ST455: Reinforcement Learning

Lecture 5: TD Learning with Function Approximation

Chengchun Shi

Lecture Outline

1. **Introduction to Value Function Approximation**

2. **Gradient Descent-based Methods**

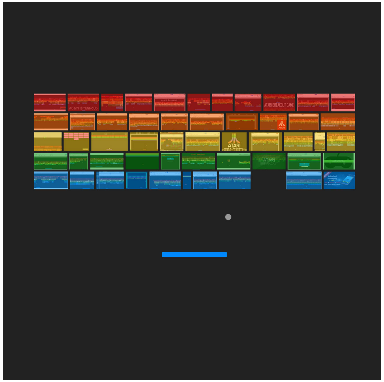
3. **Fitted Q-Iteration**

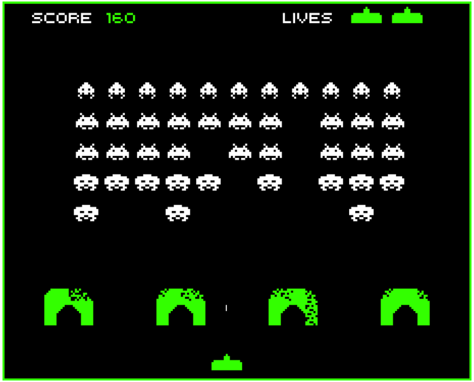
Lecture Outline

Limitations of Tabular Methods

- So far, we studied reinforcement learning methods using a **tabular representation**
  - Focus on **finite** MDPs
  - Value function represented by a **table**
  - Each state s has an entry for value $V(s)$
  - Each state-action pair (a, s) has an entry for value $Q(s, a)$
- **Limitations** of tabular methods
  - Cannot handle **large-scale** RL problems or **continuous** state space
  - **Scalability**: computation time and storage needed to maintain estimates
  - **Slow learning**: learning the value of each state individually

Large Scale RL Problems (Examples)



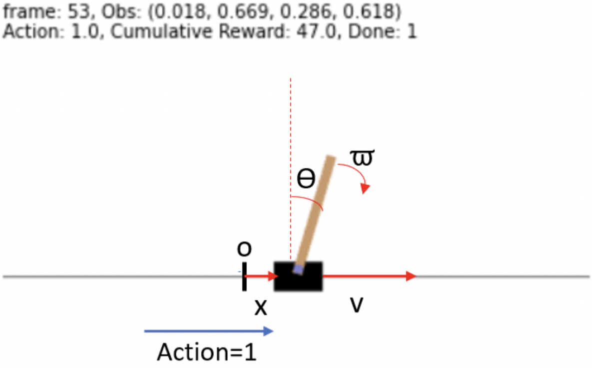


- Image-valued states (e.g., 210× 160 pixel image fames, 129 colours)

Large Scale RL Problems (Examples)

- $19 \times 19 = 361$ Go board, each location (empty, black or white) $\rightarrow 3^{361} \approx 10^{170}$ states

Continuous State Space (Examples)

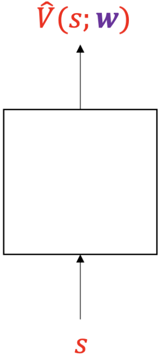


- S_t: x (**Position**); v (**velocity**); θ (**Angle**); ϖ (**Angular velocity**)
- All components are **continuous**

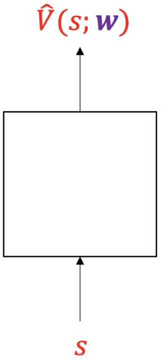
Function Approximation

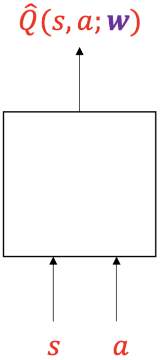
- Estimate a value function using a **parametric** approximator function
- $\widehat{V}(s;\omega)$ as an approximator for the value function $V^{\pi}(s)$
- $\widehat{Q}(s,a;\omega)$ as an approximator for the value function $Q^{\pi}(s,a)$
- Dimension of ω much smaller than the state space size. Represents a **tradeoff**:
  - **Bias** (approximation error) usually decreases with dimension of ω
  - **Variance** (estimation error) usually increase with dimension of ω
- Update parameter ω to find a good approximation using a learning method
  - Eg., MC or TD methods
- Function approximation studied in supervised learning
  - Integrate known methods in reinforcement learning

Types of Value Function Approximator

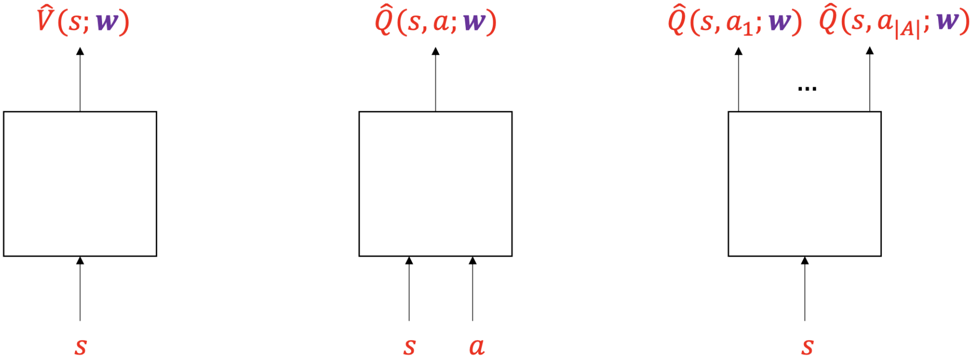


Types of Value Function Approximator (Cont'd)





Types of Value Function Approximator (Cont'd)



Types of Value Function Approximator (Cont'd)

- **Linear combinations of features**
- **Neural networks**
- Decision tree, random forest, boosting
- Nearest neighbor
- ...

Linear Methods

- **Table lookup** is a special case of linear value function approximation in finite MDP

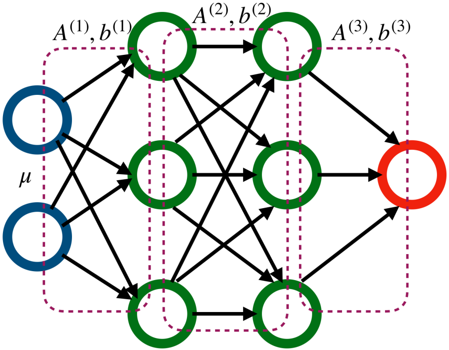
$$\phi(S) = \begin{bmatrix} \mathbb{I}(S = s_1) \\ \mathbb{I}(S = s_2) \\ \vdots \\ \mathbb{I}(S = s_n) \end{bmatrix}$$

- Parameter vector ω gives value of each individual state

$$\widehat{V}(s; \omega) = (\omega_1, \omega_2, \cdots, \omega_n)^\top \phi(S)$$

- **Equivalent** to tabular methods
- Other popular choices for ϕ: RBFSampler, splines, polynomials, etc.

Neural Networks



$$\mu^{(1)} = \sigma(A^{(1)}\mu + b^{(1)}) \in \mathbb{R}^3$$

$$\mu^{(2)} = \sigma(A^{(2)}\mu^{(1)} + b^{(2)}) \in \mathbb{R}^3$$

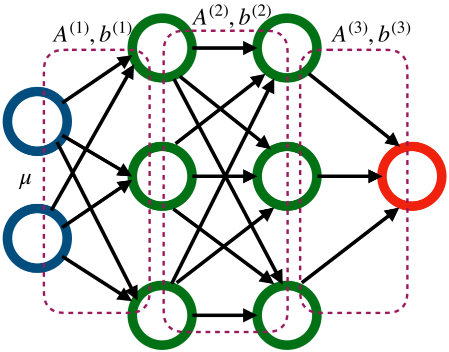
$$y = A^{(3)}\mu^{(2)} + b^{(3)} \in \mathbb{R}$$

Figure: Illustration of fully-connected neural networks with with two hidden layers and three hidden nodes per hidden layer. Here μ is the 2-dimensional input, $A^{(\ell)}$ and $b^{(\ell)}$ denote the corresponding parameters to produce the linear transformation for the $(\ell - 1)$th layer, and σ denotes the element-wise nonlinear transformation function (e.g., sigmoid or ReLU)

Linear v.s. Neural Networks

- Neural networks has **universal approximation property** [Barron, 1993]
- Able to approximate both **smooth** and **nonsmooth** (e.g., step function) functions [Imaizumi and Fukumizu, 2019]
- Difficult to **optimize** (using back propagation)
- Linear methods are **computationally efficient** to implement
- Requires **feature engineering** to have good approximation property

Linear v.s. Neural Networks (Cont'd)



$$\mu^{(1)} = \sigma(A^{(1)}\mu + b^{(1)}) \in \mathbb{R}^3$$

$$\mu^{(2)} = \sigma(A^{(2)}\mu^{(1)} + b^{(2)}) \in \mathbb{R}^3$$

$$y = A^{(3)}\mu^{(2)} + b^{(3)} \in \mathbb{R}$$

- When $A^{(1)}$, $b^{(1)}$, $A^{(2)}$ and $b^{(2)}$ are fixed, can treat $\sigma(A^{(2)}\mu^{(1)} + b^{(2)})$ as features and employ linear method to estimate $A^{(3)}$ and $b^{(3)}$
- Neural networks are **adaptive**: the features involve parameters that are adaptively constructed based on the data

Lecture Outline

1. Introduction to Value Function Approximation

2. **Gradient Descent-based Methods**

3. Fitted Q-Iteration

Function Approximation in RL

- Consider the **policy evaluation** problem: $s \to V^{\pi}(s)$
- RL Examples:
  - MC: $S_t \to G_t$
  - TD($\mathbf{0}$): $S_t \to R_t + \gamma V(S_{t+1})$
  - TD(λ): $S_t \to {G_t}^{\lambda}$
- Like supervised learning: **feature vector** $\to$ **response**
- Unique characteristics of RL: online learning, nonstationary target functions (value function changes with policy)

Parameter Estimation

- **Goal**: find a parameter $\boldsymbol{\omega}$ that minimizes a given **error** function $J : \boldsymbol{\omega} \to \mathbb{R}$
- **Mean squared error** (common supervised learning objective):

$$J(\boldsymbol{\omega}) = \frac{\mathbf{1}}{\mathbf{2}}\mathbb{E}_{\boldsymbol{s}\sim\boldsymbol{\mu}}[\widehat{\boldsymbol{V}}(\boldsymbol{s};\boldsymbol{\omega}) - \boldsymbol{V}^{\boldsymbol{\pi}}(\boldsymbol{s})]^2$$

  where $\boldsymbol{\mu}$ is a distribution on the state space
  (specifies how the error is distributed over different states)

- Common choice for $\boldsymbol{\mu}$: equal to the **on-policy** distribution
  - Distribution of states encountered under policy $\boldsymbol{\pi}$
  - Minimize the error that occur while following the policy

Gradient-Descent Methods

- Assume $\widehat{V}(s;\omega)$ is differentiable with respect to ω for each s
- Consider first a simple case where the training examples are $S_t \to V^{\pi}(S_t)$
  - Input examples give the exact value of the state value
- Gradient-Descent Algorithm: $\omega_{t+1} = \omega_t - \alpha_t \nabla_{\omega} J(\omega_t)$

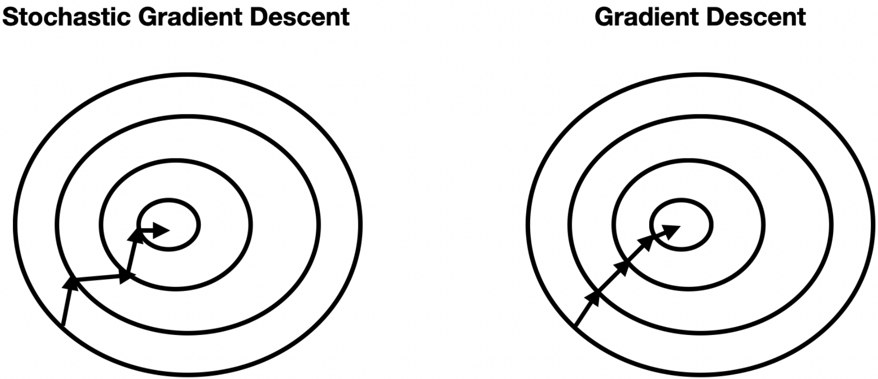
$$\omega_{t+1} = \omega_t - \alpha_t \mathbb{E}_{s\sim\mu}\Big[\Big(V^{\pi}(s) - \widehat{V}(s;\omega_t)\Big)\nabla_{\omega}\widehat{V}(s;\omega_t)\Big]$$

- Stochastic Gradient-Descent Algorithm: $\omega_{t+1} = \omega_t - \alpha_t \nabla_{\omega} J(\omega_t)$

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(V^{\pi}(S_t) - \widehat{V}(S_t;\omega_t)\Big)\nabla_{\omega}\widehat{V}(S_t;\omega_t)\Big]$$

where S_t is distributed according to μ

Gradient-Descent Methods (Cont'd)



- Each point represents a parameter
- Circle represents parameters with the same loss function

Gradient-Descent Methods (Cont'd)

- Assume training examples $S_t \rightarrow \nu_t$ where ν_t is a target, some approx of $V^\pi(S_t)$
- Stochastic Gradient-Descent Algorithm: $\omega_{t+1} = \omega_t - \alpha_t \nabla_\omega J(\omega)$

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(\nu_t - \widehat{V}(S_t; \omega_t) \Big) \nabla_\omega \widehat{V}(S_t; \omega_t) \Big]$$

  where S_t is distributed according to μ
- Some sufficient conditions for convergence to local minimum
  - Standard assumptions on step size: $\sum \alpha_t = \infty$ & $\sum \alpha_t^2 < \infty$ [Robbins and Monro, 1951]
  - ν_t is unbiased to $V^\pi(S_t)$

Monte-Carlo with Value Function Approximation

- Return G_t is an **unbiased**, noisy sample of true value $V^{\pi}(S_t)$
- Can therefore apply **supervised learning** to "training data"

$$\langle S_1, G_1\rangle, \langle S_2, G_2\rangle, \cdots, \langle S_T, G_T\rangle$$

- Applying **gradient-descent methods**

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(G_t - \widehat{V}(S_t; \omega_t) \Big) \nabla_\omega \widehat{V}(S_t; \omega_t) \Big]$$

- Monte Carlo evaluation **converges** to a local minimum

TD Learning with Value Function Approximation

- The TD-target $R_t + \gamma \widehat{V}(S_{t+1}; \omega)$ is a **biased** sample of true value $V^\pi(S_t)$
- Can still apply **supervised learning** to "training data"

$$\langle S_1, R_1 + \gamma \widehat{V}(S_2; \omega) \rangle, \langle S_2, R_2 + \gamma \widehat{V}(S_3; \omega) \rangle, \cdots, \langle S_T, G_T + \gamma \widehat{V}(S_{T+1}; \omega) \rangle$$

- Applying **gradient-descent methods**

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(R_t + \gamma \widehat{V}(S_{t+1}; \omega_t) - \widehat{V}(S_t; \omega_t) \Big) \nabla_\omega \widehat{V}(S_t; \omega_t) \Big]$$

- Linear TD(0) **converges** to a global minimum [Tsitsiklis and Van Roy, 1997]

TD(λ) with Value Function Approximation

- The λ-return ${G_t}^{\lambda}$ is a **biased** sample of true value $V^{\pi}(S_t)$
- Can again apply **supervised learning** to "training data"

$$\langle S_1, {G_1}^{\lambda}\rangle, \langle S_2, {G_2}^{\lambda}\rangle, \cdots, \langle S_T, {G_T}^{\lambda}\rangle$$

- Applying **gradient-descent methods**

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big({G_t}^{\lambda} - \widehat{V}(S_t; \omega_t) \Big) \nabla_{\omega} \widehat{V}(S_t; \omega_t) \Big]$$

- Linear TD(λ) **converges** to a global minimum [Tsitsiklis and Van Roy, 1997]

Linear Function Approximation

- Linear features: $\phi(s)$ (e.g., polynomials, trigonometric polynomials, B-splines)
- MC update rule:

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(G_t - \phi^\top(S_t)\omega_t \Big) \phi(S_t) \Big]$$

- TD(0) update rule:

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(R_t + \gamma \phi^\top(S_{t+1})\omega_t - \phi^\top(S_t)\omega_t \Big) \phi(S_t) \Big]$$

- TD(λ) update rule:

$$\omega_{t+1} = \omega_t - \alpha_t \Big[\Big(G_t{}^\lambda - \phi^\top(S_t)\omega_t \Big) \phi(S_t) \Big]$$

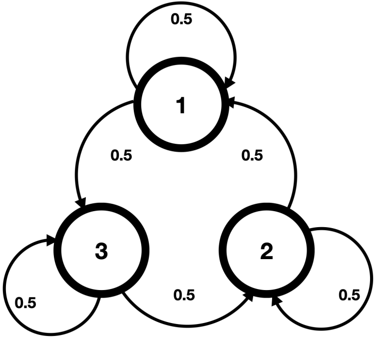
Nonlinear Function Approximation

- Unlike linear methods, gradient-based TD learning algorithm with nonlinear approximation may diverge

- Proof by an example [Tsitsiklis and Van Roy, 1997]

Bad Example

- Markov chain with state space $\{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$ and transition matrix



- All instantaneous rewards equal to zero
- The value function is zero for any state and policy

Bad Example (Cont'd)

- Nonlinear approximator $\widehat{\boldsymbol{V}}(\omega) = (\widehat{\boldsymbol{V}}(\mathbf{1};\omega), \widehat{\boldsymbol{V}}(\mathbf{2};\omega), \widehat{\boldsymbol{V}}(\mathbf{3};\omega))^\top$ for some scalar ω
- Arbitrary initial value $\sum_{\boldsymbol{s}} \widehat{\boldsymbol{V}}(\boldsymbol{s};\omega) = \mathbf{0}$
- Use an ordinary differential equation (ODE) model for parametrization

$$\frac{d\widehat{\boldsymbol{V}}(\omega)}{d\omega} = (\boldsymbol{Q} + \varepsilon \boldsymbol{I})\widehat{\boldsymbol{V}}(\omega)$$

for some small constant $\varepsilon > \mathbf{0}$ and

$$\boldsymbol{Q} = \begin{pmatrix} 1 & 1/2 & 3/2 \\ 3/2 & 1 & 1/2 \\ 1/2 & 3/2 & 1 \end{pmatrix}$$

- If the RHS of ODE does not involve $\widehat{\boldsymbol{V}}$, reduces to the linear model

Bad Example (Cont'd)

- Recall that the value function equals zero
- The mean squared error objective function

$$J(\omega) = \sum_{s=1}^{3} \left[\widehat{V}(s;\omega) - V^{\pi}(s) \right]^2 = \sum_{s=1}^{3} \widehat{V}^2(s;\omega)$$

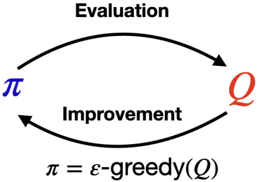
- It can be shown that under TD(0) update [Tsitsiklis and Van Roy, 1997]
  - ω_t increases with t
  - $\sum_{s=1}^{3} \widehat{V}^2(s;\omega)$ increases with ω
  - $J(\omega_t)$ diverges with t

Convergence of Prediction Algorithms

Algorithm	Tabular	Linear	Non-linear
MC	✔	✔	✔
TD($\mathbf{0}$)	✔	✔	✗
TD($\boldsymbol{\lambda}$)	✔	✔	✗

Source: Silver, UCL RL course,
https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf

Control with Gradient Descent-based Methods



$$\pi_0 \longrightarrow Q^{\pi_0} \longrightarrow \pi_1 \longrightarrow Q^{\pi_1} \longrightarrow \cdots \longrightarrow \pi^{opt} \longrightarrow Q^{\pi^{opt}}$$

- **Policy evaluation**: approximate action-state value function $\boldsymbol{Q}^{\pi} = \widehat{\boldsymbol{Q}}(\bullet, \bullet; \omega)$
- **Policy improvement**: ε-greedy policy improvement

Action-State Value Function Approximation

- Approximate action-state value function

$$Q^{\pi}(s, a) = \widehat{Q}(s, a; \omega)$$

- Minimise mean-squared error

$$J(\omega) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\mu}[\widehat{Q}(s, a; \omega) - Q^{\pi}(s, a)]^2$$

- Use stochastic gradient descent

$$\omega_{t+1} = \omega_t - \alpha_t \left[\left(q_t - \widehat{Q}(S_t, A_t; \omega_t)\right) \nabla_{\omega} \widehat{Q}(S_t, A_t; \omega_t)\right]$$

for some target q_t, some approx of $Q^{\pi}(S_t, A_t)$

Value Function Approximation (Cont'd)

- For MC, the target is the return $\boldsymbol{G_t}$

$$\boldsymbol{\omega_{t+1}} = \boldsymbol{\omega_t} - \boldsymbol{\alpha_t}\left[\left(\boldsymbol{G_t} - \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right) \nabla_\omega \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right]$$

- For TD(0) (SARSA), the target is $\boldsymbol{R_t} + \gamma \widehat{\boldsymbol{Q}}(\boldsymbol{S_{t+1}}, \boldsymbol{A_{t+1}}; \boldsymbol{\omega})$

$$\boldsymbol{\omega_{t+1}} = \boldsymbol{\omega_t} - \boldsymbol{\alpha_t}\left[\left(\boldsymbol{R_t} + \gamma \widehat{\boldsymbol{Q}}(\boldsymbol{S_{t+1}}, \boldsymbol{A_{t+1}}; \boldsymbol{\omega_t}) - \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right) \nabla_\omega \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right]$$

- For TD($\boldsymbol{\lambda}$) (SARSA($\boldsymbol{\lambda}$)), the target is $\boldsymbol{Q_t^\lambda}$

$$\boldsymbol{\omega_{t+1}} = \boldsymbol{\omega_t} - \boldsymbol{\alpha_t}\left[\left(\boldsymbol{Q_t^\lambda} - \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right) \nabla_\omega \widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega_t})\right]$$

Linear Function Approximation (Cont'd)

- For MC, the target is the return $\boldsymbol{G_t}$

$$\omega_{t+1} = \omega_t - \alpha_t \left[\left(\boldsymbol{G_t} - \phi^\top(\boldsymbol{S_t}, \boldsymbol{A_t})\omega_t \right) \phi(\boldsymbol{S_t}, \boldsymbol{A_t}) \right]$$

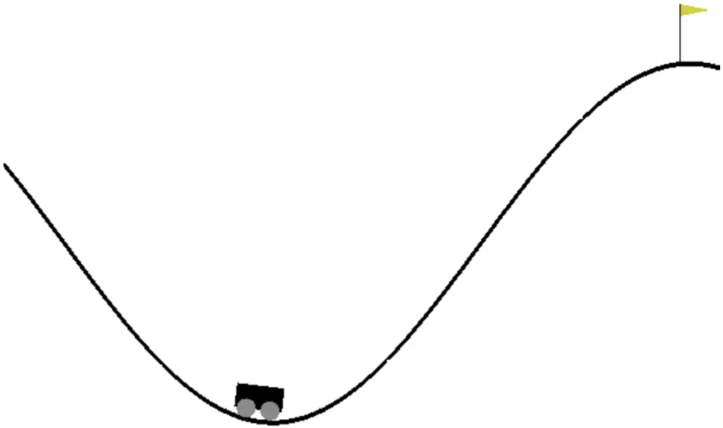
- For TD(0) (SARSA), the target is $\boldsymbol{R_t} + \gamma\phi^\top(\boldsymbol{S_{t+1}}, \boldsymbol{A_{t+1}})\omega$

$$\omega_{t+1} = \omega_t - \alpha_t \left[\left(\boldsymbol{R_t} + \gamma\phi^\top(\boldsymbol{S_{t+1}}, \boldsymbol{A_{t+1}})\omega_t - \phi^\top(\boldsymbol{S_t}, \boldsymbol{A_t})\omega_t \right) \phi(\boldsymbol{S_t}, \boldsymbol{A_t}) \right]$$

- For TD($\boldsymbol{\lambda}$) (SARSA($\boldsymbol{\lambda}$)), the target is $\boldsymbol{Q_t^\lambda}$

$$\omega_{t+1} = \omega_t - \alpha_t \left[\left(\boldsymbol{Q_t^\lambda} - \phi^\top(\boldsymbol{S_t}, \boldsymbol{A_t})\omega_t \right) \phi(\boldsymbol{S_t}, \boldsymbol{A_t}) \right]$$

The Mountain Car Example



Convergence of Control Algorithms

Algorithm	Tabular	Linear	Non-linear
MC	✔	✔	✔
SARSA	✔	✔	✗
Q-Learning	✔	✗	✗

Source: Silver, UCL RL course,
https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf

Lecture Outline

1. Introduction to Value Function Approximation
2. Gradient Descent-based Methods
3. **Fitted Q-Iteration**

Limitations of Gradient-based Control Methods

- MC control allows both linear and nonlinear approximation, but is **inefficient**
  - G_t suffers from **large variance**, so is the estimated Q-function
- SARSA is efficient, but cannot allow **nonlinear** approximation
  - When using linear approximation, the estimator can suffer from **large bias**

Efficient Control with Function Approximation

- **Batch** (offline) setting with pre-collected data $\{S_t, A_t, R_t, S_{t+1}\}_t$
- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_{a} Q^{\pi^{\text{opt}}}(S_{t+1}, a) \middle| S_t, A_t\right]$$

- Supervised learning is sample efficient in **batch** settings
- Use supervised learning to learn $Q^{\pi^{\text{opt}}}$ by solving Bellman optimality equation

Challenge

- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \middle| S_t, A_t \right]$$

- Both LHS and RHS involve $Q^{\pi^{\text{opt}}}$
- A naive approach: minimize the **mean squared Bellman error**

$$\sum_t \left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

This would yield a **biased** estimator!

The Bellman Error is Not Learnable

- For a given random variable Z, $\mathbb{E}Z^2 = (\mathbb{E}Z)^2 + \mathrm{Var}(Z)$
- The mean squared Bellman error can be decomposed into **squared bias** + **variance**

$$
\begin{aligned}
& \mathbb{E}\left[\left\{R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right\}^2 \middle| A_t, S_t\right] \\
= \quad & \left[\mathbb{E}\left\{R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t\right\}\right]^2 \\
+ \quad & \mathrm{Var}\left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t\right]
\end{aligned}
$$

- The second line is **zero** when $Q = Q^{\pi^{\text{opt}}}$
- The third line is **nonzero** for any Q and is a function of Q as well
- There is no guarantee $Q^{\pi^{\text{opt}}}$ is the minimizer

Fitted Q-Iteration [Riedmiller, 2005]

- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \,\middle|\, S_t, A_t\right]$$

  Both LHS and RHS involve $Q^{\pi^{\text{opt}}}$
- **Main idea**: Fix $Q^{\pi^{\text{opt}}}$ on the RHS
- **Repeat** the following
  1. Compute $\widehat{Q}$ as the argmin of

$$\arg\min_Q \sum_t \left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t)\right]^2$$

  2. Set $\widetilde{Q} = \widehat{Q}$

Fitted Q-Iteration (Cont'd)

- During each iteration, consider the objective function

$$
\begin{aligned}
& \mathbb{E}\left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right]^2 \\
= \; & \left[\mathbb{E}\left\{ R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right\} \right]^2 \\
+ \; & \mathrm{Var}\left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right]
\end{aligned}
$$

- When $\widetilde{Q}$ is close to $Q^{\pi^{\text{opt}}}$, the second line is **small** when $Q = Q^{\pi^{\text{opt}}}$
- The third line is the same for any Q, since $\widetilde{Q}$ is fixed

Fitted Q-Iteration: Algorithm

- **Initialization**: $\widehat{\boldsymbol{Q}}, \widetilde{Q}$ arbitrary, $\boldsymbol{k} = \mathbf{0}$
- While $(\boldsymbol{k} < \boldsymbol{K})$ **Repeat**

  Generated data $\{(\boldsymbol{S_t}, \boldsymbol{A_t}, \boldsymbol{R_t}, \boldsymbol{S_{t+1}})\}$ using policy derived from $\widehat{\boldsymbol{Q}}$ (e.g., ε-greedy)

  Compute $\widehat{\boldsymbol{Q}}$ as the argmin of

$$\arg\min_{\boldsymbol{Q}} \sum_{\boldsymbol{t}} \left[\boldsymbol{R_t} + \gamma \max_{\boldsymbol{a}} \widetilde{\boldsymbol{Q}}(\boldsymbol{S_{t+1}}, \boldsymbol{a}) - \boldsymbol{Q}(\boldsymbol{S_t}, \boldsymbol{A_t})\right]^2$$

  Set $\widetilde{\boldsymbol{Q}} = \widehat{\boldsymbol{Q}}$

Advantages of Fitted Q-Iteration

- **Flexibility**: any supervised learning method (e.g., deep learning, boosting, random forest) is applicable to learn the Q-function during each iteration.
  - Gradient Descent-based methods require the Q-function model to be a smooth function of the model parameters

- **Efficiency**: borrows the strength of supervised learning for sample-efficient estimation. Allows high-dimensional state information.

Theoretical Analysis of Fitted Q-Iteration

- Let $\widehat{\boldsymbol{Q}}_{\boldsymbol{k}}$ denote the Q-estimator during the $\boldsymbol{k}$th iteration
- Error decomposition: **bias due to initialization** $+$ **stochastic estimation error**
- The initialization bias $\rightarrow \mathbf{0}$ as $\boldsymbol{k} \rightarrow \infty$
- The estimation error $\rightarrow \mathbf{0}$ when supervised learning provides a **consistent** estimator at each iteration

Theoretical Analysis of Fitted Q-Iteration (Cont'd)

- At the $\boldsymbol{k}$th iteration,

$$\widehat{Q}_k = \arg\min_{Q} \sum_t \left[R_t + \gamma \max_a \widehat{Q}_{k-1}(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

- Supervised learning target:

$$Q_k(s, a) = \mathbb{E}\left[R_t + \gamma \max_a \widehat{Q}_{k-1}(S_{t+1}, a) \middle| S_t = s, A_t = a \right]$$

Theoretical Analysis of Fitted Q-Iteration (Cont'd)

- A key inequality

$$\sup_{s,a} |\widehat{Q}_k(s,a) - Q^{\pi^{\text{opt}}}(s,a)| \le \sup_{s,a} |\widehat{Q}_k(s,a) - Q_k(s,a)| + \gamma \sup_{s,a} |\widehat{Q}_{k-1}(s,a) - Q^{\pi^{\text{opt}}}(s,a)|$$

- Iteratively applying the inequality

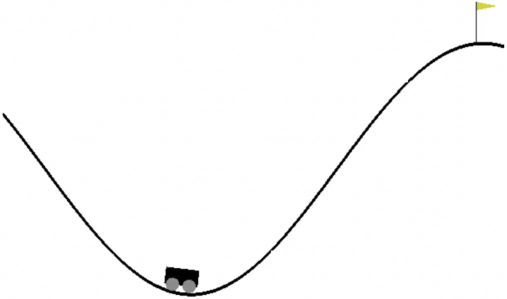
$$\sup_{s,a} |\widehat{Q}_k(s,a) - Q^{\pi^{\text{opt}}}(s,a)| \le \gamma^k \sup_{s,a} \underbrace{|\widehat{Q}_0(s,a) - Q^{\pi^{\text{opt}}}(s,a)|}_{\text{Initialization Bias}} + \sup_{s,a} \max_{j=\{1,\cdots,k\}} \underbrace{|\widehat{Q}_j(s,a) - Q_j(s,a)|}_{\text{Estimation Error}}$$

Summary

- Linear function approximation
- Gradient-based methods
- Gradient-based MC, TD, SARSA
- Neural networks
- Stochastic gradient-based methods
- Fitted Q-iteration

Seminar Exercises

- Solution to HW4 (Deadline, Wed 12:00 PM)
- The mountain car example: gradient-based methods and fitted Q-iteration



References I

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Masaaki Imaizumi and Kenji Fukumizu. Deep neural networks learn non-smooth functions effectively. In *The 22nd international conference on artificial intelligence and statistics*, pages 869–878. PMLR, 2019.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

Questions