Value Function Approximation

Junni Zou

Institute of Media, Information and Network
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
http://min.sjtu.edu.cn

Spring, 2024

Outline

Table of Contents

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve large problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up the model-free methods for prediction and control from the last two lectures?

Value Function Approximation

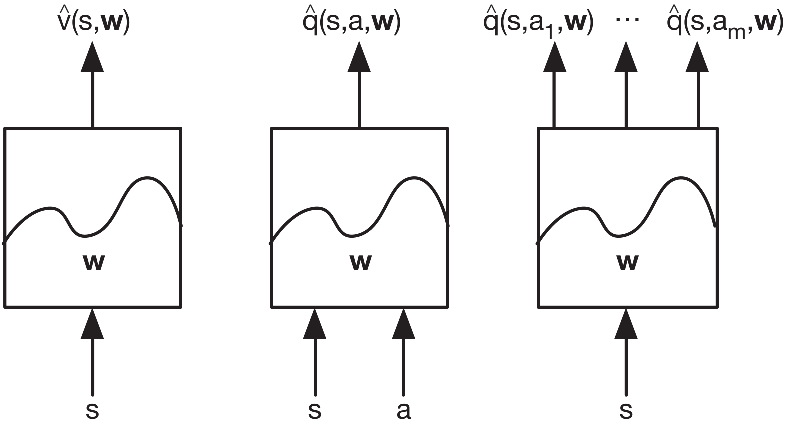
- So far we have represented value function by a *lookup* table
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

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

  - *Generalize* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Institute of Media, Information, and Network

Types of Value Function Approximation



Institute of Media, Information, and Network

Which Function Approximator?

There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbor
- Fourier/wavelet bases
- $\cdots$

Which Function Approximator?

We consider differentiable function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbor
- Fourier / wavelet bases
- $\cdots$

Furthermore, we require a training method that is suitable for non-stationary, non-iid data

Table of Contents

Gradient Descent

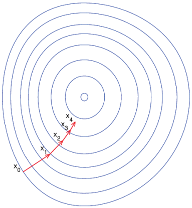
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

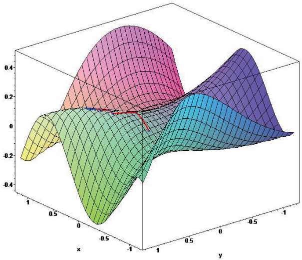
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

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

where α is a step-size parameter





Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimizing mean-squared error between approximate value function $\hat{v}(s, \mathbf{w})$ and true value function $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})] \end{aligned}$$

- Stochastic gradient descent *sample* the gradient

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^T \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$$

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

$$J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(S) - \mathbf{x}(S)^T \mathbf{w})^2]$$

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

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$
$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

Update = *step-size* × *prediction error* × *feature value*

Institute of Media, Information, and Network

Table Lookup Features

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

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

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

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

Incremental Prediction Algorithms

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_\mathbf{w} \hat{v}(S, \mathbf{w})$$

- Have assumed true value function $v_\pi(s)$ given by supervisor
- 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

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_\mathbf{w} \hat{v}(S, \mathbf{w})$$

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

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

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

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_\mathbf{w} \hat{v}(S, \mathbf{w})$$

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":

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

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

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

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

Institute of Media, Information, and Network

Monte-Carlo Prediction with Value Function Approximation

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
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 $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Institute of Media, Information, and Network

TD Learning with Value Function Approximation

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, \cdots, \langle S_{T-1}, R_T \rangle$$

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

$$\begin{aligned} \Delta \mathbf{w} &= \alpha ({\color{red} R + \gamma \hat{v}(S', \mathbf{w})} - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S) \end{aligned}$$

- Linear TD(0) converges (close) to global optimum

TD(0) Prediction with Value Function Approximation

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

Institute of Media, Information, and Network

TD(λ) with Value Function Approximation

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

$$\langle S_1, G_1^\lambda \rangle, \langle S_2, G_2^\lambda \rangle, \cdots, \langle S_{T-1}, G_{T-1}^\lambda \rangle$$

- Forward view linear TD(λ)

$$\begin{aligned} \Delta \mathbf{w} &= \alpha({\color{red}G_t^\lambda} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma\lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t \end{aligned}$$

TD(λ) with Value Function Approximation

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

$$\langle S_1, G_1^\lambda \rangle, \langle S_2, G_2^\lambda \rangle, \cdots, \langle S_{T-1}, G_{T-1}^\lambda \rangle$$

- Forward view linear TD(λ)

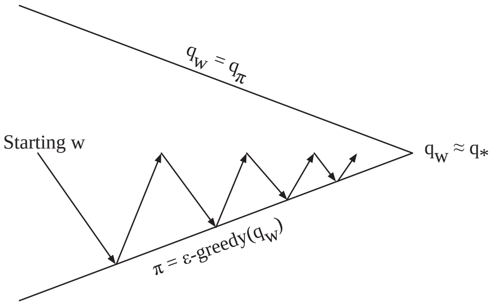
$$\begin{aligned} \Delta \mathbf{w} &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma\lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t \end{aligned}$$

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

Control with Value Function Approximation



Policy evaluation Approximation policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
Policy improvement ϵ-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value function $\hat{q}(S, A, \mathbf{w})$ and true action-value function $q_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$
$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^T \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A) \mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$
$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Incremental Control Algorithms

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

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

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

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

  - For forward-view TD(λ), the target is the λ-return q_t^λ

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

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

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\
E_t &= \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w}) \\
\Delta \mathbf{w} &= \alpha \delta_t E_t
\end{aligned}$$

Institute of Media, Information, and Network

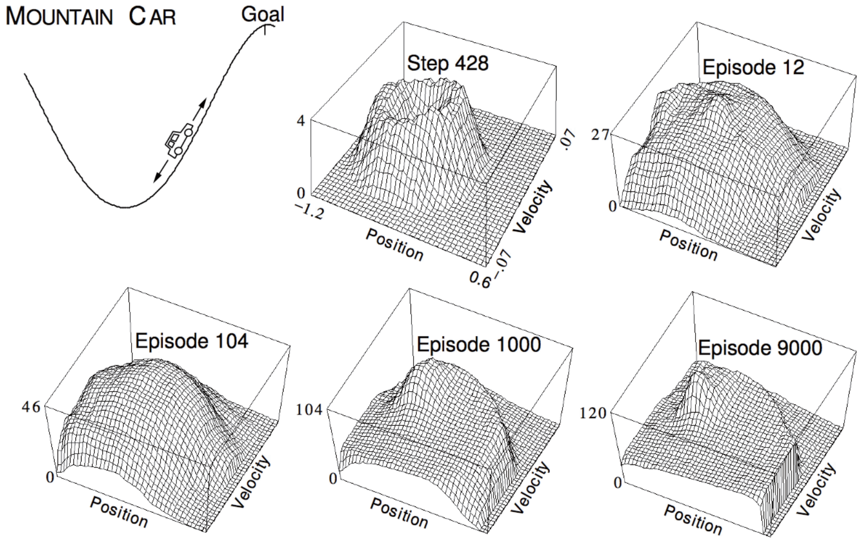
On-Policy Control with Value Function Approximation: Sarsa

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

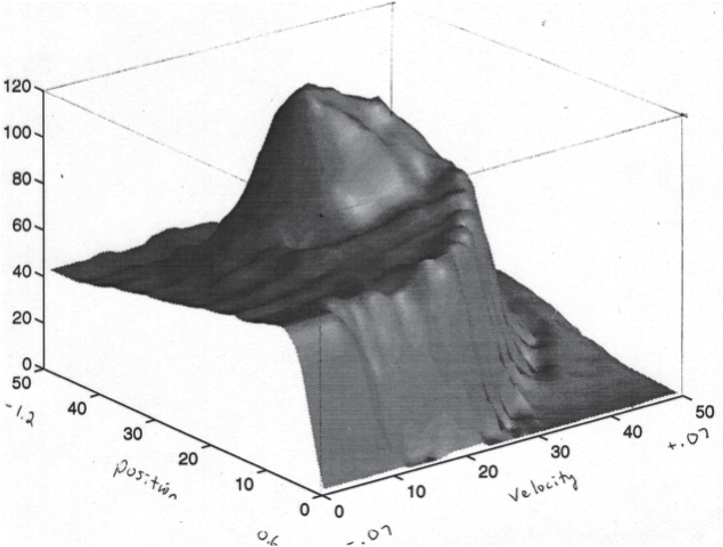
Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R - \hat{q}(S, A, \mathbf{w})\big] \nabla \hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Linear Sarsa with Coarse Coding in Mountain Car

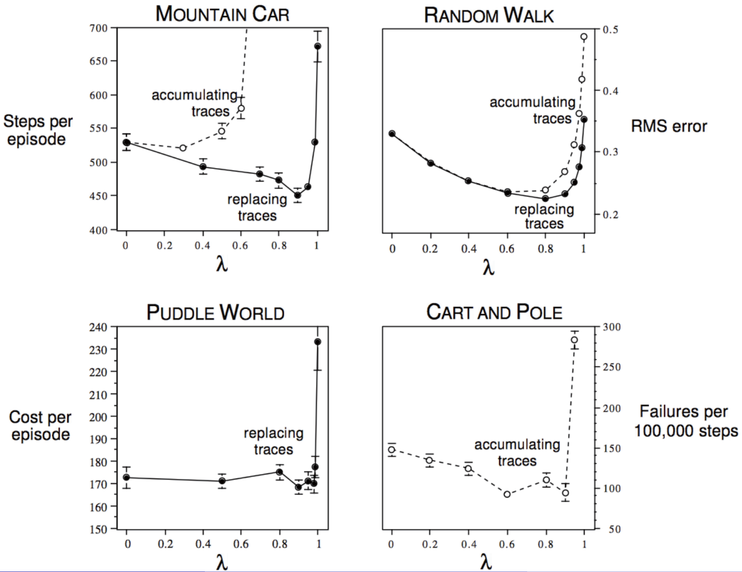


:twork

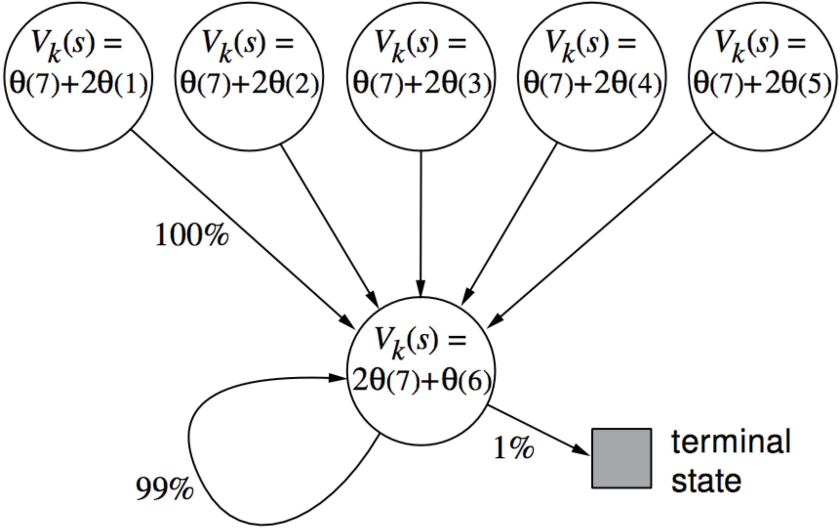
Linear Sarsa with Radial Basis Functions in Mountain Car



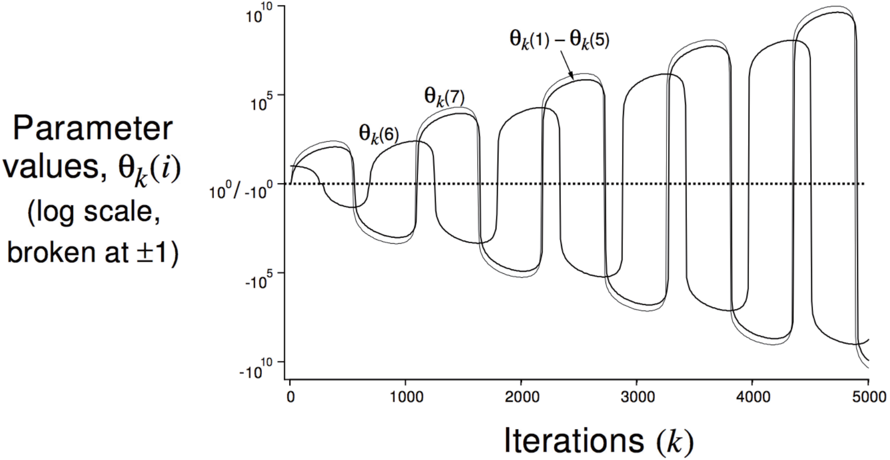
Study of λ: Should We Bootstrap?



Baird's Counterexample



Parameter Divergence in Baird's Counterexample



Convergence of Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Institute of Media, Information, and Network

Gradient Temporal-Difference Learning

- TD does not follow the gradient of any objective function
- This is why TD can diverge when off-policy or using non-linear function approximation
- Gradient TD follows true gradient of projected Bellman error

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

Institute of Media, Information, and Network

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

(✓) = chatters around near-optimal value function

Table of Contents

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is not sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience ("training data")

Least Squares Prediction

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle$ state, value $\rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \cdots, \langle s_T, v_T^\pi \rangle\}$$

- Which parameters $\mathbf{w}$ give the *best fitting* value function $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms find parameter vector $\mathbf{w}$ minimizing sum-squared error between $\hat{v}(s, \mathbf{w})$ and target values v_t^π

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_{\mathcal{D}}\left[(v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2\right] \end{aligned}$$

Institute of Media, Information, and Network

Deep Q-Networks (DQN): Q-learning Framework

DQN based on Q-learning framework, uses two tips: fixed target network and experience replay

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Institute of Media, Information, and Network

Deep Q-Networks (DQN): Fixed Target Network

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$\boxed{\begin{aligned} &\mathrm{Q}^{\pi}(s_t, a_t) \\ &\quad = r_t + \mathrm{Q}^{\pi}\big(s_{t+1}, \pi(s_{t+1})\big) \end{aligned}}$$

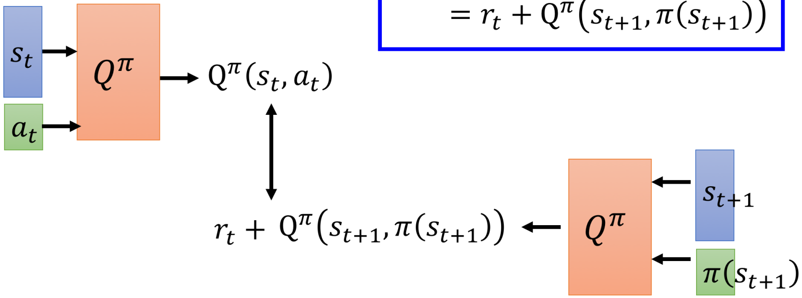
twork

Deep Q-Networks (DQN): Fixed Target Network

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^\pi(s_t, a_t) = r_t + Q^\pi\big(s_{t+1}, \pi(s_{t+1})\big)$$



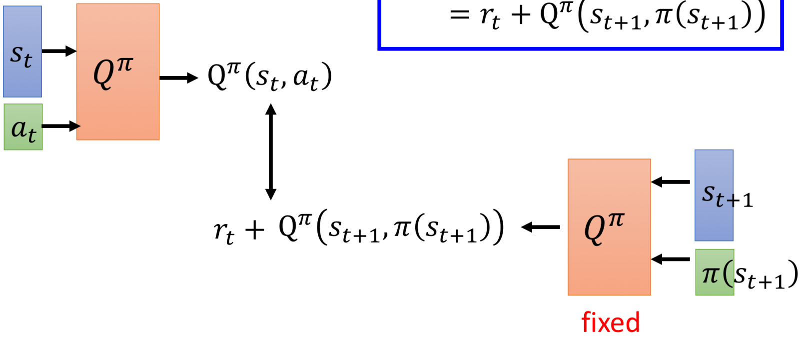
twork

Deep Q-Networks (DQN): Fixed Target Network

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^\pi(s_t, a_t) = r_t + Q^\pi\big(s_{t+1}, \pi(s_{t+1})\big)$$



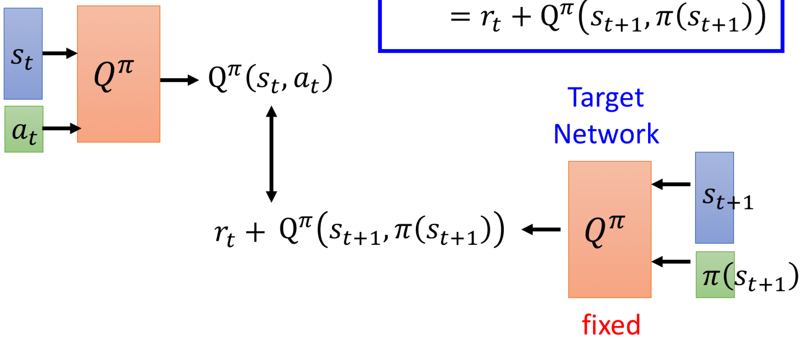
twork

Deep Q-Networks (DQN): Fixed Target Network

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^\pi(s_t, a_t) = r_t + Q^\pi(s_{t+1}, \pi(s_{t+1}))$$



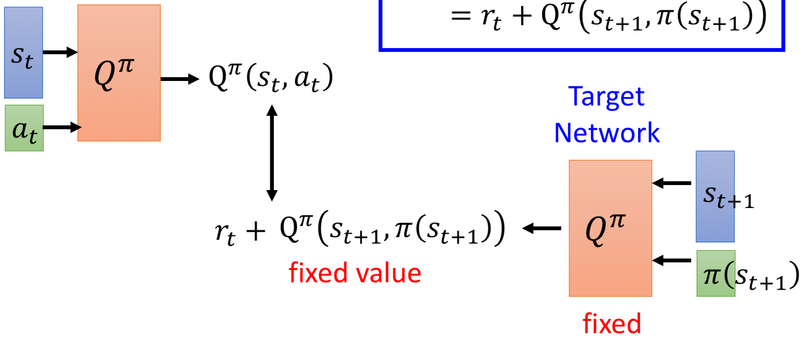
twork

Deep Q-Networks (DQN): Fixed Target Network

Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$\begin{aligned} Q^{\pi}(s_t, a_t) \\ &= r_t + Q^{\pi}\big(s_{t+1}, \pi(s_{t+1})\big) \end{aligned}$$



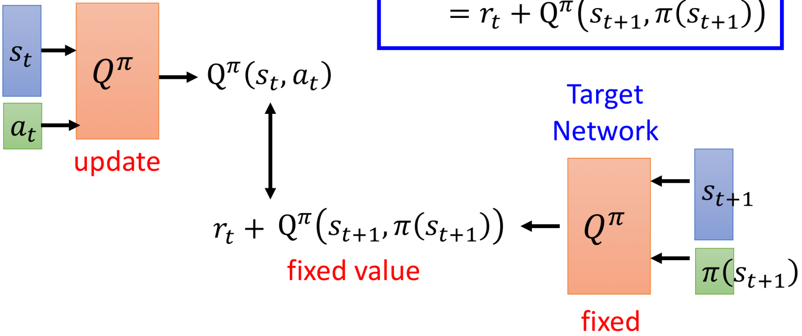
twork

Deep Q-Networks (DQN): Fixed Target Network

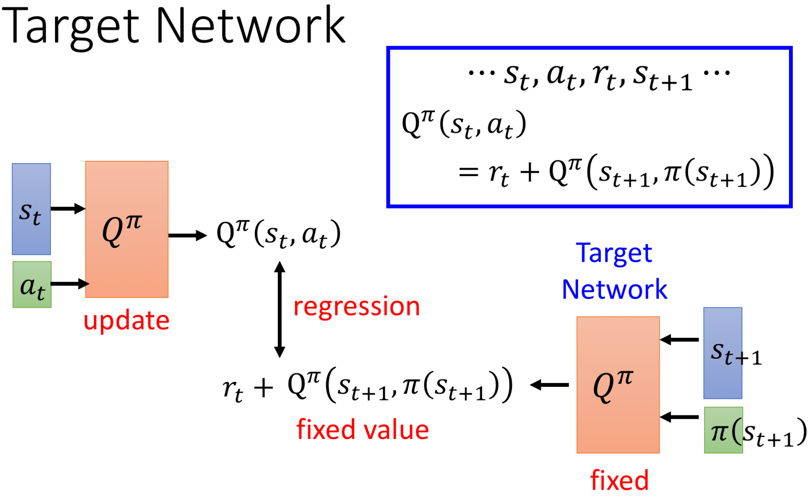
Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

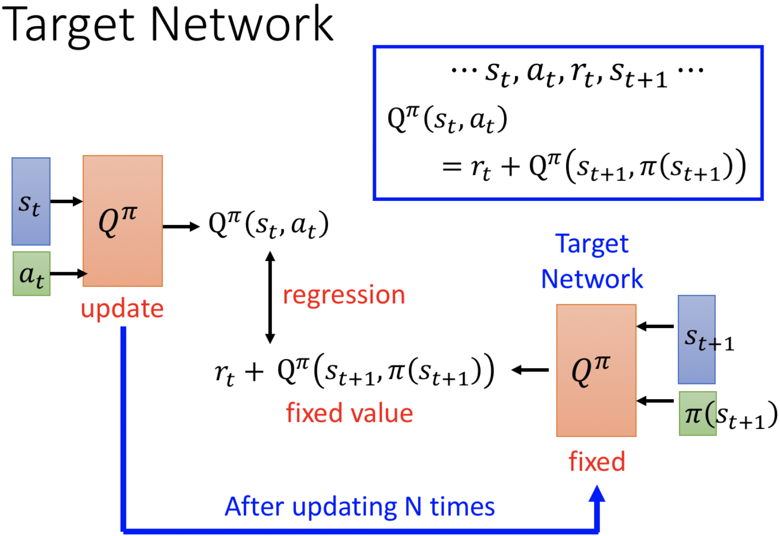
$$Q^{\pi}(s_t, a_t) \\ = r_t + Q^{\pi}\big(s_{t+1}, \pi(s_{t+1})\big)$$



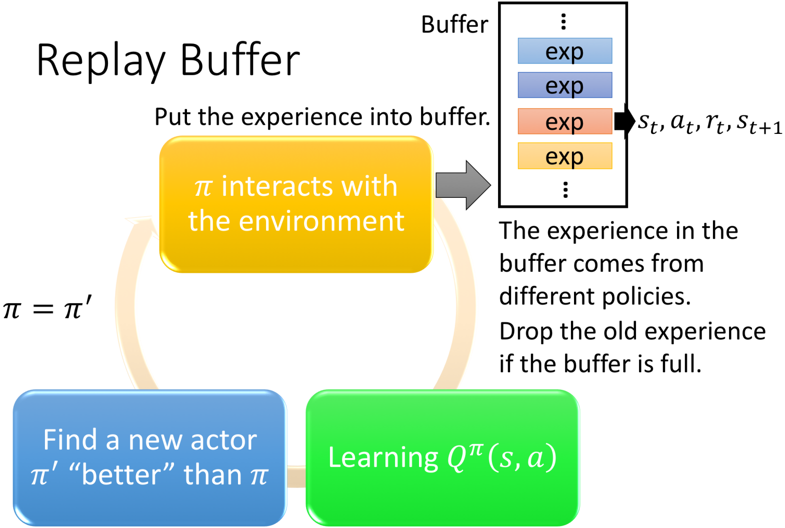
Deep Q-Networks (DQN): Fixed Target Network



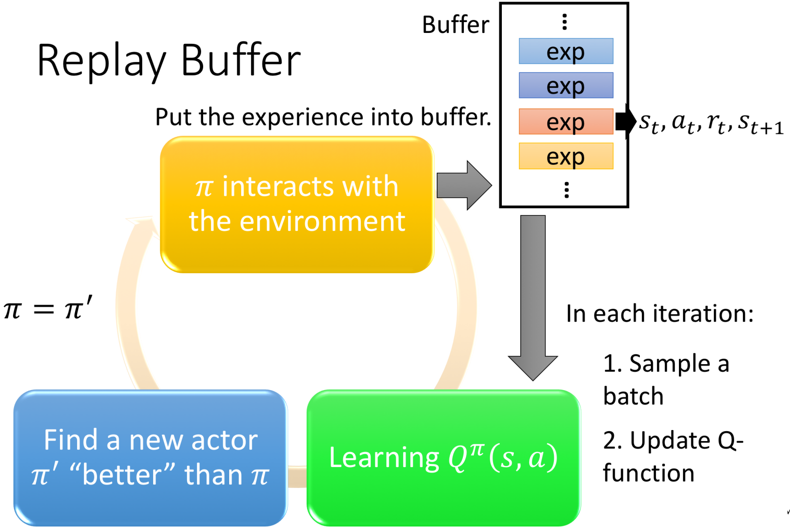
Deep Q-Networks (DQN): Fixed Target Network



Deep Q-Networks (DQN): Experience Replay



Deep Q-Networks (DQN): Experience Replay



DQN Algorithm

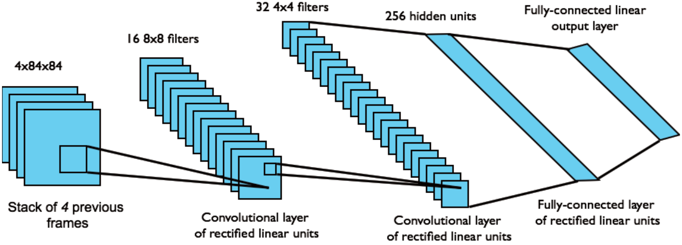
Typical Q-Learning Algorithm

- Initialize Q-function Q, target Q-function $\hat{Q} = Q$
- In each episode
  - For each time step t
    - Given state s_t, take action a_t based on Q (epsilon greedy)
    - Obtain reward r_t, and reach new state s_{t+1}
    - Store (s_t, a_t, r_t, s_{t+1}) into buffer
    - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
    - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
    - Every C steps reset $\hat{Q} = Q$

f Media,
on, and Network

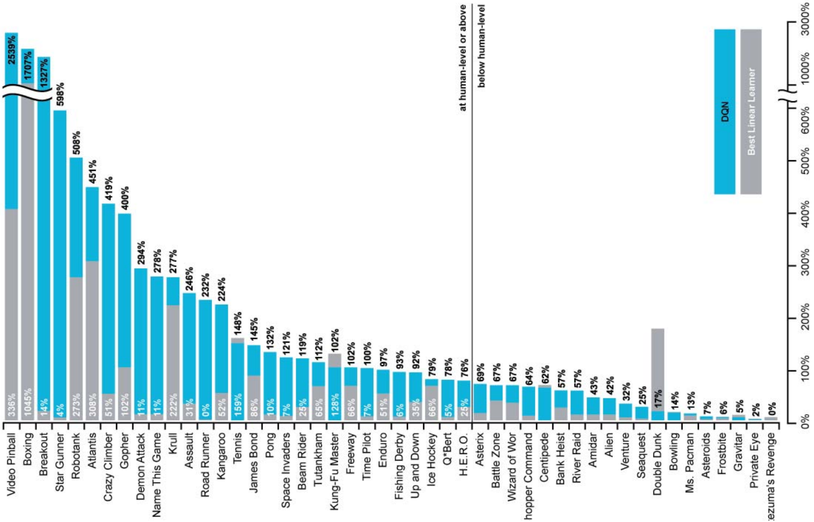
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick / button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

DQN Results in Atari



a,
Network

How much does DQN help?

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99