S, A \leftarrow$ initial state and action using the exploration policy
  loop for each step of episode
    Take action A, observe reward R, and next state S'
    if S' is terminal **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R - q_{\mathbf{w}}(S, A)\right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S, A)$
      break
    end if
    Choose A' using the exploration policy
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R + \gamma q_{\mathbf{w}}(S', A') - q_{\mathbf{w}}(S, A)\right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S, A)$
    $S \leftarrow S'$
    $A \leftarrow A'$
  end loop
end loop

Semi-Gradient SARSA for Control

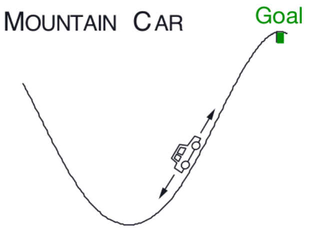
- Many variations are possible:
  - **Expected** SARSA(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R + \gamma \underbrace{\sum_{a' \in \mathcal{A}} \pi(a'|S') q_{\mathbf{w}}(S', a')}_{\text{replaces } q_{\mathbf{w}}(S', A') \text{ of SARSA(0)}} - q_{\mathbf{w}}(S, A) \right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S, A)$$

    where π is the exploration policy
  - SARSA(n) with n-step returns
  - SARSA(λ) with λ-returns (van Seijen, 2016)

Mountain Car Task

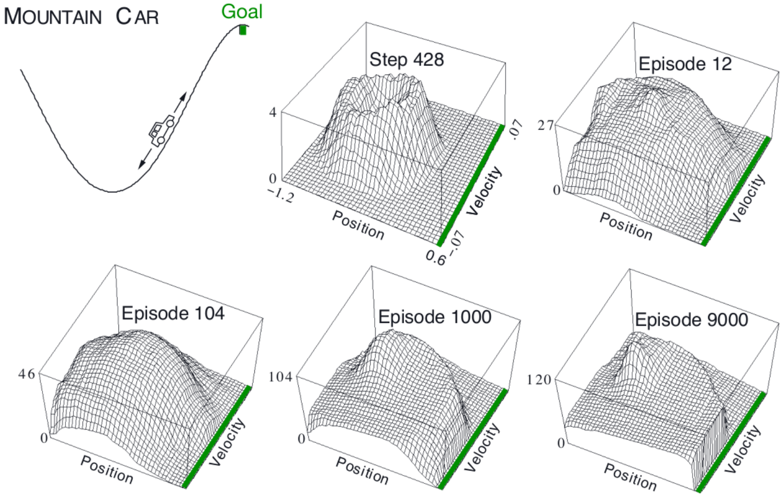
- **Goal**: drive an underpowered car up a steep mountain road
- **Optimal policy**: first, move up the opposite slope on the left, then, full throttle to reach goal
- **Actions**: full throttle forward $(+1)$, full throttle reverse (-1), and zero throttle (0)
- **Reward**: -1 if not at goal
- **Features** $\mathbf{x}(s)$ 8 tiles over the state space: (position, velocity)



Pictures from (Sutton and Barto, 2018)

Linear Semi-Gradient SARSA(0) with Tile Coding in Mountain Car

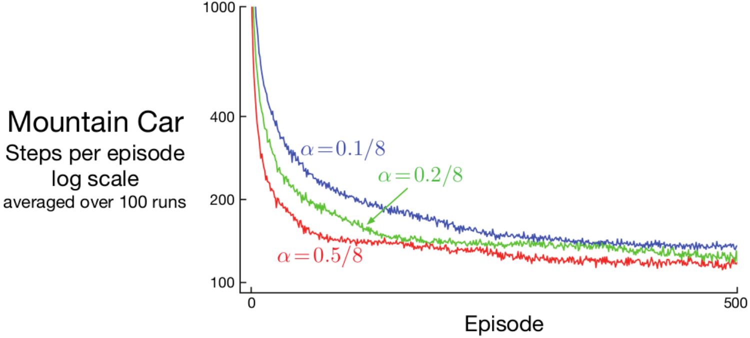
Negative of the value function $-\max_{a \in \mathcal{A}} q_{\mathbf{w}}(s, a)$



Pictures from (Sutton and Barto, 2018)

Linear Semi-Gradient SARSA(0) with Tile Coding in Mountain Car

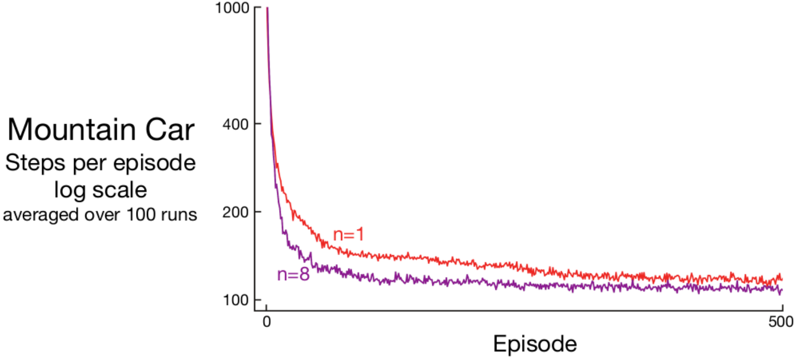
Varying the learning rate α



Pictures from (Sutton and Barto, 2018)

Linear Semi-Gradient SARSA(n) with Tile Coding in Mountain Car

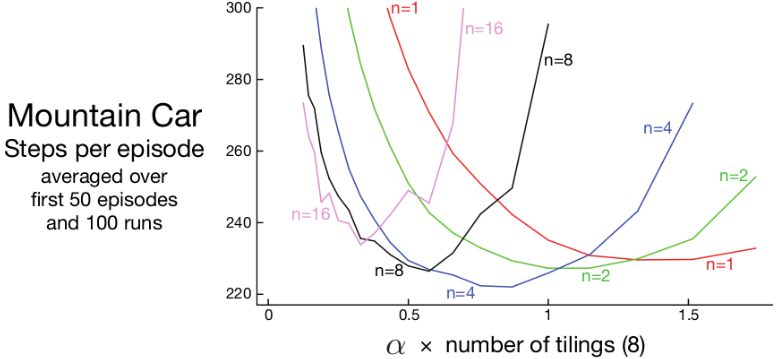
Varying the number of steps for TD(n) with $\alpha = 0.5/8$ for $n = 1$ and $\alpha = 0.3/8$ for $n = 8$



Pictures from (Sutton and Barto, 2018)

Linear Semi-Gradient SARSA(n) with Tile Coding in Mountain Car

Varying α and n



Pictures from (Sutton and Barto, 2018)

Outline

Incremental Off-Policy Control

- Directly learn $q_*(s, a)$ while executing an **exploration** policy
- We need to define a **target** for $q_*(s, a)$
  - For Q-learning, the target is $R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_{\mathbf{w}}(S_{t+1}, a')$

$$\Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_{\mathbf{w}}(S_{t+1}, a') - q_{\mathbf{w}}(S_t, A_t) \right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S_t, A_t)$$

  - Possible variations multi-step returns

Semi-Gradient Q-Learning for Control

- We need an **exploration policy** that selects the action based on $q_{\mathbf{w}}$ (e.g., ϵ-greedy, Boltzmann, or even uniform...)

Initialize $\mathbf{w}$ arbitrarily
loop for each episode
    $S, A \leftarrow$ initial state and action using the exploration policy
    loop for each step of episode
        Take action A, observe reward R, and next state S'
        if S' is terminal **then**
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R - q_{\mathbf{w}}(S, A)\right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S, A)$
            break
        end if
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R + \gamma \max_{a' \in \mathcal{A}} q_{\mathbf{w}}(S', a') - q_{\mathbf{w}}(S, A)\right) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S, A)$
        Choose A' using the exploration policy
        $S \leftarrow S'$
        $A \leftarrow A'$
    end loop
end loop

Summary

	Dynamic Programming	Table Lookup	Function Approximation
	Planning (no samples)	Learning (samples)	
	Finite $\mathcal{S}$ and $\mathcal{A}$		Infinite $\mathcal{S}$
Prediction	Iterative Policy Evaluation $v(S) \leftarrow \underset{\substack{A \sim \pi(\cdot\|S) \\ S' \sim P(\cdot\|S,A)}}{\mathbb{E}} [r(S,A) + \gamma v(S')]$	TD(0) Prediction $v(S) \overset{\alpha}{\leftarrow} R + \gamma v(S')$	Semi-gradient TD(0) Prediction $v_{\mathbf{w}}(S) \overset{\alpha}{\leftarrow} (R + \gamma v_{\mathbf{w}}(S')) \nabla_{\mathbf{w}} v_{\mathbf{w}}(S')$
Control	Q-Policy Iteration $q(S,A) \leftarrow \underset{\substack{S' \sim P(\cdot\|S,A) \\ A' \sim \pi(\cdot\|S')}}{\mathbb{E}} [r(S,A) + \gamma q(S',A')]$ $+$ greedy improvement	SARSA(0) (**on-policy**) $q(S,A) \overset{\alpha}{\leftarrow} R + \gamma q(S',A')$ $+$ e.g., ϵ-greedy improvement	Semi-gradient SARSA(0) (**on-policy**) $q_{\mathbf{w}}(S,A) \overset{\alpha}{\leftarrow} (R + \gamma q_{\mathbf{w}}(S',A')) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S',A')$ $+$ e.g., ϵ-greedy improvement
Control	Q-Value Iteration $q(S,A) \leftarrow \underset{S' \sim p(\cdot\|S,A)}{\mathbb{E}} [r(S,A) + \gamma \max_{a' \in \mathcal{A}} q(S',A')]$	Q-learning (**off-policy**) $q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} q(S',a')$	Semi-gradient Q-learning (**off-policy**) $q_{\mathbf{w}}(S,A) \overset{\alpha}{\leftarrow} (R + \gamma \max_{a' \in \mathcal{A}} q_{\mathbf{w}}(S',A')) \nabla_{\mathbf{w}} q_{\mathbf{w}}(S',a')$

- What about $|\mathcal{A}| = \infty$?

Outline

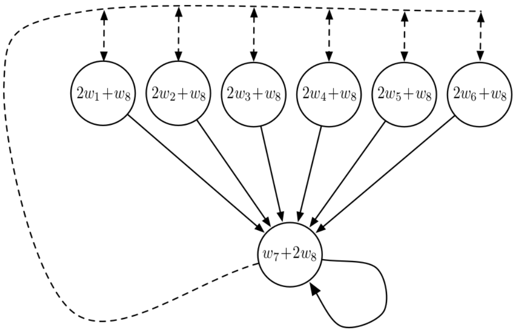
Convergence Questions

- When do incremental prediction algorithms **converge**?
  - When using **bootstrapping** (i.e., TD with $\lambda < 1$)?
  - When using **linear** function approximation?
  - When using **off–policy** learning?
- We want to understand **under which conditions** the algorithms converge

Outline

Baird's Counterexample

- Episodic MDP, 7 states, 2 actions (Baird, 1995)
  - Dashed action takes the system to one of the six upper states with equal probability
  - Solid action takes the system to the seventh state
  - Initial state uniform
  - Zero reward everywhere
- Linear approximation $v_{\mathbf{w}}(s) = \mathbf{x}(s)^{\mathsf{T}}\mathbf{w}$ with **linearly independent** features
- True value function $v_\pi(s) = 0$ representable with $\mathbf{w} = \mathbf{0}$



$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

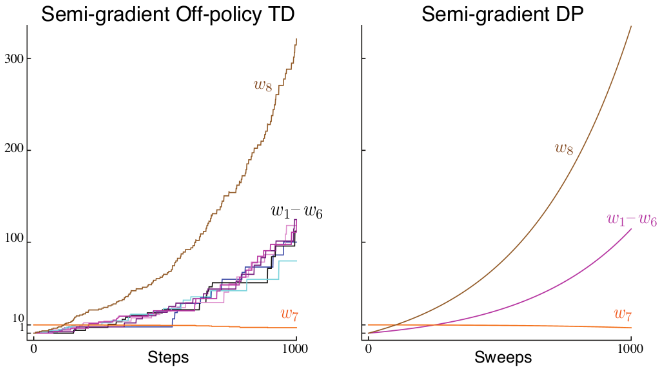
$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

Pictures from (Sutton and Barto, 2018)

Parameter Divergence in Baird's Counterexample

- **Off-policy** visitation distribution under b is **uniform**
- Initial weights $\mathbf{w}_0 = (1, 1, 1, 1, 1, 1, 10, 1)^\mathsf{T}$
- Weights **diverge** to infinity
- Even if we perform a **semi-gradient DP** update!



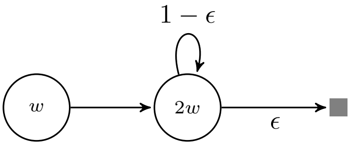
Pictures from (Sutton and Barto, 2018)

Baird's Counterexample Fixes

- Baird's counterexample has two simple **fixes** (Tsitsiklis and Van Roy, 1997):
  - use the **on–policy** distribution induced by π
  - Instead of taking small steps towards expected one–step returns, change value function to the best **least–squares approximation**
- This works if the feature vectors $\{\mathbf{x}(s) : s \in \mathcal{S}\}$ form a **linearly independent set**
  - Then, the **least–squares solution** is **exact** (zero error)

Tsitsiklis and Van Roy's Counterexample

- When the **exact least–squares solution** does not exists, it does not work even if we consider the **best approximation** at each iteration
  - 0 reward everywhere
  - True value function representable with $w = 0$
  - $\{\mathbf{x}(s) \,:\, s \in \mathcal{S}\} = \{1, 2\}$ is **not linearly independent**!



$$
\begin{aligned}
w_{k+1} &= \arg\min_{w \in \mathbb{R}} \sum_{s \in \mathcal{S}} \left(v_w(s) - \mathbb{E}_\pi [R_{t+1} + \gamma v_{w_k}(S_{t+1}) | S_t = s] \right)^2 \\
&= \arg\min_{w \in \mathbb{R}} [w - 2\gamma w_k]^2 + [2w - 2(1-\epsilon)\gamma w_k]^2 = \frac{6-4\epsilon}{5} \gamma w_k
\end{aligned}
$$

- The sequence w_k diverge s when $\gamma > \frac{5}{6-4\epsilon}$ and $w_0 \neq 0$

Outline

Notation

Let $v : \mathcal{S} \to \mathbb{R}$:

- $L_2(\mu)$-norm:

$$\|v\|_\mu^2 = \mathbb{E}_{S\sim\mu}\left[v(S)^2\right]$$

- $L_2(\mu)$-projection operator:

$$v_{\mathbf{w}} = \Pi v \qquad \text{where} \qquad \mathbf{w} \in \arg\min_{\mathbf{w}\in\mathbb{R}^n} \|v - v_{\mathbf{w}}\|_\mu^2$$

- Bellman operator $B_\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ induced by policy π:

$$(B_\pi v)(s) = \mathbb{E}_{\substack{A\sim\pi(\cdot|s)\\ S'\sim P(\cdot|s,A)}}\left[r(s,A) + \gamma v(S')\right]$$

- TD error

$$\delta_{\mathbf{w}}(s,a,s') = r(s,a) + \gamma v_{\mathbf{w}}(s') - v_{\mathbf{w}}(s)$$

- Bellman error

$$\overline{\delta}_{\mathbf{w}}(s) = (B_\pi v_{\mathbf{w}})(s) - v_{\mathbf{w}}(s) = \mathbb{E}_{\substack{A\sim\pi(\cdot|s)\\ S'\sim P(\cdot|s,A)}}\left[\delta_{\mathbf{w}}(s,A,S')\right]$$

The Landscape of Objectives

- Mean square value error

$$\overline{\mathsf{VE}}(\mathbf{w}) = \|v_\pi - v_\mathbf{w}\|_\mu^2 = \mathbb{E}_{S\sim\mu}\left[(v_\pi(S) - v_\mathbf{w}(S))^2\right]$$

- Mean square Bellman error

$$\overline{\mathsf{BE}}(\mathbf{w}) = \left\|\bar{\delta}_\mathbf{w}\right\|_\mu^2 = \|B_\pi v_\mathbf{w} - v_\mathbf{w}\|_\mu^2$$

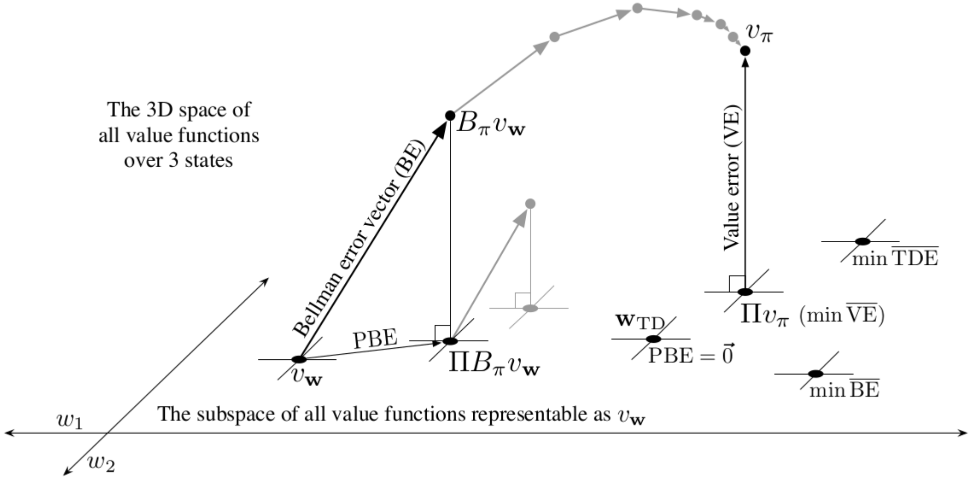
- Mean square projected Bellman error

$$\overline{\mathsf{PBE}}(\mathbf{w}) = \left\|\Pi\bar{\delta}_\mathbf{w}\right\|_\mu^2 = \|\Pi B_\pi v_\mathbf{w} - v_\mathbf{w}\|_\mu^2$$

- Mean square TD error

$$\overline{\mathsf{TDE}}(\mathbf{w}) = \mathbb{E}_{S\sim\mu}\left[\mathbb{E}_{\substack{A\sim\pi(\cdot|S)\\ S'\sim P(\cdot|S,A)}}\left[\delta_\mathbf{w}(S,A,S')^2\right]\right]$$

The Landscape of Objectives



Pictures from (Sutton and Barto, 2018)

Convergence of Gradient Monte Carlo Prediction

- Gradient Monte Carlo Prediction is minimizing the **mean square value error** $\overline{\mathsf{VE}}(\mathbf{w})$
- Under Robbins-Monro conditions on the learning rate α (e.g., $\alpha_t = 1/t$):

$$\sum_{t=1}^{+\infty} \alpha_t = +\infty \qquad \text{and} \qquad \sum_{t=1}^{+\infty} \alpha_t^2 < +\infty$$

  - Convergence to a **local optimum** when using **non–linear** value function approximation
  - Convergence to the **global optimum** when using **linear** value function approximation

$$\min_{\mathbf{w}\in\mathbb{R}^n} \overline{\mathsf{VE}}(\mathbf{w}) = \mathbb{E}_{S\sim d_\pi}\left[\left(v_\pi(S) - \mathbf{x}(S)^\intercal \mathbf{w}\right)^2\right]$$

$$\mathbf{w}^{\mathsf{MC}} = \mathbb{E}\left[\mathbf{x}\mathbf{x}^\intercal\right]^{-1}\mathbb{E}\left[\mathbf{x}v_\pi\right]$$

- Convergence proof follows from standard stochastic approximation
- Works even if the problem is **not Markovian**

Semi-Gradient TD(0) Prediction

- Semi-Gradient TD(0) **does not converge** in general for **non-linear** value function approximation
- Under Robbins-Monro conditions, Semi-Gradient TD(0) with **linear** value function approximation **converges** to the **TD fixed point**

$$\mathbf{w}^{\mathsf{TD}} = \mathbb{E}\left[\mathbf{x}(\mathbf{x} - \gamma \mathbf{x}')^{\mathsf{T}}\right]^{-1} \mathbb{E}\left[\mathbf{x}R\right]$$

  - Some additional technical assumptions (e.g., on-policy distribution, ...) (Tsitsiklis and Van Roy, 1997)
- $\mathbf{w}^{\mathsf{TD}}$ minimizes the **mean square projected Bellman error** $\overline{\mathsf{PBE}}$
- and it is "close" to $\mathbf{w}^{\mathsf{MC}}$:

$$\overline{\mathsf{VE}}(\mathbf{w}^{\mathsf{TD}}) \leq \frac{1}{1-\gamma} \overline{\mathsf{VE}}(\mathbf{w}^{\mathsf{MC}})$$

- but TD **does not** follow the **gradient** of any objective function!
- This is why TD(0) can diverge when off–policy or using non-linear function approximation

Convergence of Prediction Algorithms

On/Off–Policy	Algorithm	Table Lookup	Linear	Non–Linear
On–Policy	MC	OK	OK	OK
	TD(0)	OK	OK	KO
Off–Policy	MC	OK	OK	OK
	TD(0)	OK	KO	KO

Convergence point of Semi-gradient TD(0)

Exercise 1

Consider the semi-gradient TD(0) algorithm with **linear** function approximation (abbreviations $\mathbf{x} = \mathbf{x}(S)$ and $\mathbf{x}' = \mathbf{x}(S')$):

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha \left(R + \gamma \mathbf{w}_k^\intercal \mathbf{x}' - \mathbf{w}_k^\intercal \mathbf{x} \right) \mathbf{x}, \tag{1}$$

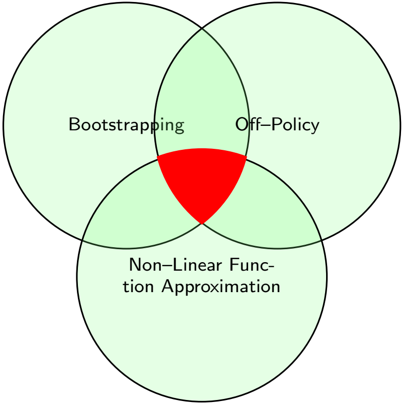
assuming that $S \sim d$ where d is some distribution over the state space. Prove that if the iterate (1) converges, then the convergence point in expectation is:

$$\mathbf{w}^{\mathsf{TD}} = \mathbb{E}\left[\mathbf{x} \left(\mathbf{x} - \gamma \mathbf{x}' \right)^\intercal \right]^{-1} \mathbb{E}\left[\mathbf{x} R \right],$$

where the expectation is w.r.t. $S \sim d$.

Outline

The Deadly Triad



We have not **quite** achieved our ideal goal for prediction algorithms (Sutton, 1995)

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-Linear
Monte–Carlo Control	OK	(OK)	KO
SARSA	OK	(OK)	KO
Q–learning	OK	KO	KO

(OK) $=$ **chatters** around near–optimal value function (Gordon, 1995)

Outline

Why not computing the full gradient for the TD(0) prediction?

- First compute the gradient, then replace v_π with the TD target $\rightarrow$ **Semi-gradient TD(0)**

$$\nabla_{\mathbf{w}} \frac{1}{2}\left(v_\pi(S_t) - v_{\mathbf{w}}(S_t)\right)^2$$

$$\xrightarrow{v_\pi(S_t)=R_{t+1}+\gamma v_{\mathbf{w}}(S_{t+1})} \Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)\right) \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$$

  - Gradient of no objective function
  - Convergence to the minimum of the $\overline{\text{PBE}}(\mathbf{w})$
  - Linear case $\rightarrow \mathbf{w} = \mathbb{E}[\mathbf{x}(\mathbf{x} - \gamma \mathbf{x}')^{\mathtt{T}}]^{-1}\mathbb{E}[\mathbf{x}R]$
- First replace v_π with the TD target, then compute the gradient $\rightarrow$ **Naïve residual-gradient** (Baird, 1995)

$$\nabla_{\mathbf{w}} \frac{1}{2}\left(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)\right)^2 = \nabla_{\mathbf{w}} \frac{1}{2} \underbrace{\delta^2_{\mathbf{w},t}}_{(\text{TD error})^2}$$

$$\rightarrow \Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)\right) \left(\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) - \gamma \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_{t+1})\right)$$

  - Convergence to the minimum of the $\overline{\text{TDE}}(\mathbf{w})$
  - Not necessarily a good solution! See Example 11.2 from (Sutton and Barto, 2018)
  - Linear case $\rightarrow \mathbf{w} = \mathbb{E}[(\mathbf{x} - \gamma \mathbf{x}')(\mathbf{x} - \gamma \mathbf{x}')^{\mathtt{T}}]^{-1}\mathbb{E}[(\mathbf{x} - \gamma \mathbf{x}')R]$

What is the Right Objective for TD Prediction?

- We might want to be as close as possible to **Semi-Gradient TD(0)**

$$\Delta \mathbf{w} = \alpha \delta_{t,\mathbf{w}} \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) \qquad \delta_{t,\mathbf{w}} = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$$

- Mean square TD error $\overline{\mathsf{TDE}}(\mathbf{w})$? $\rightarrow$ **Naive residual-gradient algorithm**

$$\Delta \mathbf{w} = \alpha \delta_{t,\mathbf{w}} \left(\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) - -\gamma \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_{t+1}) \right)$$

  - Good convergence guarantees
  - Might convergence **far** from v_π
- Mean square Bellman error $\overline{\mathsf{BE}}(\mathbf{w})$? $\rightarrow$ **Residual-gradient algorithm**

$$\Delta \mathbf{w} = \alpha \bar{\delta}_{t,\mathbf{w}} \left(\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) - \gamma \mathbb{E}[\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_{t+1})] \right) \qquad \bar{\delta}_{t,\mathbf{w}} = \mathbb{E}[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t) | S_t]$$

  - Product of expectations $\rightarrow$ **double sampling** problem
  - If so, good convergence, but **empirically slow** and possibly **far** from v_π
  - The Bellman error is **"non-learnable"**

Gradient TD Methods

- If $\mathbf{w}^{\text{TD}}$ optimizes the **mean square projected Bellman error** $\overline{\text{PBE}}$...
- ...what about directly following the **gradient** of $\overline{\text{PBE}}$?

$$\Delta \mathbf{w} = -\alpha \frac{1}{2} \nabla_{\mathbf{w}} \overline{\text{PBE}}(\mathbf{w})$$

- With **linear function approximation** we obtain

$$\frac{1}{2} \nabla_{\mathbf{w}} \overline{\text{PBE}}(\mathbf{w}) = \mathbb{E}[(\gamma \mathbf{x}' - \mathbf{x})\mathbf{x}^{\mathtt{T}}] \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathtt{T}}]^{-1} \mathbb{E}[\delta_{\mathbf{w}} \mathbf{x}]$$

- Not easy to estimate: product of 3 expectations!
- Store separately the **lest-square solution** and learn it via gradient descent

$$\mathbf{v} = \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathtt{T}}]^{-1} \mathbb{E}[\delta_{\mathbf{w}} \mathbf{x}]$$

Gradient TD Methods

- **Gradient TD 2 (GTD2)** (Maei et al., 2009)

$$\Delta \mathbf{w} = \alpha \left(\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1})\right) \mathbf{x}(S_t) \mathbf{v}$$
$$\Delta \mathbf{v} = \beta \left(\delta_{\mathbf{w},t} - \mathbf{v}^{\mathsf{T}} \mathbf{x}(S_t)\right) \mathbf{x}(S_t)$$

- **Temporal Difference with Correction (TDC)** (Sutton et al., 2009)

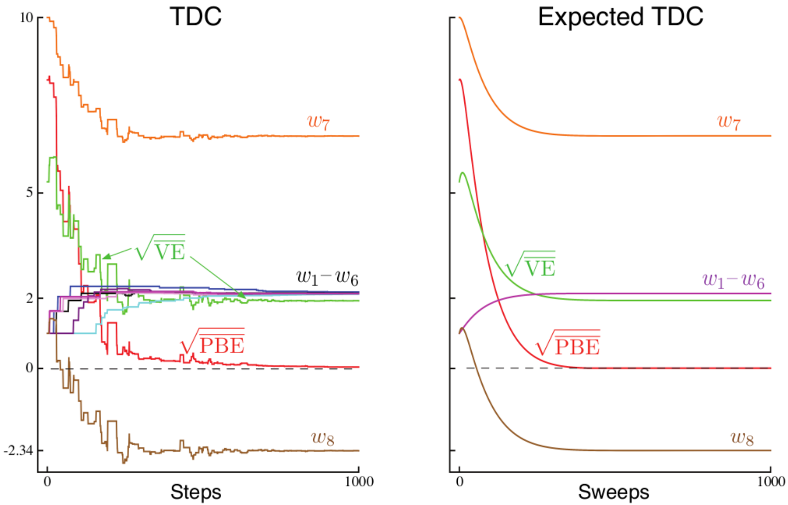
$$\Delta \mathbf{w} = \alpha \left(\delta_{\mathbf{w},t} \mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}) \mathbf{x}(S_t)^{\mathsf{T}} \mathbf{v}\right)$$
$$\Delta \mathbf{v} = \beta \left(\delta_{\mathbf{w},t} - \mathbf{v}^{\mathsf{T}} \mathbf{x}(S_t)\right) \mathbf{x}(S_t)$$

  - $\gamma \mathbf{x}(S_{t+1}) \mathbf{x}(S_t)^{\mathsf{T}} \mathbf{v}$ is the **correction** compared to Semi-Gradient TD(0)
- Convergence with **two-time scales** (β faster than α)

$$\beta \to 0 \qquad\qquad \frac{\alpha}{\beta} \to 0$$

- Extensions to **control** Gradient Q-learning (GQ) (Maei et al., 2010), elegibility traces GQ(λ) (Maei and Sutton, 2010), and hybrid (Sutton et al., 2009)

The Baird's Counterexample with TDC



Pictures from (Sutton and Barto, 2018)

Convergence of Incremental Prediction Algorithms

On/Off–Policy	Algorithm	Table Lookup	Linear	Non–Linear
On–Policy	MC	OK	OK	OK
	TD(0)	OK	OK	KO
	GTD2 (Sutton et al., 2009),	**OK**	**OK**	**OK**
	TDC (Maei, 2011)	**OK**	**OK**	**OK**
Off–Policy	MC	OK	OK	OK
	TD(0)	OK	KO	KO
	GTD2 (Sutton et al., 2009)	**OK**	**OK**	**OK**
	TDC (Maei, 2011)	**OK**	**OK**	**OK**

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-Linear
Monte–Carlo Control	OK	(OK)	KO
SARSA	OK	(OK)	KO
Q–learning	OK	KO	KO
Gradient Q-learning	OK	OK (Maei et al., 2010)	KO (Lee and Anderson, 2014)

(OK) $=$ **chatters** around near–optimal value function

Outline

Batch Reinforcement Learning

- **Incremental** methods are inherently **online**
- Gradient descent is simple but **not** sample efficient
- **Batch** methods are **offline**
- They seek for the **best-fitting** value function given a **batch** of data

Outline

Least Squares Prediction

- Given value function **approximation** $v_{\mathbf{w}}(s) \approx v_\pi(s)$
- And **experience** $\mathcal{D}$ (the batch) consisting of:

$$\mathcal{D} = \{(S_1, \widetilde{G}_1), (S_2, \widetilde{G}_2), \dots, (S_T, \widetilde{G}_T)\}$$

- Which parameters $\mathbf{w}$ give the **best fitting** value function $v_{\mathbf{w}}(s)$?
- Least squares algorithms find parameter vector $\mathbf{w}$ **minimizing mean–squared error** between $v_{\mathbf{w}}(S_t)$ and target values $\widetilde{G}_t$

$$L(\mathbf{w}) = \frac{1}{T} \sum_{t=1}^{T} (\widetilde{G}_t - v_{\mathbf{w}}(S_t))^2$$

Linear Least Squares Prediction Algorithms

- In practice, $\widetilde{G}_t$, our "training data", we must use **noisy** or **biased** samples of $v_\pi(S_t)$
  - LSMC: Least Squares Monte–Carlo uses **return**

$$\widetilde{G}_t = G_t$$

  - LSTD: Least Squares Temporal–Difference uses **TD target**

$$\widetilde{G}_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$$

  - LSTD(λ): Least Squares TD(λ) uses λ**–return**

$$\widetilde{G}_t = G_t^\lambda$$

- One may use the **gradient** but
- we solve directly for **fixed point** of MC/TD/TD(λ) by **vanishing the gradient**
  - Not the **semi-gradient**!

Linear Least Squares Prediction Algorithms

We have **closed-form** solutions:

- LSMC

$$\widehat{\mathbf{w}}^{\mathsf{MC}} = \arg\min_{\mathbf{w}\in\mathbb{R}^n} \widehat{\mathsf{VE}}(\mathbf{w}) = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)\mathbf{x}(S_t)^{\intercal}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

- LSTD

$$\widehat{\mathbf{w}}^{\mathsf{TD}} = \arg\min_{\mathbf{w}\in\mathbb{R}^n} \widehat{\mathsf{TDE}}(\mathbf{w}) = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)(\gamma\mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^{\intercal}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) R_{t+1}$$

- LSTD(λ)

$$\widehat{\mathbf{w}}^{\mathsf{TD}(\lambda)} = \left(\sum_{t=1}^{T} \mathbf{z}_t(\gamma\mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^{\intercal}\right)^{-1} \sum_{t=1}^{T} \mathbf{z}_t R_{t+1}$$

Convergence of Linear Least Squares Prediction Algorithms

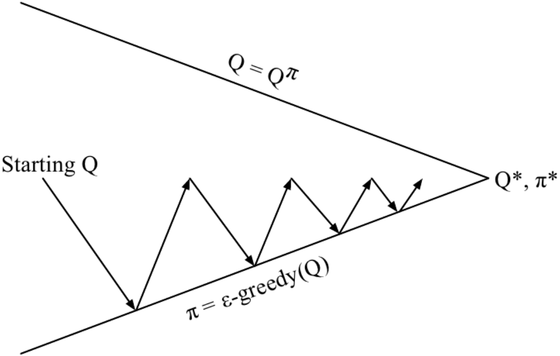
On/Off–Policy	Algorithm	Table Lookup	Linear
On–Policy	LSMC	OK	OK
	LSTD(0)	OK	OK
	LSTD(λ)	OK	OK
Off–Policy	LSMC	OK	OK
	LSTD(0)	OK	OK
	LSTD(λ)	OK	OK

Outline

Approximate Policy and Value Iteration

- **Approximate Policy Iteration (API)** (Scherrer, 2014)
  - **Policy evaluation**: estimate $q_{\mathbf{w}} \approx q_\pi(s, a)$ from data
  - **Policy improvement**: ϵ–greedy policy improvement
  - E.g., Least Squares Policy Iteration (LSPI)
- **Approximate Value Iteration (AVI)** (Munos, 2005)
  - Directly estimate $q_{\mathbf{w}} \approx q_*(s, a)$ from data
  - Return the greedy policy
  - E.g., Fitted Q-Iterations (FQI) (Ernst et al., 2005)

Least Squares Policy Iteration



- **Policy Evaluation**: Least Squares policy evaluation, $q_{\mathbf{w}} \approx q_\pi$
- **Policy Improvement**: ϵ–greedy policy improvement

Pictures from (Sutton and Barto, 2018)

Least Squares Action–Value Function Approximation

- **Approximate** $q_\pi(s,a)$ using linear combination of features

$$q_{\mathbf{w}} = \mathbf{x}(s,a)^{\mathrm{T}}\mathbf{w} \approx q_\pi(s,a)$$

- **Minimize** least squares error between approximate action–value function $q_{\mathbf{w}}(s,a)$ and true action–value function $q_\pi(s,a)$
- LSTDQ algorithm minimizes least squares **TD error** (Lagoudakis and Parr, 2003)

$$\widehat{\mathbf{w}}^{\mathsf{TD}} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t, A_t)(\gamma \mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{x}(S_t, A_t))\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t, A_t) R_{t+1}$$

- If π is deterministic, then we directly set $A_{t+1} = \pi(S_{t+1})$
- Similarly for LSMCQ and LSTD$Q(\lambda)$

Least Squares Policy Iteration Algorithm

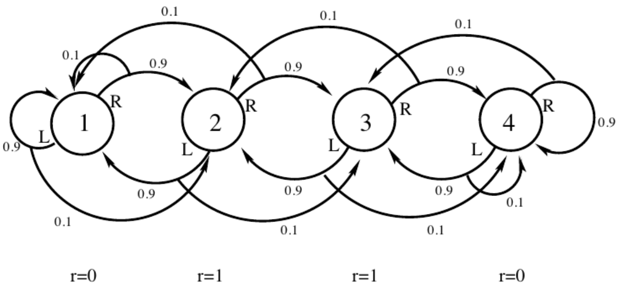
- LSTDQ for policy evaluation

$\pi' \leftarrow \pi_0$
repeat
    $\pi \leftarrow \pi'$
    $\mathbf{w} \leftarrow \text{LSTD}Q(\pi, \mathcal{D})$
    for all $s \in \mathcal{S}$ **do**
        $\pi'(s) \leftarrow \arg\max_{a \in \mathcal{A}} q_{\mathbf{w}}(s, a)$
    end for
until $\pi \approx \pi'$
return π

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear
LSPI–MC	OK	OK
LSPI–TD(0)	OK	OK
LSPI–TD(λ)	OK	OK

Chain Walk Example

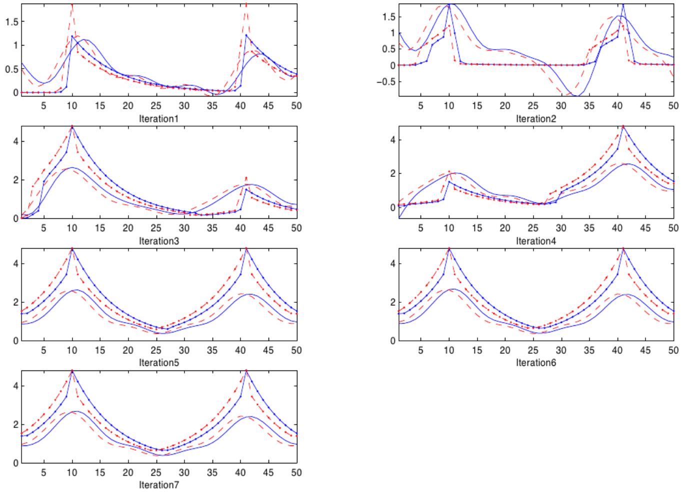


- Consider the **50–state version** of this problems
- **Reward** $+1$ in states 10 and 41, 0 elsewhere
- **Optimal policy**: R (1–9), L (10–25), R (26–41), L (42–50)
- **Features**: 10 evenly spaced RBFs ($\sigma = 4$) for each action
- **Experience**: 10,000 steps from random walk policy

Pictures from (Lagoudakis and Parr, 2003)

LSPI in Chain Walk

Action–Value Function

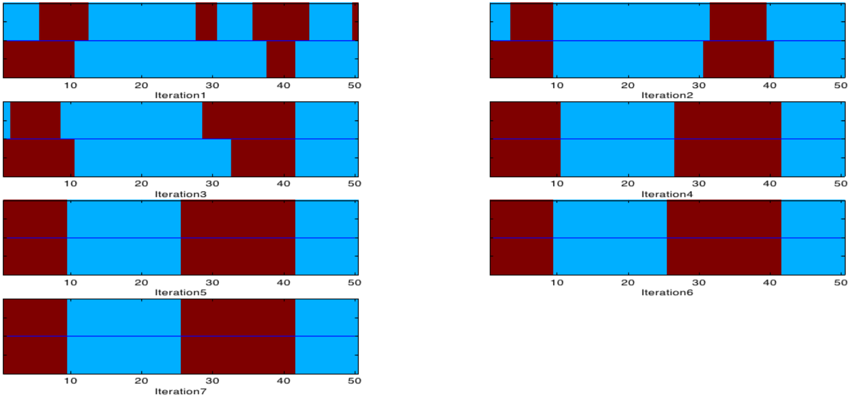


Pictures from (Lagoudakis and Parr, 2003)

- LSPI approximation: solid lines
- Exact values: dotted lines

LSPI in Chain Walk

Policy



- R action: dark/red shade - L action: light/blue shade
- LSPI: top stripe - exact: bottom stripe

Pictures from (Lagoudakis and Parr, 2003)

Fitted Q–Iteration

- Implements **fitted value iteration** (Ernst et al., 2005)
- Given a dataset of experience tuples $\mathcal{D}$, solve a **sequence of regression problems**
  - At iteration i, build an approximation $\hat{q}_i$ over a dataset obtained by $(B_* q_{i-1})(s, a)$
- Allows to use a large class of regression methods (**averagers**), e.g.
  - Kernel averaging
  - Regression trees
  - Fuzzy regression
- With other regression methods it **may diverge**
- In practice, good results also with **neural networks** (Riedmiller, 2005)

Fitted Q–Iteration Algorithm

Input: a set of four–tuples $\mathcal{D} = \{(S_i, A_i, R_{i+1}, S'_i)\}_{i=1}^{L}$ and a regression algorithm
Initialize $i \leftarrow 0$, $\hat{q}_i(s,a) \leftarrow 0, \quad \forall s, a$
repeat
$i \leftarrow i + 1$
Build a training set:

$$\mathcal{TS} = \{(S_i, A_i, R_{i+1} + \gamma \max_{a' \in \mathcal{A}} \hat{q}_{i-1}(S'_i, a'))\}_{i=1}^{L}$$

Use the regression algorithm on $\mathcal{TS}$ to build $\hat{q}_i(s,a)$
until Stopping condition

- Stopping condition:
  - **Fixed** number of iterations
  - When the **distance** between $\hat{q}_i$ and $\hat{q}_{i-1}$ drops below a **threshold**

References I

J. S. Albus. A theory of cerebellar function. *Mathematical biosciences*, 10(1-2):25–61, 1971.

J. S. Albus. Brains, behavior, and robotics. 1981.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556, 2005.

G. J. Gordon. Stable function approximation in dynamic programming. In A. Prieditis and S. J. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 261–268. Morgan Kaufmann, 1995. doi: 10.1016/b978-1-55860-377-6.50040-2.

G. E. Hinton. Distributed representations. 1984.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, 2003.

M. Lee and C. W. Anderson. Convergent reinforcement learning control with neural networks and continuous action search. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–8. IEEE, 2014.

H. R. Maei. Gradient temporal-difference learning algorithms. 2011.

H. R. Maei and R. S. Sutton. Gq (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the third conference on artificial general intelligence*, volume 1, pages 91–96. Citeseer, 2010.

H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 1204–1212, 2009.

H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton. Toward off-policy learning control with function approximation. In *International Conference on Machine Learning (ICML)*, 2010.

A. M. Metelli, M. Papini, F. Faccio, and M. Restelli. Policy optimization via importance sampling. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 5447–5459, 2018.

References II

R. Munos. Error bounds for approximate value iteration. In *The Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 1006–1011. AAAI Press / The MIT Press, 2005.

A. B. Owen. *Monte Carlo theory, methods and examples*. 2013.

T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical report, Massachusetts INST of TECH Cambridge Artificial Intelligence LAB, 1989.

T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247(4945):978–982, 1990.

D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

D. Precup, R. S. Sutton, and S. Dasgupta. Off-policy temporal difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 417–424, 2001.

M. A. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *16th European Conference on Machine Learning (ECML)*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

B. Scherrer. Approximate policy iteration schemes: A comparison. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1314–1322. JMLR.org, 2014.

R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3:9–44, 1988.

R. S. Sutton. On the virtues of linear learning and trajectory distributions. In *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference*, page 85, 1995.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

References III

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, volume 382, pages 993–1000, 2009.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Trans. Autom. Control.*, 42 (5):674–690, 1997.

H. van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.