Value Function Approximation

Outline

- ✓ Introduction
- ✓ Incremental methods
  - ✓ Stochastic Gradient Descent
  - ✓ Gradient TD learning
  - ✓ Linear value approximation
- ✓ Batch methods
  - ✓ Linear least squares
  - ✓ Experience replay & DQN

Introduction

Reinforcement Learning on the Scale

We would like to be able to use reinforcement learning in problems with non-trivial state spaces

- ✓ Backgammon: 1020 states
- ✓ Go: 10170 states
- ✓ Robot control: continuous state space
- ✓ Molecule search: $>10^{60}$ states

Scale up model-free methods for *prediction* and *control* from the last two lectures?

Value Function Approximation

So far $V(s)/Q(s,a)$ = lookup table

- ✓An entry for every state s or state-action pair s, a
- ✓Large MDPs $\Longrightarrow$ too many states and/or actions to store in memory
- ✓Too slow to learn the value of each state individually
- ✓Generalization issues

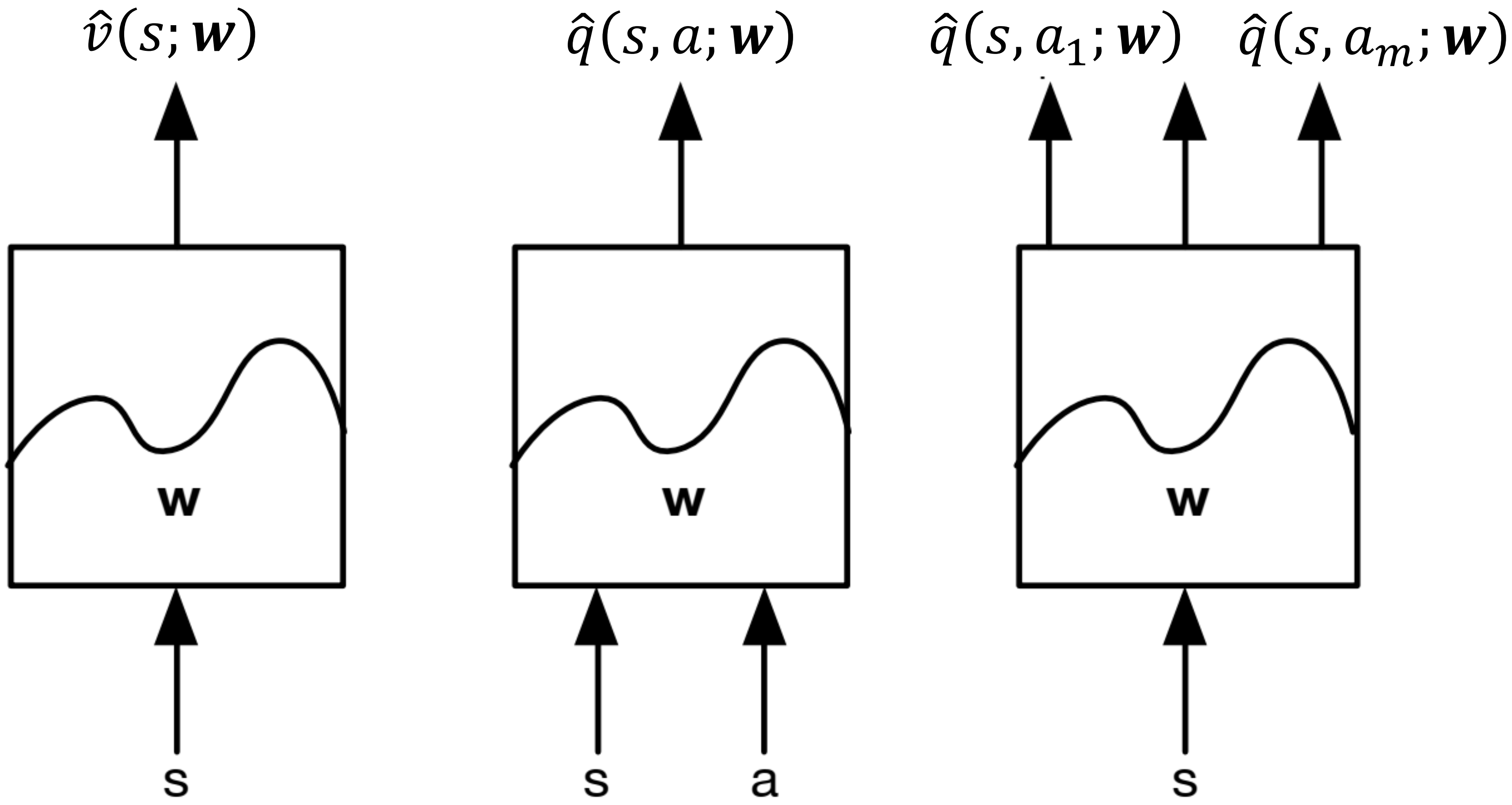
The *new* approach

- ✓Estimate value function with function approximation

$$\hat{v}(s; \boldsymbol{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a; \boldsymbol{w}) \approx q_\pi(s, a)$$

- ✓Generalise from seen states to unseen states
- ✓Update parameters $\boldsymbol{w}$ using MC or TD learning

Value Function Approximation Approaches



Which Function Approximator?

- ✓ Linear combinations of features
- ✓ Neural network
- ✓ Decision tree
- ✓ Nearest neighbour
- ✓ Fourier / wavelet bases
- ✓ …

Which Function Approximator?

- ✓ Linear combinations of features
- ✓ Neural network
- ✓ Decision tree
- ✓ Nearest neighbour
- ✓ Fourier / wavelet bases
- ✓ …

Focus on differentiable methods

In addition, we require training methods suitable for

- ✓ non-stationary data
- ✓ non-iid data

Incremental Methods

Stochastic Gradient Descent in Value Approximation

- ✓ Goal - Find parameter vector $\boldsymbol{w}$ minimising mean-squared error between approximate value $\hat{v}(s;\boldsymbol{w})$ and true value function $v_\pi(s)$

$$J(\boldsymbol{w}) = \mathbb{E}_\pi\left[\left(v_\pi(S) - \hat{v}(S;\boldsymbol{w})\right)^2\right]$$

- ✓ Gradient solution

$$\Delta\boldsymbol{w} = -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}}J(\boldsymbol{w}) = \alpha\mathbb{E}_\pi[v_\pi(S) - \hat{v}(S;\boldsymbol{w})]\nabla_{\boldsymbol{w}}\hat{v}(S;\boldsymbol{w})$$

- ✓ Stochastic approximation (sampling)

$$\Delta\boldsymbol{w} = \alpha\left(v_\pi(S) - \hat{v}(S;\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{v}(S;\boldsymbol{w})$$

Feature Vector State

- ✓Represent state by a feature vector

$$\boldsymbol{x}(S) = \begin{bmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{bmatrix}$$

For example:

- ✓Robot distance from landmarks
- ✓Trends in the stock market
- ✓Piece configurations in chess
- ✓Neural embedding

Linear Value Function Approximation

- ✓ Value as a linear combination of state features

$$\hat{v}(S;\boldsymbol{w}) = \boldsymbol{x}(S)^T\boldsymbol{w}$$

- ✓ Objective function is quadratic in parameters w
  - ✓ Stochastic gradient descent converges to global optimum
  - ✓ Simple (LMS) update rule

$$\nabla_{\boldsymbol{w}}\hat{v}(S;\boldsymbol{w}) = \boldsymbol{x}(S)$$

$$\Delta\boldsymbol{w} = \alpha\big(v_\pi(S) - \hat{v}(S;\boldsymbol{w})\big)\boldsymbol{x}(S)$$

Lookup Table & Features

- ✓A special case of linear value function approximation
- ✓Using table lookup features

$$\boldsymbol{x}(S) = \begin{bmatrix} \mathbf{1}(S; s_1) \\ \vdots \\ \mathbf{1}(S; s_n) \end{bmatrix}$$

- ✓Parameter vector $\boldsymbol{w}$ values each state

$$\hat{v}(S; \boldsymbol{w}) = \begin{bmatrix} \mathbf{1}(S; s_1) \\ \vdots \\ \mathbf{1}(S; s_n) \end{bmatrix}^T \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

Incremental Prediction

Incremental Prediction Algorithms

- ✓ So far have assumed access to true $v_\pi(s)$, but in RL there is no supervisor, only rewards
- ✓ In practice, we substitute a target for $v_\pi(s)$
  - ✓ MC - Target is the return G_t
  $$\Delta \boldsymbol{w} = \alpha\left(G_t - \hat{v}(S_t;\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{v}(S_t;\boldsymbol{w})$$
  - ✓ TD(0) - Target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1};\boldsymbol{w})$
  $$\Delta \boldsymbol{w} = \alpha\left(R_{t+1} + \gamma\hat{v}(S_{t+1};\boldsymbol{w}) - \hat{v}(S_t;\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{v}(S_t;\boldsymbol{w})$$
  - ✓ TD(λ) - Target is the λ-return G_t^λ
  $$\Delta \boldsymbol{w} = \alpha\left(G_t^\lambda - \hat{v}(S_t;\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{v}(S_t;\boldsymbol{w})$$

Value Function Approximation - MC

- ✓ Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- ✓ Can therefore apply supervised learning to training samples

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

- ✓ Linear Monte-Carlo policy evaluation

$$\Delta \boldsymbol{w} = \alpha\big(G_t - \hat{v}(S_t; \boldsymbol{w})\big)\nabla_{\boldsymbol{w}}\hat{v}(S_t; \boldsymbol{w}) = \alpha\big(G_t - \hat{v}(S_t; \boldsymbol{w})\big)\boldsymbol{x}(S_t)$$

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

Value Function Approximation – TD

- ✓ TD-target is a biased sample of true value $v_\pi(S_t)$
- ✓ Can still apply supervised learning to training samples

$$\langle S_1, R_2 + \gamma \hat{v}(S_2; \boldsymbol{w}) \rangle, \dots, \langle S_{T-1}, R_T \rangle$$

- ✓ Linear TD(0) policy evaluation

$$\Delta \boldsymbol{w} = \alpha \big(R + \gamma \hat{v}(S'; \boldsymbol{w}) - \hat{v}(S; \boldsymbol{w}) \big) \nabla_{\boldsymbol{w}} \hat{v}(S; \boldsymbol{w}) = \alpha \delta \boldsymbol{x}(S)$$

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

Value Function Approximation - TD(λ)

- ✓ λ-return G_t^λ is also a biased sample of true value $v_\pi(S_t)$
- ✓ Again supervised learning to training samples

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

- ✓ Forward view linear TD(λ)

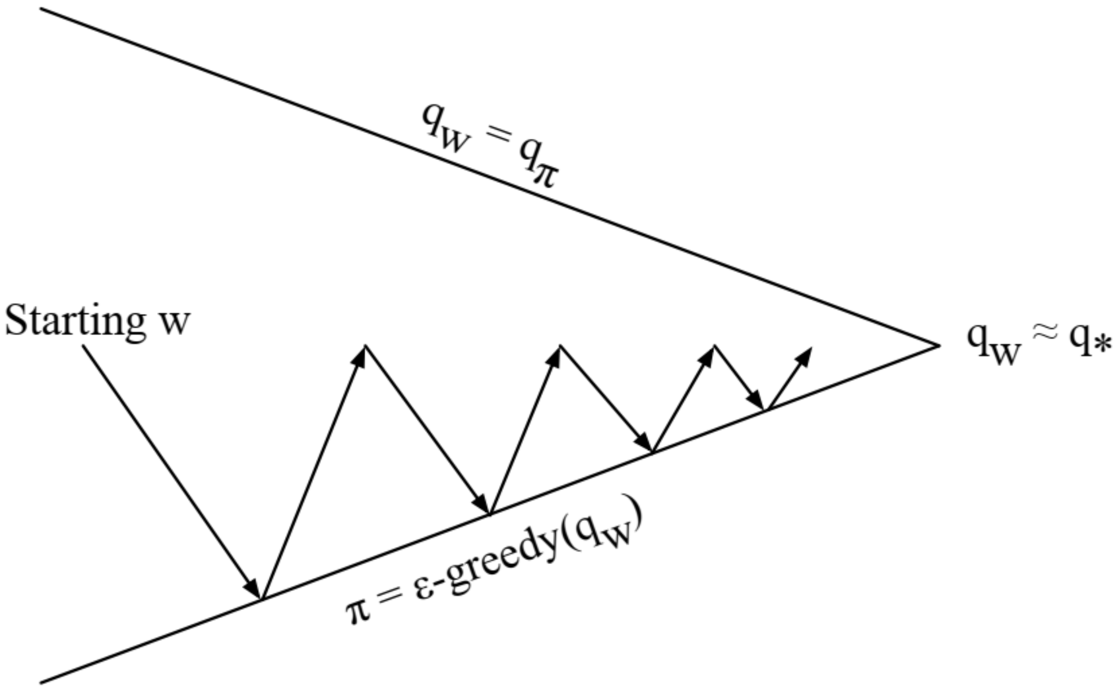
$$\Delta \boldsymbol{w} = \alpha \left(G_t^\lambda - \hat{v}(S_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S_t; \boldsymbol{w}) = \alpha \left(G_t^\lambda - \hat{v}(S_t; \boldsymbol{w}) \right) \boldsymbol{x}(S_t)$$

- ✓ Backward view linear TD(λ)

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}; \boldsymbol{w}) - \hat{v}(S_t; \boldsymbol{w})$$
$$E_t = \lambda \gamma E_{t-1} + \boldsymbol{x}(S_t)$$
$$\Delta \boldsymbol{w} = \alpha \delta_t E_t$$

Incremental Control

Control with Value Function Approximation



- ✓ Policy evaluation - Approximate policy evaluation , $\hat{q}(\cdot,\cdot;\boldsymbol{w}) \approx q_\pi(\cdot,\cdot)$
- ✓ Policy improvement - ϵ-greedy policy improvement

Action-Value Function Approximation

- ✓ Approximate the action-value function $\hat{q}(S, A; \boldsymbol{w}) \approx q_\pi(S, A)$
- ✓ Minimise MSE between approximate action-value $\hat{q}(S, A; \boldsymbol{w})$ and true action-value $q_\pi(S, A)$

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

- ✓ Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \left(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{q}(S, A; \boldsymbol{w})$$

$$\Delta\boldsymbol{w} = \alpha\left(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{q}(S, A; \boldsymbol{w})$$

Linear Action-Value Function Approximation

- ✓ Use table feature action-states

$$\boldsymbol{x}(S,A) = \begin{bmatrix} x_1(S,A) \\ \vdots \\ x_n(S,A) \end{bmatrix}$$

- ✓ Represent action-value function by linear combination of features

$$\hat{q}(S,A;\boldsymbol{w}) = \boldsymbol{x}(S,A)^T\boldsymbol{w}$$

- ✓ Stochastic gradient descent update

$$\nabla_{\boldsymbol{w}}\hat{q}(S,A;\boldsymbol{w}) = \boldsymbol{x}(S,A)$$

$$\Delta\boldsymbol{w} = \alpha\left(q_\pi(S,A) - \hat{q}(S,A;\boldsymbol{w})\right)\boldsymbol{x}(S,A)$$

Incremental Control Algorithms

Again we need a non-oracular target for $q_\pi(S, A)$

✓MC - Target is return return G_t

$$\Delta \mathbf{w} = \alpha\big(G_t - \hat{q}(S_t, A_t; \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(S_t, A_t; \mathbf{w})$$

✓TD(0) - Target is the TD target $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}; \mathbf{w})$

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

✓Forward TD(λ) - Target is the action-value λ-return

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

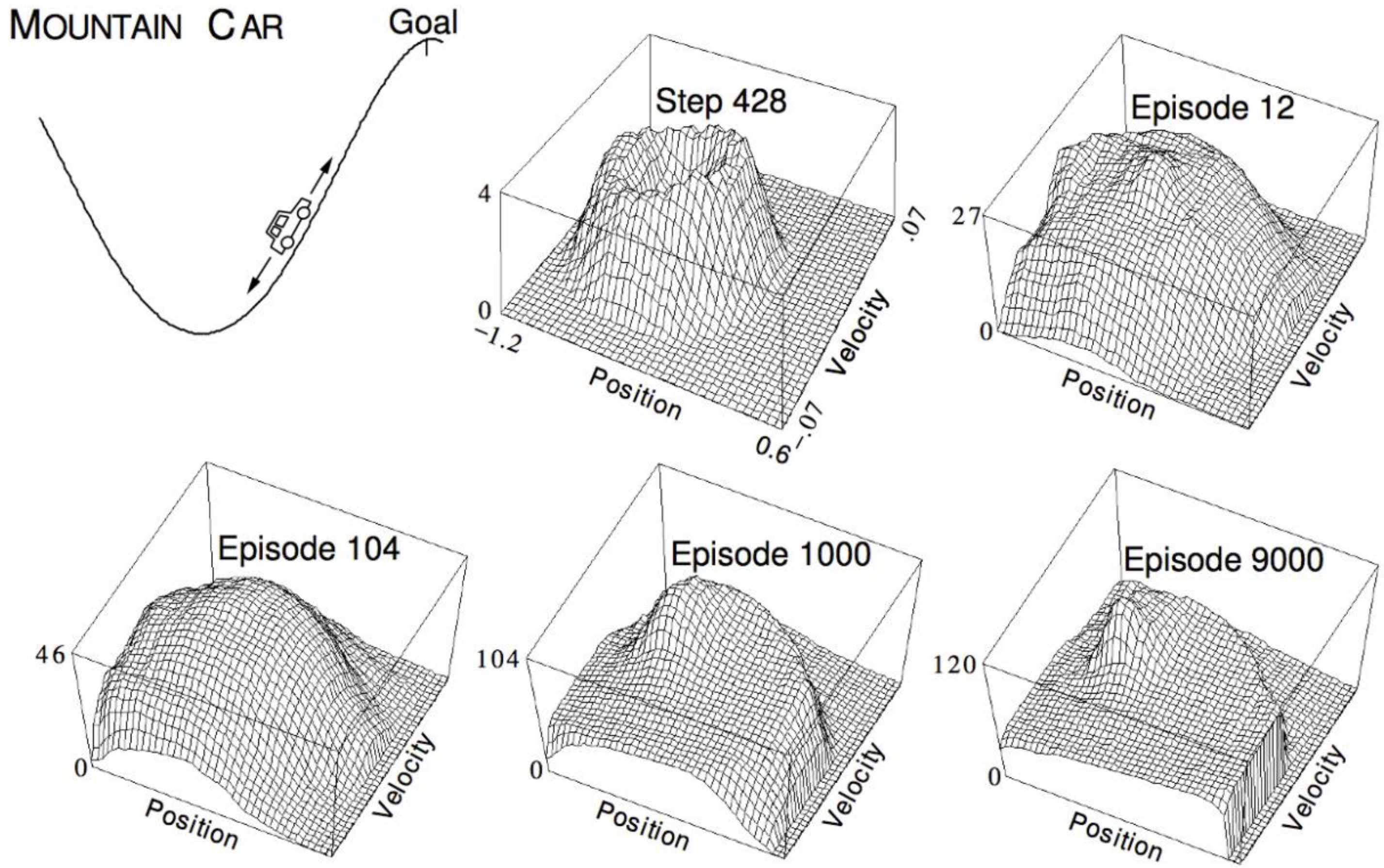
✓Backward TD(λ) - Equivalent target

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

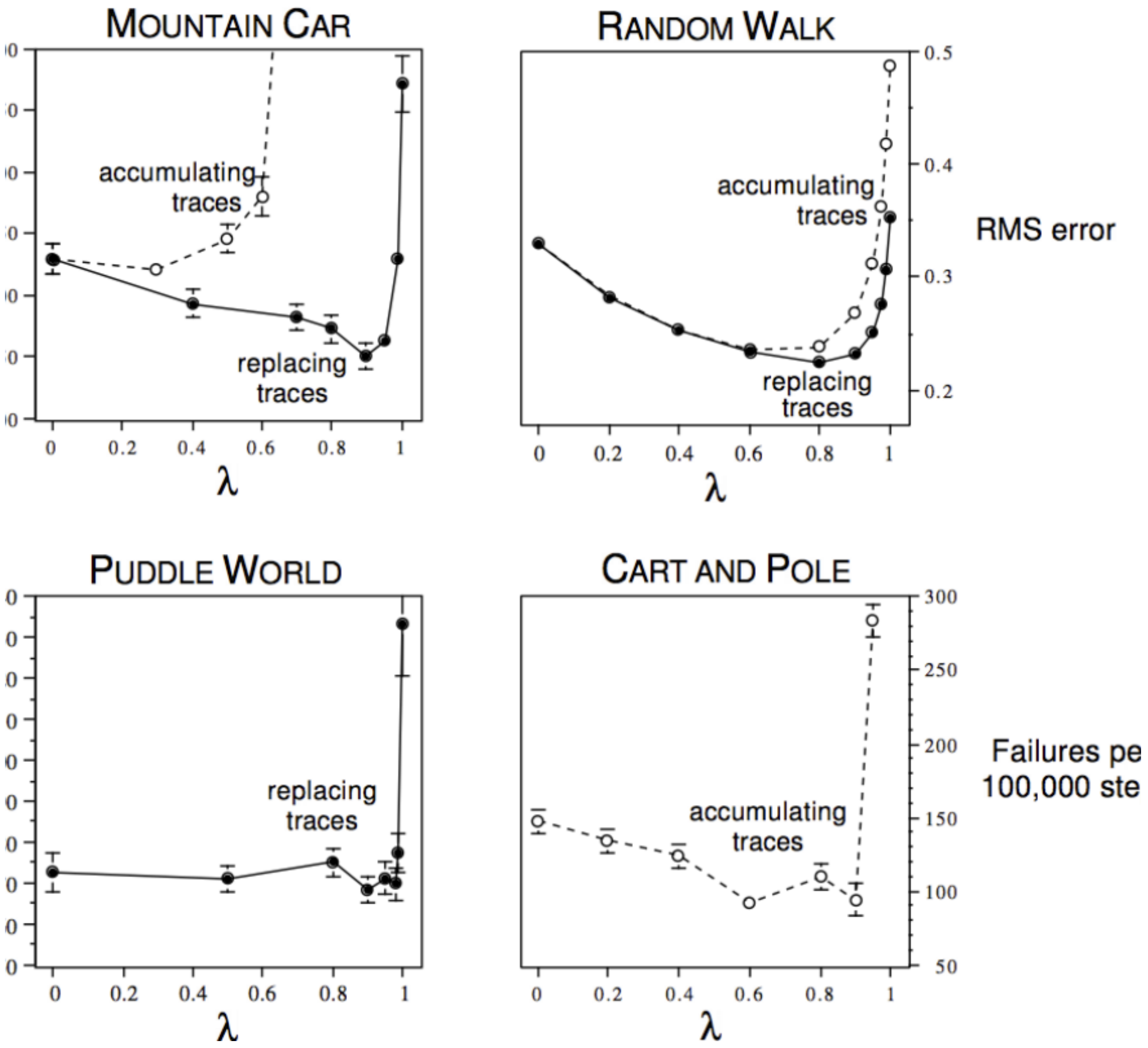
$$E_t = \lambda\gamma E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t; \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha\delta_t E_t$$

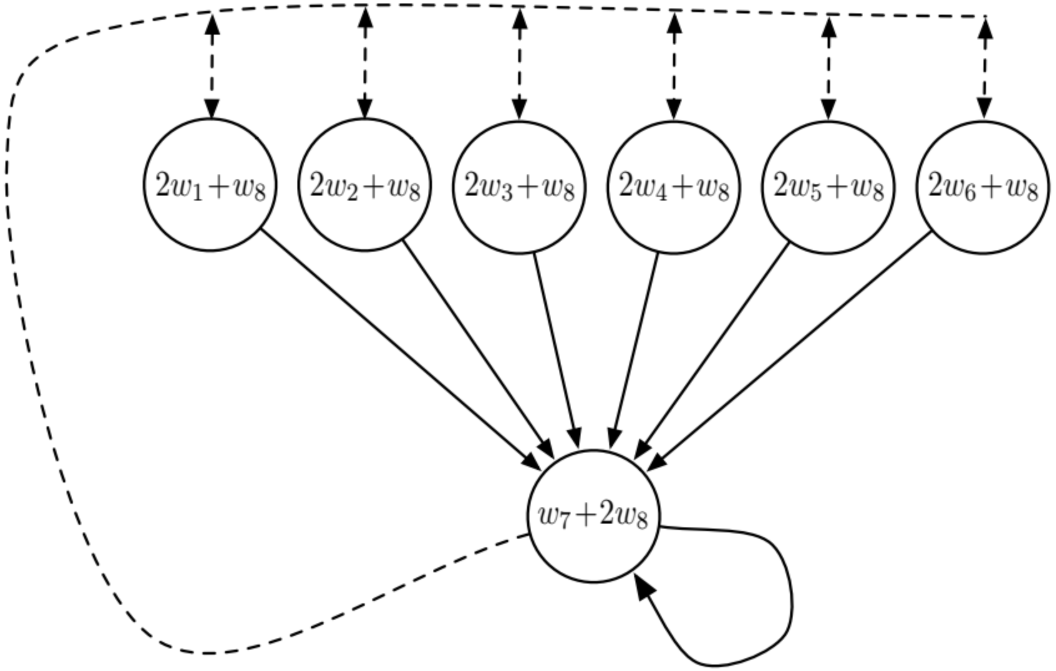
Linear SARSA with Coarse Coding in Mountain Car



Study of λ: Should We Bootstrap?



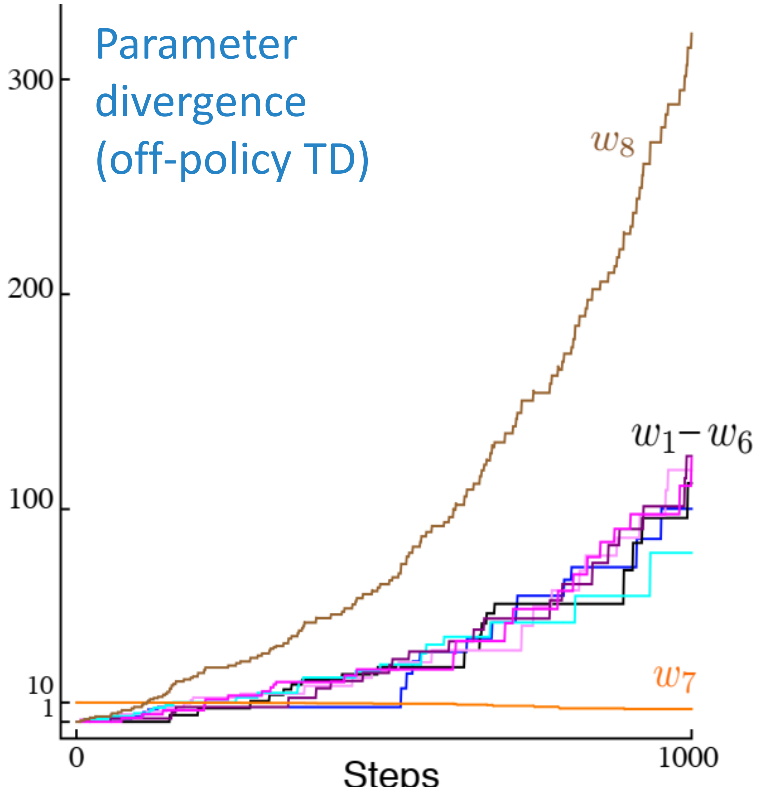
Baird's Counterexample (SB Book)



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

$b(\text{dashed}|\cdot) = 6/7$
$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$



The Deadly Triad

- ✓ Function approximation
- ✓ Bootstrapping
- ✓ Off policy training

Convergence of Prediction Algorithms

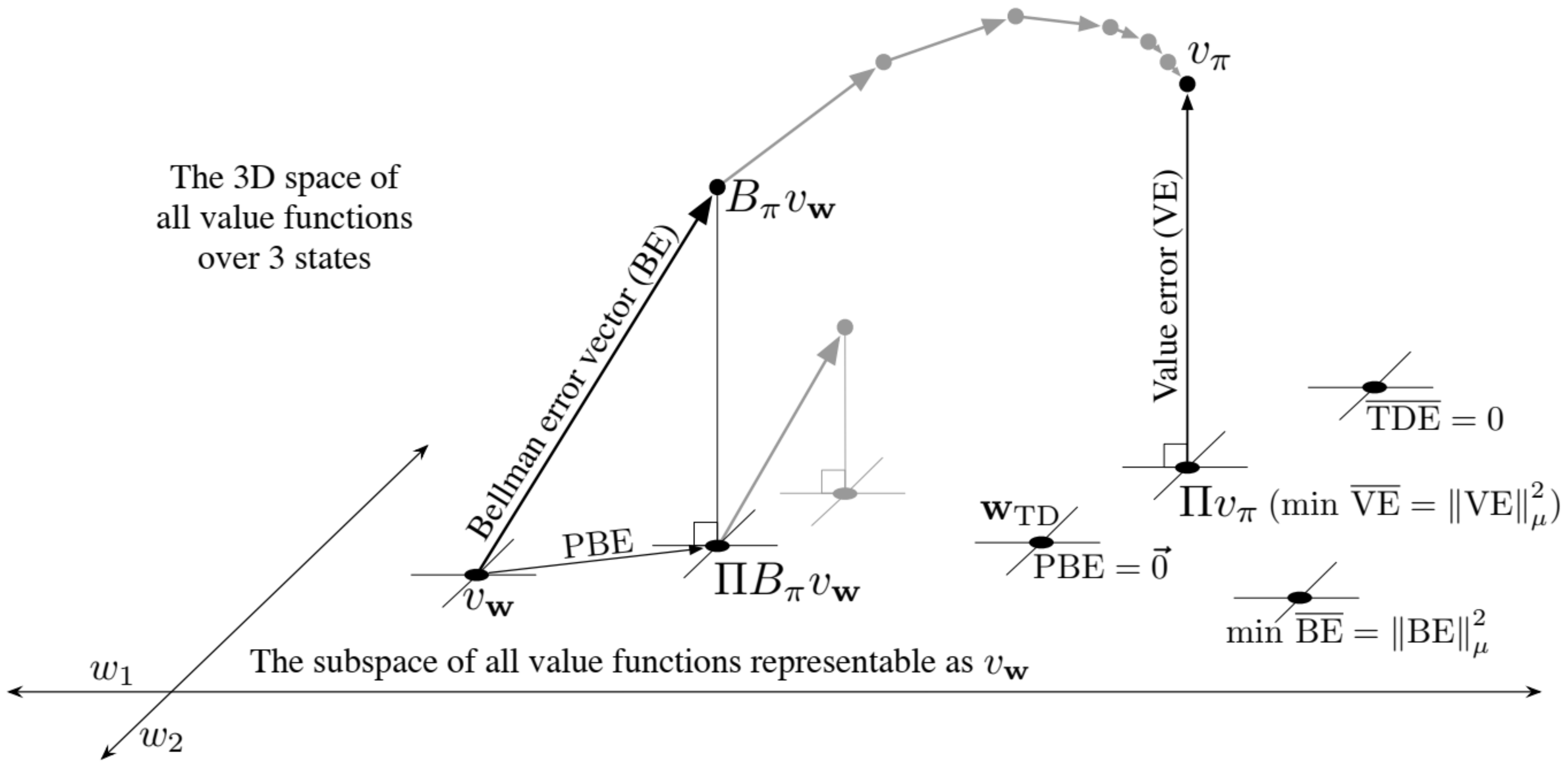
On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Gradient Temporal-Difference Learning

- ✓TD does not follow the gradient of any objective function
- ✓TD can diverge when off-policy or using non-linear function approximation
- ✓Gradient TD performs SGD in the projected Bellman error

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

Projected Bellman Error



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

Batch Methods

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 $\hat{v}(s; \boldsymbol{w}) \approx v_\pi(s)$ and experience $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle S_1, v_1^\pi \rangle, \langle S_2, v_2^\pi \rangle, \dots, \langle S_T, v_T^\pi \rangle\}$$

- ✓Which parameters $\boldsymbol{w}$ give the best fitting value function $\hat{v}(s; \boldsymbol{w})$?

- ✓Least squares algorithms find parameter vector $\boldsymbol{w}$ minimising sum-squared error between $\hat{v}(s; \boldsymbol{w})$ and target values $v_\pi(s)$

$$LS(\boldsymbol{w}) = \sum_{t=1}^{T} \left(v^\pi - \hat{v}(s_t; \boldsymbol{w})\right)^2 = \mathbb{E}_{\mathcal{D}}\left[\left(v^\pi - \hat{v}(s; \boldsymbol{w})\right)^2\right]$$

Experience Replay

SGD with Experience Replay

Given value function approximation $\hat{v}(s;\boldsymbol{w}) \approx v_\pi(s)$ and experience $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle S_1, v_1^\pi \rangle, \langle S_2, v_2^\pi \rangle, \dots, \langle S_T, v_T^\pi \rangle\}$$

Repeat

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha (v^\pi - \hat{v}(s;\boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(s;\boldsymbol{w})$$

Converges to least squares solution

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} LS(\boldsymbol{w})$$

Deep Q-Networks (DQN)

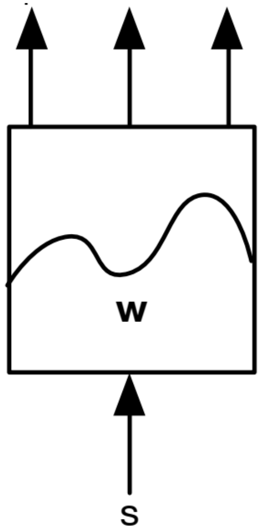
DQN uses experience replay and fixed Q-targets

- ✓ Take action a_t according to ϵ-greedy policy
- ✓ Store transition $(s_\mathrm{t}, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- ✓ Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- ✓ Compute Q-learning targets with respect to old fixed parameters $\boldsymbol{w}^{-}$
- ✓ Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^{-}) - Q(s, a; w_i)\right)^2\right]$$

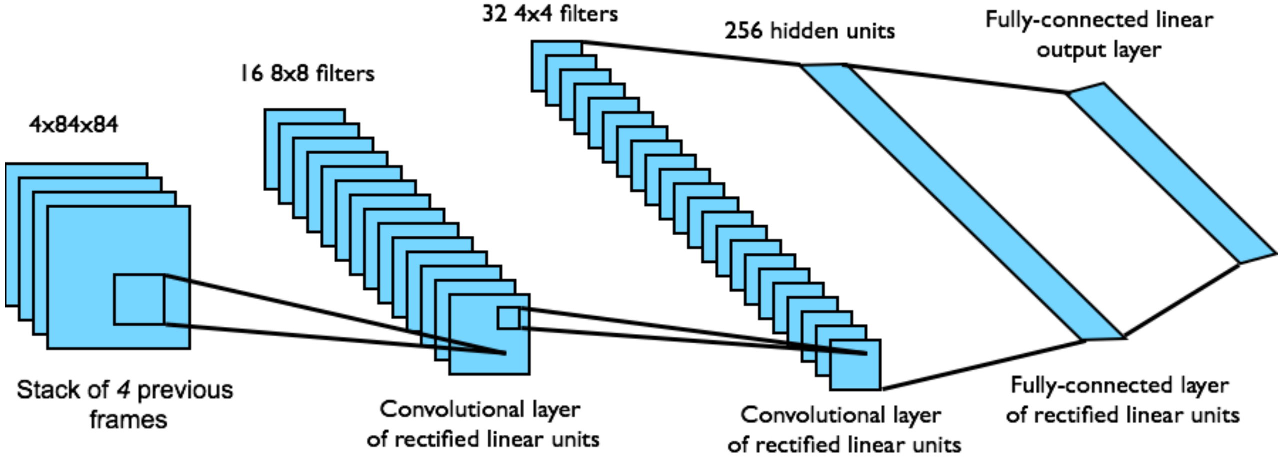
- ✓ Using variant of stochastic gradient descent

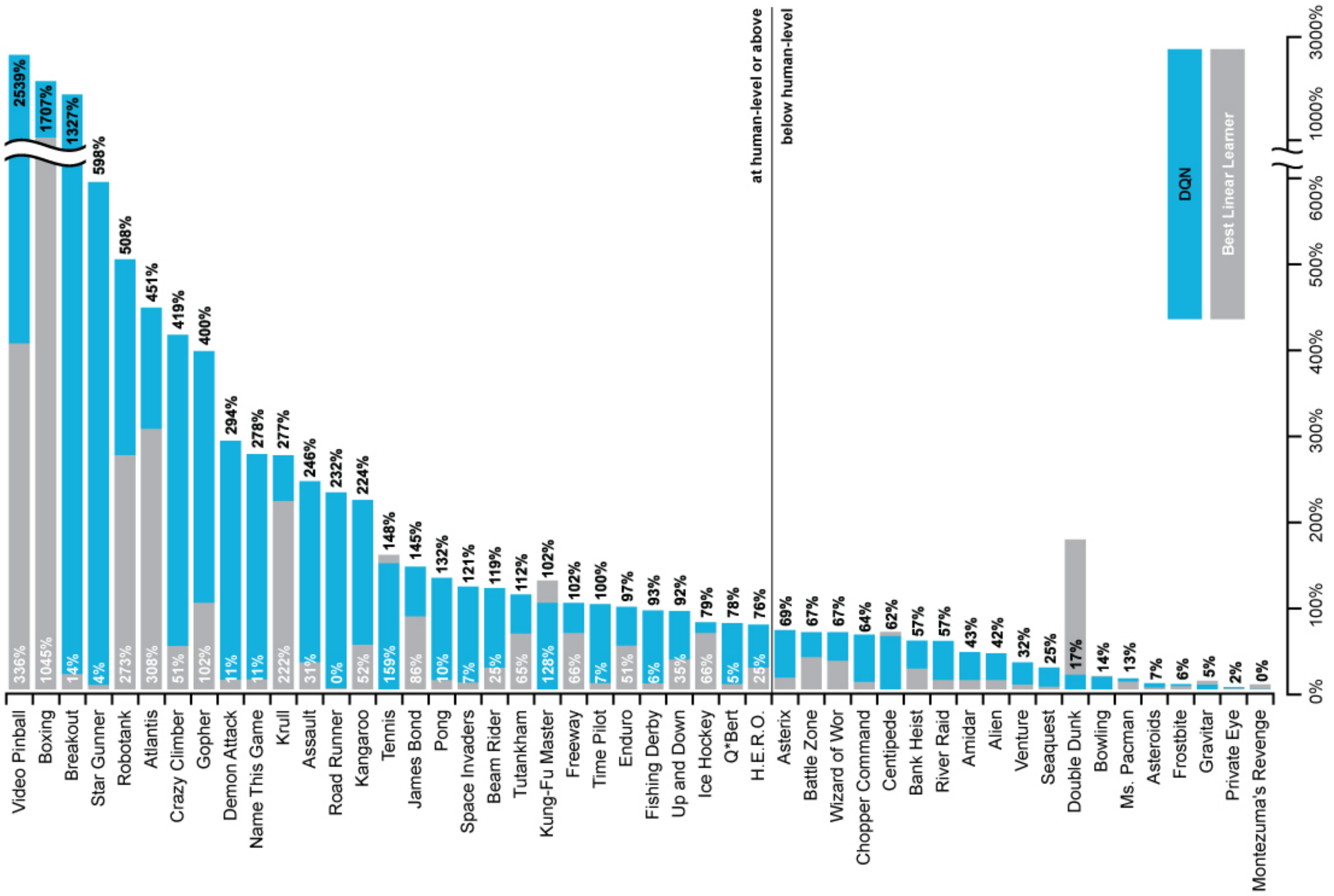
Atari-DQN



- ✓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





Atari-DQN Results

Atari-DQN Ablation Study

	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

Improvements on Original DQN

- ✓Double DQN - Remove upward bias caused by $\max_{\mathrm{a}} Q(s, a; \boldsymbol{w})$
  - ✓Current Q-network $\boldsymbol{w}$ is used to select actions
  - ✓Older Q-network $\boldsymbol{w}^{-}$ is used to evaluate actions

$$\mathcal{L}_i = \left(r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \boldsymbol{w}); \boldsymbol{w}^{-}\right) - Q(s, a; \boldsymbol{w}) \right)^2$$

- ✓Prioritised replay - Weight experience according to surprise
  - ✓ Store experience in priority queue according to DQN error
- ✓Duelling network - Split Q-network into two channels
  - ✓Action-independent value function $V(s)$
  - ✓Action-dependent advantage function $A(s, a; \boldsymbol{w})$

$$Q(s, a) = V(s) + A(s, a; \boldsymbol{w})$$

Least Squares Prediction and Control

Linear Least Squares Prediction

- ✓ Experience replay finds least squares solution, but it may take many iterations
- ✓ Using linear value function approximation $\hat{v}(s; \boldsymbol{w}) = \mathbf{x}(\mathrm{s})^{\mathrm{T}}\mathbf{w}$
- ✓ We can solve the least squares solution directly

Linear Least Squares Prediction - Batch

- ✓ At minimum of $LS(\boldsymbol{w})$, the expected update must be zero

$$\mathbb{E}_{\mathcal{D}}[\Delta \boldsymbol{w}] = 0$$

$$\alpha \sum_{t=1}^{T} \boldsymbol{x}(S_t)(v_t^{\pi} - \boldsymbol{x}(S_t)^T \boldsymbol{w}) = 0$$

$$\boldsymbol{w} = \left(\sum_{t=1}^{T} \boldsymbol{x}(S_t)\boldsymbol{x}(S_t)^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t) v_t^{\pi}$$

- ✓ Usual direct (N^3) and incremental (N^2) costs

Linear Least Squares Prediction - Algorithms

- ✓Again: we do not know true v_t^π
- ✓Training data use noisy or biased samples of v_t^π
  - ✓Least Squares Monte-Carlo (LSMC) uses return $v_t^\pi \approx G_t$
  - ✓Least Squares TD (LSTD) uses TD target $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}; \boldsymbol{w})$
  - ✓Least Squares TD(λ) (LSTD(λ)) uses λ-return $v_t^\pi \approx G_t^\lambda$
- ✓In each case solve directly for fixed point of MC / TD / TD(λ)

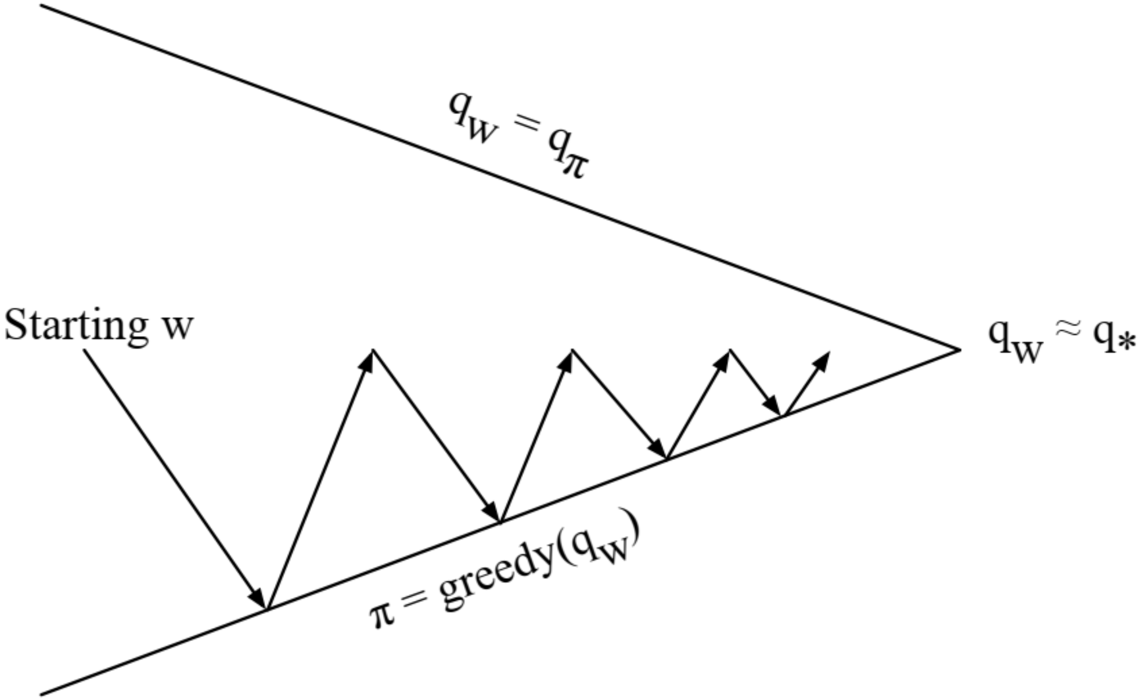
LS Updates

LSMC	$\boldsymbol{w} = \left(\sum_{t=1}^{T} \boldsymbol{x}(S_t) \boldsymbol{x}(S_t)^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t) G_t$
LSTD	$\boldsymbol{w} = \left(\sum_{t=1}^{T} \boldsymbol{x}(S_t) \left(\boldsymbol{x}(S_t) - \gamma \boldsymbol{x}(S_{t+1}) \right)^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t) R_{t+1}$
LSTD(λ)	$\boldsymbol{w} = \left(\sum_{t=1}^{T} E_t \left(\boldsymbol{x}(S_t) - \gamma \boldsymbol{x}(S_{t+1}) \right)^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t) R_{t+1}$

Convergence of Linear Least Squares Prediction Algorithms

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

Least Squares Control



- ✓ Policy evaluation - Policy evaluation by least squares Q-learning
- ✓ Policy improvement - Greedy policy improvement

Least Squares Action-Value Function Approximation

- ✓Approximate $q_\pi(s,a)$ using linear combination of features $\boldsymbol{x}(s,a)$

$$\hat{q}(s,a;\boldsymbol{w}) = \boldsymbol{x}(s,a)^T\boldsymbol{w} \approx q_\pi(s,a)$$

- ✓Minimise least squares error between $\hat{q}(s,a;\boldsymbol{w})$ and $q_\pi(s,a)$
  - ✓From experience generated using policy π
  - ✓Consisting of $\langle(state, action), value\rangle$ pairs

$$\mathcal{D} = \{\langle(s_1,a_1), v_1^\pi\rangle, \dots, \langle(s_T,a_T), v_T^\pi\rangle\}$$

Least Squares Control

- ✓For policy evaluation, we want to efficiently use all experience
- ✓For control, we also want to improve the policy
- ✓This experience is generated from many policies
- ✓So to evaluate $q_\pi(S, A)$ we must learn off-policy
- ✓We use the same idea as Q-learning:
  - ✓Use experience generated by old policy $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$
  - ✓Consider alternative successor action $A' \sim \pi_{new}(S_{t+1})$
  - ✓Update $\hat{q}(S_t, A_t; \mathbf{w})$ towards value of alternative action $R_{t+1} + \gamma \hat{q}(S_{t+1}, A'; \mathbf{w})$

Least Squares Q-Learning

Consider the following linear Q-learning update

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}); \boldsymbol{w}) - \hat{q}(S_t, A_t; \boldsymbol{w})$$

$$\Delta \boldsymbol{w}_t = \alpha \delta_t \boldsymbol{x}(S_t, A_t)$$

LSTDQ algorithm: solve for total update equal to zero

$$\Delta \boldsymbol{w}_t = 0$$

$$\boldsymbol{w} = \left(\sum_{t=1}^{T} \boldsymbol{x}(S_t, A_t) \big(\boldsymbol{x}(S_t, A_t) - \gamma \boldsymbol{x}(S_{t+1}, \pi(S_{t+1})) \big)^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t, A_t) R_{t+1}$$

Convergence of Control Algorithms

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

(✓) = chatters around near-optimal value function

Wrap-up

Take home messages

- ✓ Value function approximation: scaling up RL to real-world sized problems
- ✓ Use MC, TD and TD(λ) to generate sample experiences for training the (action) value approximators
  - ✓ Stochastic Gradient Descent (incremental)
  - ✓ Least Squares (batch)
- ✓ Experience replay store data and updates with targets from old parameters (DQN)
- ✓ Linear least squares provides a closed form solution
- ✓ Convergence of learning might be tricky

Next

Policy Gradients (Parts I, II & III)

✓Parameterise the policy NOT the value function

✓REINFORCE (Monte-Carlo Policy Gradient)

✓Natural gradient

✓ TRPO & PPO