Reinforcement Learning

Value Function Approximation

Michael Herrmann, David Abel

Based on slides by Stefano V. Albrecht

25 February 2025

THE UNIVERSITY of EDINBURGH
informatics

Lecture Outline

- Curse of dimensionality and generalisation
- Value function approximation
- Stochastic gradient descent
- Linear value functions and feature construction
- Semi-gradient TD control

Curse of Dimensionality

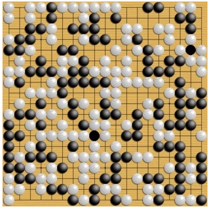
Theory so far has assumed:

- **Unlimited space:** can store value function as table
- **Unlimited data:** many (infinite) visits to all state-action pairs

In practice these assumptions are usually violated, because...

Curse of Dimensionality:

- Number of states grows *exponentially* with number of state variables
- If state described by k variables with values in $\{1, ..., n\}$, then $O(n^k)$ states



$>$

Go: 10^{170} states

Hydrogen atoms: 10^{80}

Compact Value Functions and Generalisation

Two problems...

Compact Value Functions and Generalisation

Two problems...

Not enough memory to store value function as table

- Tabular methods require storage proportional to $|\mathcal{S}|$ for $v(s)$ or $|\mathcal{S}||\mathcal{A}|$ for $q(s, a)$
- Need compact representation of value function

  (But sometimes can be enough to store only partial value function; e.g. MCTS)

Compact Value Functions and Generalisation

Two problems...

Not enough memory to store value function as table

- Tabular methods require storage proportional to $|\mathcal{S}|$ for $v(s)$ or $|\mathcal{S}||\mathcal{A}|$ for $q(s, a)$
- Need compact representation of value function

  (But sometimes can be enough to store only partial value function; e.g. MCTS)

No data (or not enough data) to estimate return in each state

- Many states may never be visited
- Need to generalise observations to unknown state-action pairs

Generalisation (Example)

Blue circle must move to red goal

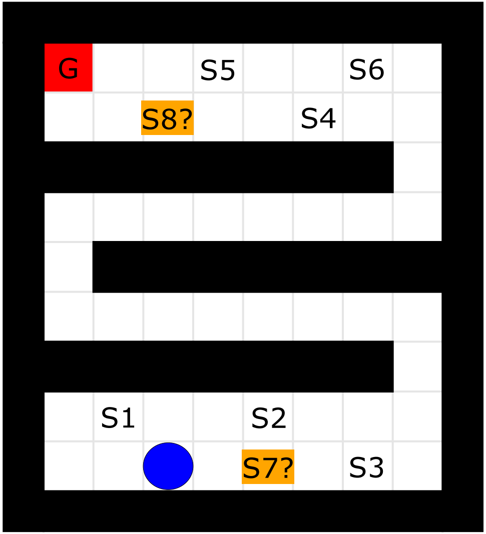
- Agent uses optimal policy (shortest path)

Suppose we have return estimates (steps to go) for locations $S1$–$S6$

- e.g. $v(S5) = -3$, $v(S4) = -6$, $v(S2) = -31$

We have no data for locations $S7$ and $S8$ (not visited yet)

- Can we estimate $v(S7)$ and $v(S8)$ based on other return estimates?



Value Function Approximation

Replace tabular value function with parameterised function:

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

$\mathbf{w} \in \mathbb{R}^d$ is parameter ("weight") vector

e.g. linear function, neural network, regression tree, ...

- **Compact:** number of parameters d much smaller than $|\mathcal{S}|$
- **Generalises:** changing one parameter value may change value estimate of many states/actions

Supervised Learning

Learning a value function is a form of supervised learning:

Examples are pairs of states and return estimates, (S_t, U_t), e.g.

- MC: $U_t = G_t$
- TD(0): $U_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)$
- n-step TD: $U_t = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \, \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1})$

Supervised Learning

Desired properties in supervised learning method:

- Incremental updates

  update **w** using only partial data, e.g. most recent (S_t, U_t) or batch

Supervised Learning

Desired properties in supervised learning method:

- Incremental updates
  update **w** using only partial data, e.g. most recent (S_t, U_t) or batch
- Ability to handle noisy targets
  e.g. different MC updates G_t for same state S_t

Supervised Learning

Desired properties in supervised learning method:

- Incremental updates
  update $\mathbf{w}$ using only partial data, e.g. most recent (S_t, U_t) or batch
- Ability to handle noisy targets
  e.g. different MC updates G_t for same state S_t
- Ability to handle non-stationary targets
  e.g. changing target policy, bootstrapping

$\Rightarrow$ If $\hat{v}$ or $\hat{q}$ differentiable, **stochastic gradient descent** is a suitable approach

Gradient Descent

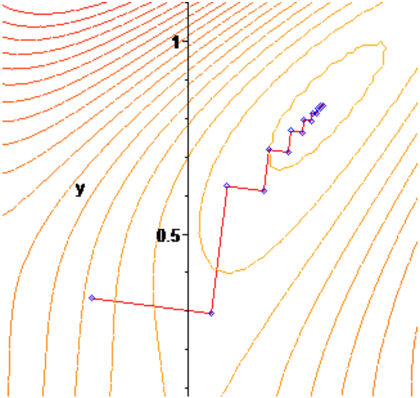
- Let $J(\mathbf{w})$ be differentiable function of $\mathbf{w}$
- Gradient of $J(\mathbf{w})$ is

$$\nabla J(\mathbf{w}) = \left(\frac{\partial J(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial J(\mathbf{w})}{\partial w_d} \right)^\top$$

- To find local minimum of $J(\mathbf{w})$, adjust $\mathbf{w}$ in negative direction of gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2} \alpha \nabla J(\mathbf{w}_t)$$

- α is step-size parameter
  convergence requires standard α-reduction



Example: Gradient Bandit Algorithm

- Can we select actions without computing estimates of q_*?

See Lecture 2

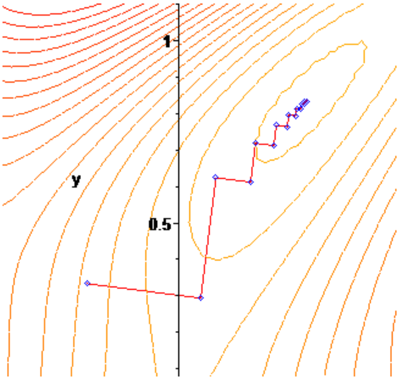
Gradient-based policy optimisation:

- Use differentiable policy $\pi_t(a|\theta)$ with parameter vector $\theta \in \mathbb{R}^d$

$$\pi_t(a|\theta) = \Pr\{A_t = a \mid \theta_t = \theta\}$$

- Use gradient ascent on policy parameters to maximise expected reward

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} \mathbb{E}[R_t]$$



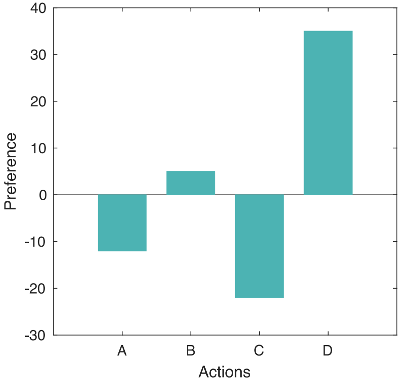
Gradient Bandit Algorithm with Softmax

See Lecture 2

- Represent π_t with softmax distribution:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}$$

$H_t(a)$ are preference values (parameters)



- Update policy parameters:

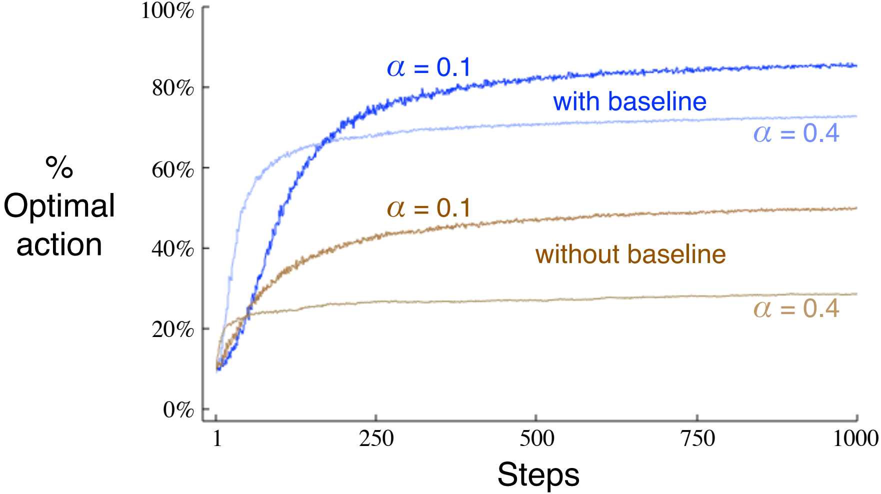
$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$$

$$= H_t(a) + \alpha(R_t - \bar{R}_t)([a = A_t]_1 - \pi_t(a))$$

with baseline $\bar{R}_t = \frac{1}{t}\sum_{\tau=1}^{t} R_\tau = \bar{R}_{t-1} + \frac{1}{t}\left(R_t - \bar{R}_{t-1}\right)$ which reduces variance in updates

Gradient Bandit Algorithm

See Lecture 2



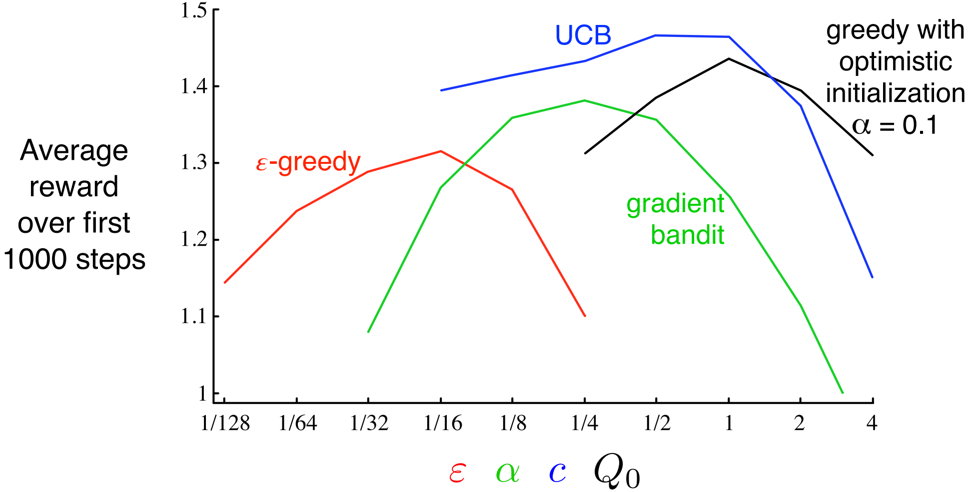
$\bar{R}_t = \frac{1}{t} \sum_\tau R_\tau$

$\bar{R}_t = 0$

Baseline reduces variance in updates

Summary: Comparing Gradient Bandits with other Bandit Algorithms

See Lecture 2



Stochastic Gradient Descent (back to general case)

Objective: find parameter vector $\mathbf{w}$ by minimising *mean-squared error* between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

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

Stochastic Gradient Descent

Objective: find parameter vector $\mathbf{w}$ by minimising *mean-squared error* between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

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

- Gradient descent finds local minimum:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\,\nabla J(\mathbf{w}_t) \\ &= \mathbf{w}_t + \alpha\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}_t))\,\nabla\hat{v}(s, \mathbf{w}_t)] \end{aligned}$$

Stochastic Gradient Descent

Objective: find parameter vector $\mathbf{w}$ by minimising *mean-squared error* between approximate value $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$

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

- Gradient descent finds local minimum:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha \nabla J(\mathbf{w}_t) \\ &= \mathbf{w}_t + \alpha\mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}_t)) \nabla \hat{v}(s, \mathbf{w}_t)]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient:

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

Stochastic Gradient Descent — Convergence

Stochastic gradient descent *samples* the gradient:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{1}$$

Stochastic Gradient Descent — Convergence

Stochastic gradient descent *samples* the gradient:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{1}$$

- $\mathbf{w}_t$ will converge to local optimum under standard α-reduction and if U_t is unbiased estimate $\mathbb{E}_\pi[U_t | S_t] = v_\pi(S_t)$

  $\Rightarrow$ MC update is unbiased, but TD update is biased (why?)

Stochastic Gradient Descent — Convergence

Stochastic gradient descent *samples* the gradient:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{1}$$

- $\mathbf{w}_t$ will converge to local optimum under standard α-reduction and if U_t is unbiased estimate $\mathbb{E}_\pi[U_t | S_t] = v_\pi(S_t)$

  $\Rightarrow$ MC update is unbiased, but TD update is biased (why?)

- Note: (1) is not a true TD gradient because U_t also depends on $\mathbf{w}$

  $$U_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$$

  Hence, we call it semi-gradient TD

Semi-gradient TD(0) for Policy Evaluation

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

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

Linear Value Function Approximation

See Tutorial 5

Linear value function approximation:

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

- $\mathbf{x}(s) = (x_1(s), ..., x_d(s))^\top$ is *feature vector* of state s
- Simple gradient: $\nabla \hat{v}(s, \mathbf{w}) = \left(\frac{\partial \mathbf{w}^\top \mathbf{x}}{\partial w_1}, \cdots, \frac{\partial \mathbf{w}^\top \mathbf{x}}{\partial w_d} \right)^\top = \mathbf{x}(s)$
- Gradient update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \mathbf{x}(S_t)$

Linear Value Function Approximation

See Tutorial 5

Linear value function approximation:

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

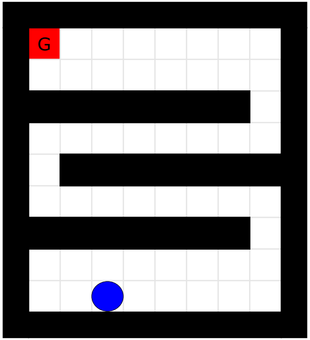
- $\mathbf{x}(s) = (x_1(s), ..., x_d(s))^\top$ is *feature vector* of state s
- Simple gradient: $\nabla \hat{v}(s, \mathbf{w}) = \left(\frac{\partial \mathbf{w}^\top \mathbf{x}}{\partial w_1}, \cdots, \frac{\partial \mathbf{w}^\top \mathbf{x}}{\partial w_d} \right)^\top = \mathbf{x}(s)$
- Gradient update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \mathbf{x}(S_t)$

In linear case, there is only one optimum!

$\Rightarrow$ MC gradient updates converge to global optimum

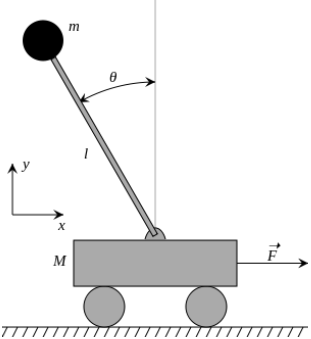
$\Rightarrow$ TD gradient updates converge *near* global optimum (TD fixed point)

Feature Vectors



$$\mathbf{x}(s) = \begin{pmatrix} \text{x-pos}(s) \\ \text{y-pos}(s) \end{pmatrix}$$

$$\mathbf{x}(s) = \begin{pmatrix} \theta(s) \\ \theta\text{-vel}(s) \\ \text{x-pos}(s) \\ \vdots \end{pmatrix}$$



Remember:
State must be Markov

State Aggregation

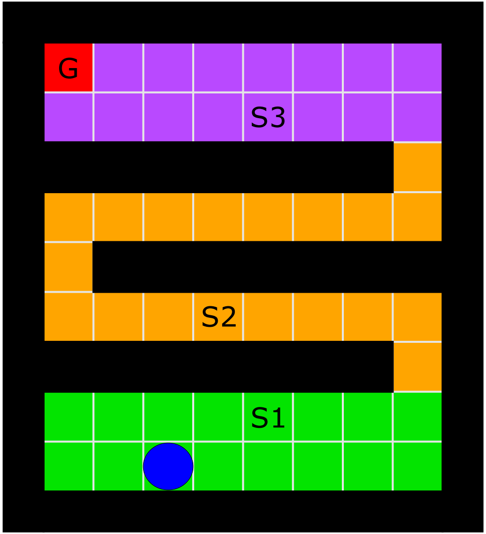
Exact representation:

$$\mathbf{x}(s) = \begin{pmatrix} \text{x-pos}(s) \\ \text{y-pos}(s) \end{pmatrix}$$

Generalise with state aggregation:

- Partition states into disjoint sets $\mathcal{S}_1$, $\mathcal{S}_2$, ...
  with indicator functions $\mathbf{x}_k(s) = [s \in \mathcal{S}_k]_1$

$$\mathbf{x}(s) = \begin{pmatrix} \text{in-S1}(s) \\ \text{in-S2}(s) \\ \text{in-S3}(s) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$



State Aggregation

Exact representation:

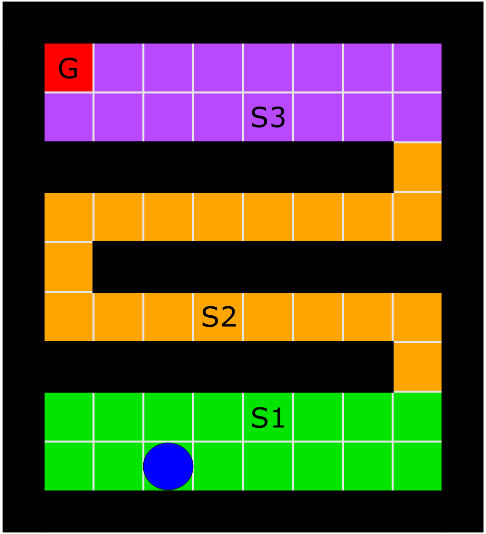
$$\mathbf{x}(s) = \begin{pmatrix} \text{x-pos}(s) \\ \text{y-pos}(s) \end{pmatrix}$$

Generalise with state aggregation:

- Partition states into disjoint sets $\mathcal{S}_1, \mathcal{S}_2, \ldots$ with indicator functions $\mathbf{x}_k(s) = [s \in \mathcal{S}_k]_1$

Special case: every state s has its own set $\mathcal{S}_s = \{s\}$

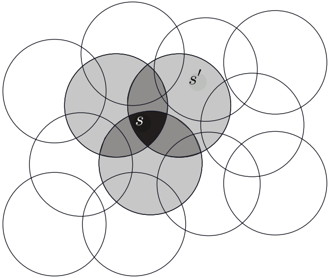
$\Rightarrow$ Same as tabular representation!

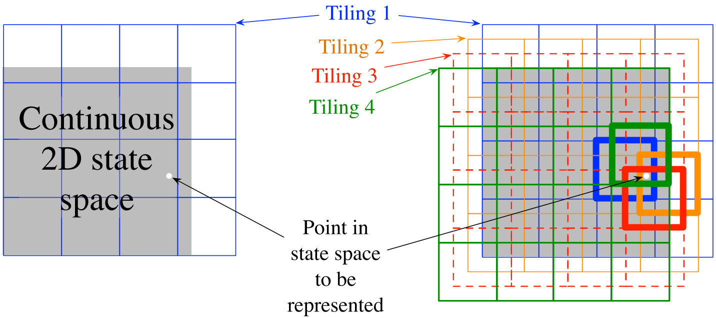


Coarse/Tile Coding

State aggregation generalises only within sets $\mathcal{S}_1$, $\mathcal{S}_2, \ldots$

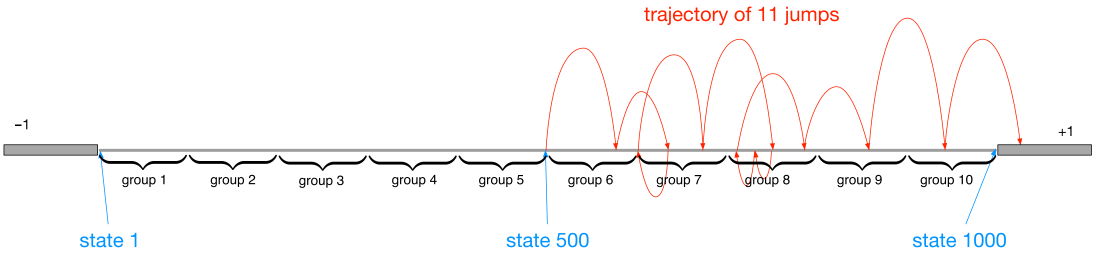
- Allow generalisation *across* sets by allowing $\mathcal{S}_k$ to overlap
- e.g. coarse coding and tile coding





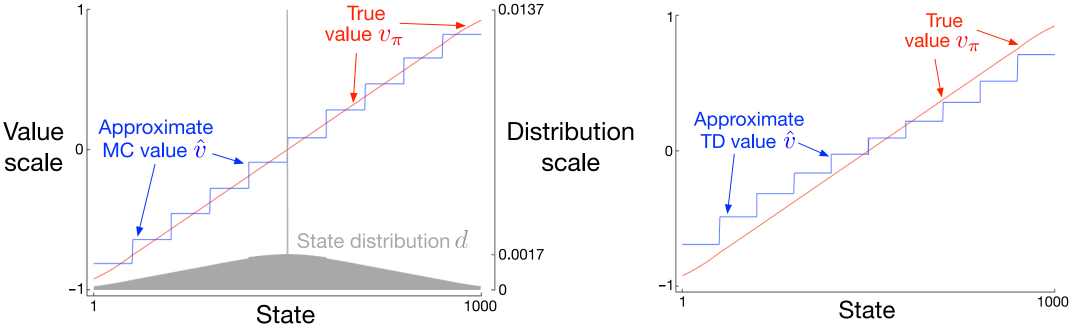
Example: Random Walk

- States: numbered 1 to 1000, start at state 500
- Policy: randomly jump to one of 100 states to left, or one of 100 states to right
- If jump goes beyond 1/1000, terminates with reward $-1/+1$
- State aggregation: 10 groups of 100 states each



Random Walk: MC and TD Prediction

Linear gradient MC: **Linear gradient TD:**



After 100,000 episodes with $\alpha = 2 \times 10^{-5}$

Approximate Control in Episodic Tasks

- Estimate state-action values: $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- For linear approx., features defined over states *and* action:

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

- Stochastic gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Approximate Control in Episodic Tasks

- Estimate state-action values: $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- For linear approx., features defined over states *and* action:

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

- Stochastic gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

e.g. **Sarsa:** $U_t = R_{t+1} + \gamma \, \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$

Q-learning: $U_t = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$

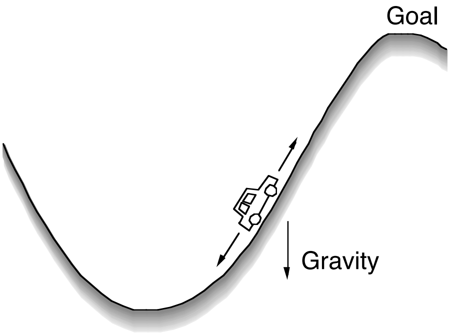
Expected Sarsa: $U_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \, \hat{q}(S_{t+1}, a, \mathbf{w}_t)$

Episodic Semi-gradient Sarsa

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)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

Example: Mountain Car with Linear Semi-Gradient Sarsa



STATES:
car's position and velocity

ACTIONS:
three thrusts: forward, reverse, none

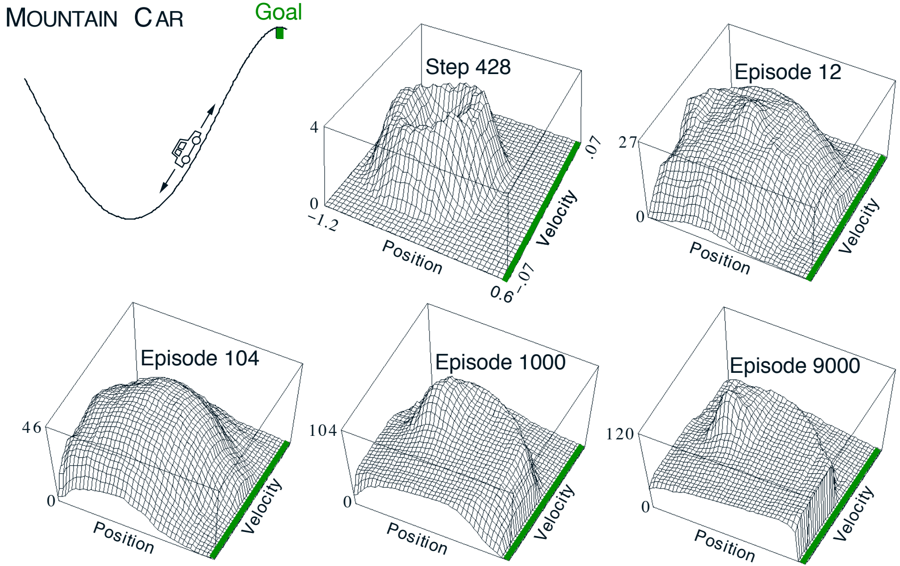
REWARDS:
always –1 until car reaches the goal

Episodic, No Discounting, γ=1

Semi-gradient Sarsa with linear approximation over 8 8x8 tilings

$\epsilon = 0$ (optimistic initial values $\hat{q}(s, a, \mathbf{w}) = 0$)

Learned Action Values in Mountain Car



Cost-to-go:

$- \max_a \hat{q}(s, a, \mathbf{w})$

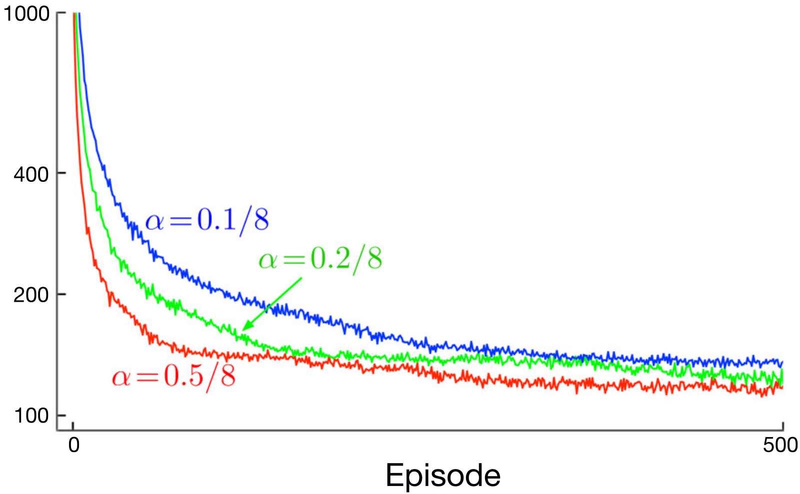
Learning Curves in Mountain Car

Mountain Car

Steps per episode

log scale

averaged over 100 runs



Convergence to Global Optimum in Episodic Control

Algorithm	Tabular	Linear	Non-linear
MC control	yes	chatter*	no
(semi-gradient) *n*-step Sarsa	yes	chatter*	no
(semi-gradient) *n*-step Q-learning	yes	no	no

*Chatters near optimal solution because optimal policy may not be representable under value function approximation

Deadly Triad

Risk of divergence arises when the following three are combined:

1. Function approximation
2. Bootstrapping
3. Off-policy learning

Possible fixes:

- Use importance sampling to warp off-policy distribution into on-policy distribution
- Use gradient TD methods which follow true gradient of projected Bellman error (see book, p. 266)

Reading

Required (RL book):

- Chapter 9 (9.1–9.5)
  (Box "Proof of Convergence of Linear TD(0)" in Sec 9.4 is not examined)
- Chapter 10 (10.1)
- Chapter 11 (11.1)

Optional:

- Remaining sections of chapters
- Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, 42(5):674–690
- Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. Machine Learning, 22(1):159–196