DR 101
Function Approximation

2019-11-16

김영삼

Resources

- Github
  - https://github.com/Youngsam/dr101/
- Book
  - Reinforcement Learning: An Introduction (Sutton, 2018)
  - Deep Reinforcement Learning Hands On (Lapan, 2018)

Content

- Introduction to Function Approximation
- Linear methods
- Non-linear methods
- The deadly triad

Why we need FA?

- Practical problems in RL often require large number of states or actions
  - Backgammon: 10^{20} states
  - Go (Baduk): 10^{170} states
  - Autonomous car driving: (infinite?)
- Dealing with large-scale action space is still on-going research topic
  - The set of all sentences is infinite

Problem of large-scale MDP

- In tabular methods, value function is represented as a lookup table
  - V(s) → value of state s
  - Q(s, a) → value of state s and action a
- Problem of large states or actions
  - Memory problem
  - Learning speed problem (too slow for one iteration)
  - Null experience problem (a state/action never occurred in previous learning)

Value Function Approximation

- Value function is represented as parameterized functional form with weight vector $w \in \mathbb{R}^d$

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

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

- The dimensionality of weights is much less than the number of states ($d < |S|$)
- A single update from a state generalizes to other states

Update target for state

- DP update
  - $s \longmapsto \mathbb{E}_{\pi}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) | S_t = s]$
- Monte-Carlo update
  - $s \longmapsto G_t$
- TD(0) update
  - $s \longmapsto R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)$
- N-step TD update
  - $s \longmapsto G_{t:t+n}$

Function Approximation for RL

- Not all FA methods are equally suited for RL
- Most supervised methods assume a static training set over which multiple passes are made
- But RL requires FA methods able to handle nonstationary target functions
- It often seeks to learn q_π while π changes

Prediction objective

- With function approximation, it is impossible to get the values of all states correctly
- In FA, making one state's estimate more accurate means making others' less accurate
- Mean squared value error (VE)

$$\mathrm{VE}(\mathbf{w}) = \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

$\mu(s)$ is a state distribution, $\mu(s) \geq 0$ and $\sum_s \mu(s) = 1$

On-policy distribution in episodic tasks

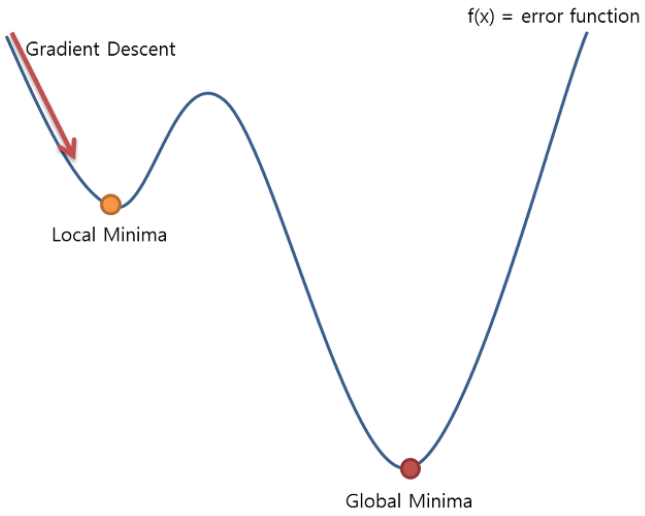
- In episodic cases, on-policy distribution depends on how the initial states of episodes are chosen
- Let $h(s)$ denote the probability an episode begins in state s
- Let $\eta(s)$ denote the average number of time steps spent in an episode

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_{a} \pi(a|\bar{s}) p(s|\bar{s}, a)$$

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$$

Note that the equation is undiscounted

VE and global optimum



- An ideal goal of VE is to find a global optimum
  - $\mathrm{VE}(\mathbf{w}^*) \leq \mathrm{VE}(\mathbf{w})$ for all possible $\mathbf{w}$
- But complex function approximators instead seek a local optimum
  - $\mathrm{VE}(\mathbf{w}^*) \leq \mathrm{VE}(\mathbf{w})$ in some neighborhood of $\mathbf{w}^*$
  - The goal is the best for most non-linear FA methods

Stochastic-gradient methods

- SGD methods are the most widely used of FA methods to online RL
- In SGD, weight vector is slightly adjusted in the direction that minimize the error function

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t),\end{aligned}$$

- The alpha is a step-size parameter and $\nabla f(w)$ is a vector of partial derivatives w.r.t the components of the vector

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top$$

Stochastic or semi-gradient methods

- If the target output, $v_\pi(S_t)$ is an unbiased estimate, $\mathbb{E}[v_\pi(S_t)|S_t = s] = v_\pi(S_t)$, then w_t will converge to a local optimum under SGD for decreasing α
- Thus, Monte-Carlo estimate is guaranteed to find a local optimum
- But bootstrapping methods (e.g., DP or TD) are not instances of true gradient descent
- Bootstrapping methods only include a part of the gradient, thus they are called ‘semi-gradient methods’

Linear methods

Feature vectors

- In order to represent state efficiently, we use feature vector for state
- Corresponding to every state, there is a real-valued vector $x(s)$:

$$x(s) = \left(x_1(s), x_2(s), \ldots, x_d(s)\right)^T$$

Example: 1000-state random walk

- All episodes begin near the state 500
- State aggregation method is used
  - $x_0(s) = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)^T$
- State transitions are from the current state to one of the 100 neighboring states to its left or right
- Termination on the left gives a reward of -1
- Termination on the right gives a reward of +1

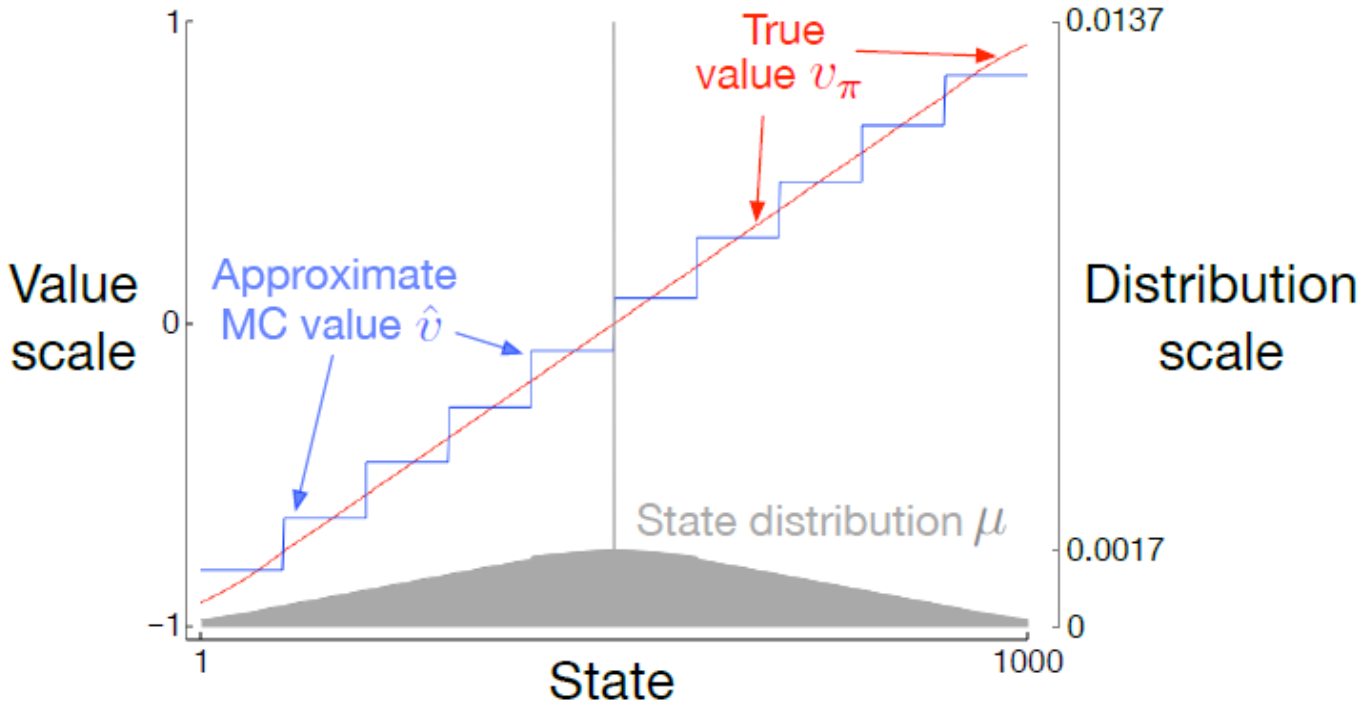


Figure 9.1: Function approximation by state aggregation on the 1000-state random walk task, using the gradient Monte Carlo algorithm (page 202).

Linear methods for FA

- Linear methods approximate state-value function by the inner product between $\mathbf{w}$ and x(s):

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

- The gradient of the approximate value function:

$$\nabla \hat{\mathrm{v}}(s, \mathbf{w}) = \mathbf{x}(s)$$

- Update rule:

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

Targets of various methods

- For MC:

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

- For TD(0):

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

- For TD(λ):

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

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

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

Feature construction for Linear methods

Polynomials

- Suppose each state *s* corresponds to *k* numbers, $s_1, s_2, \ldots, s_k$, with each $s_i \in R$
- For this *k*-dimensional state space, each order-n polynomial-basis feature x_i can be written as:

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}$$

where each $c_{i,j}$ is {0, 1, …, n} for n ≥ 0

Polynomials: example

- Let $\mathbf{x}(s) = (s_1, s_2)^T$
- If we want to use four-dimensional feature vectors, then it would like as below:

$$x(s) = (1, s_1, s_2, s_1 s_2)^T$$

- The initial 1 feature is added to represent affine functions in the original state numbers
- $s_1 s_2$ enables interactions of the features to be taken into account

Fourier Basis

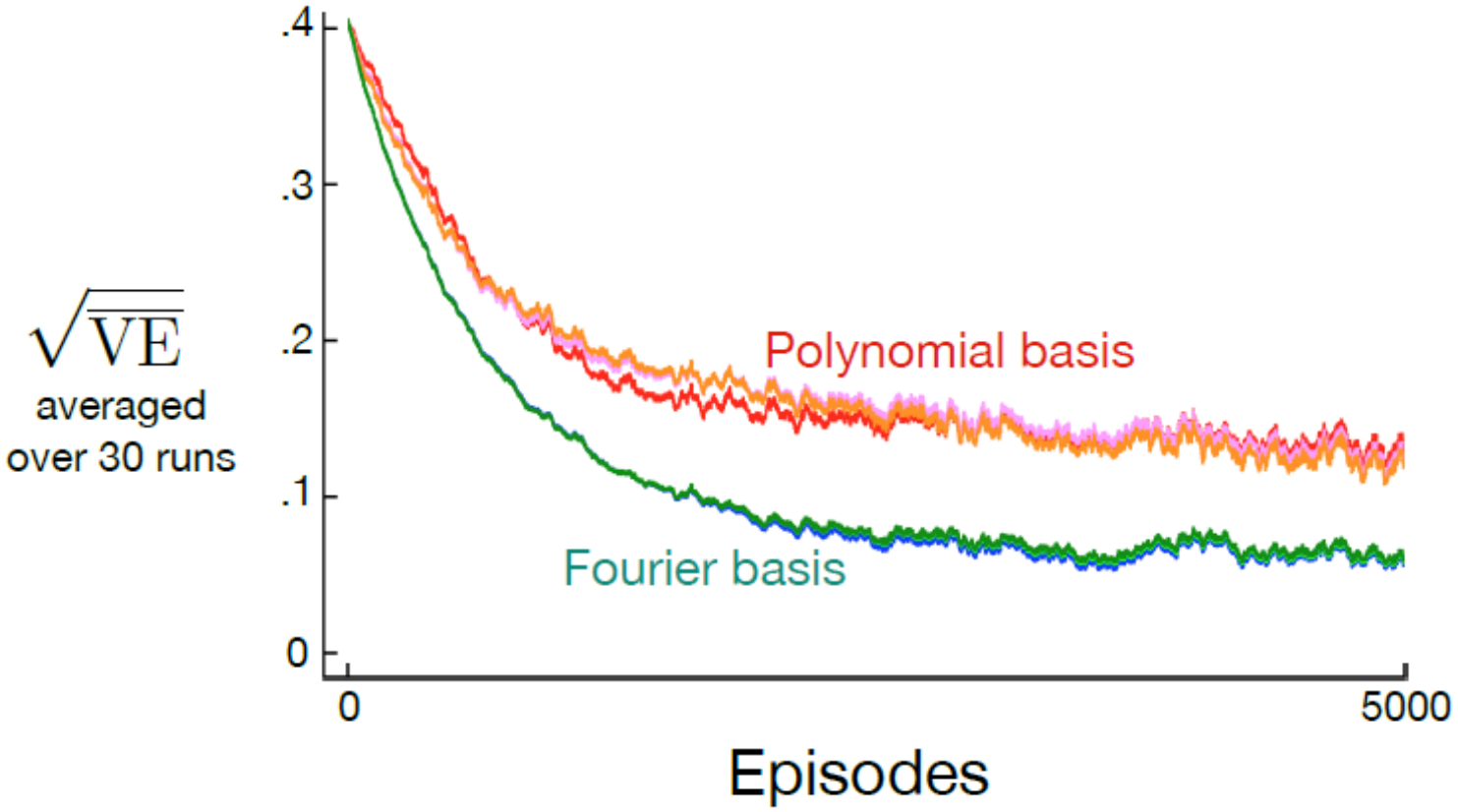
- This method expresses value function as weighted sums of sine and cosine basis features of different frequencies
- Each state $\mathbf{s} = (s_1, s_2, \ldots, s_k)^T$ with each $s_i \in [0, 1]$
- The *i*-th feature in the order-n Fourier cosine basis is written:

$$x_i(s) = \cos(\pi \mathbf{s}^T \mathbf{c}^i)$$

where $\mathbf{c}^i = \left(c_1^i, \ldots, c_k^i\right)^T$, with $\mathrm{c}_j^i \in \{0, \ldots, n\}$ for $j = 1, \ldots, k$ and $i = 0, \ldots, (n+1)^k$

Performance of Polynomials and Fourier basis

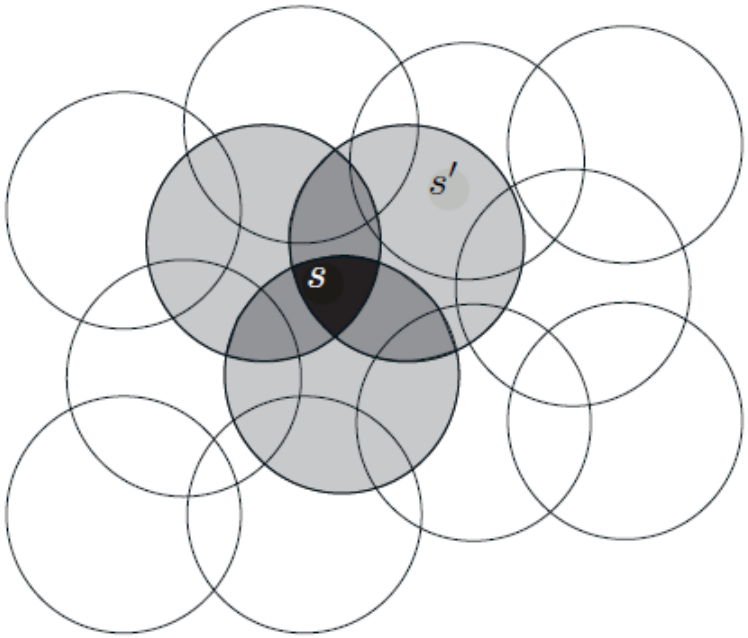
- Comparison results on the 1000-state random walk



Problems of Polynomials and Fourier basis

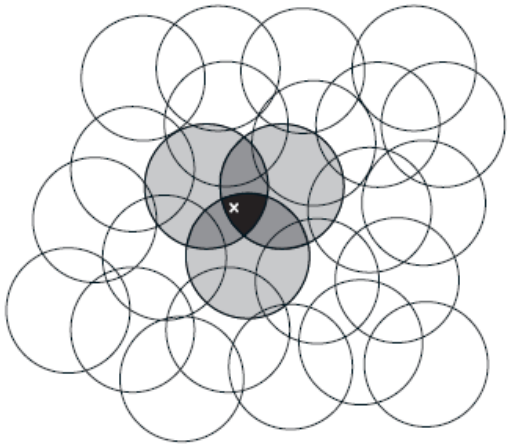
- The number of features in an order-n basis grows exponentially with the dimension k
- Thus, it is generally necessary to select a subset of them for function approximation
- This can be done using prior beliefs about the nature of the function

Coarse coding

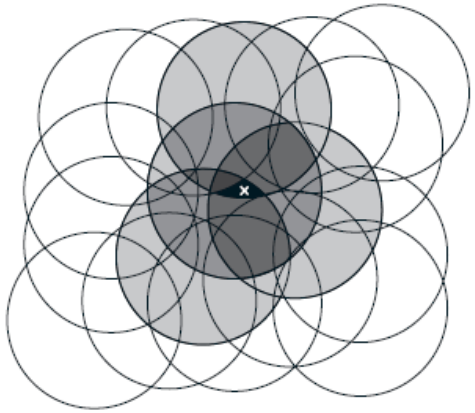


- This kind of representation is made up of features with circles in state space
- If the state is inside a circle, then the feature has the value 1 and is said to be present
- This is called coarse coding

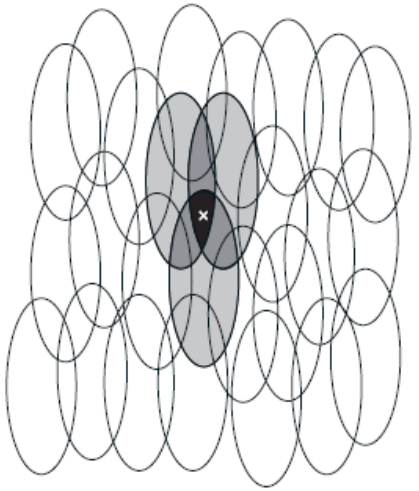
Types of coarse coding



Narrow generalization

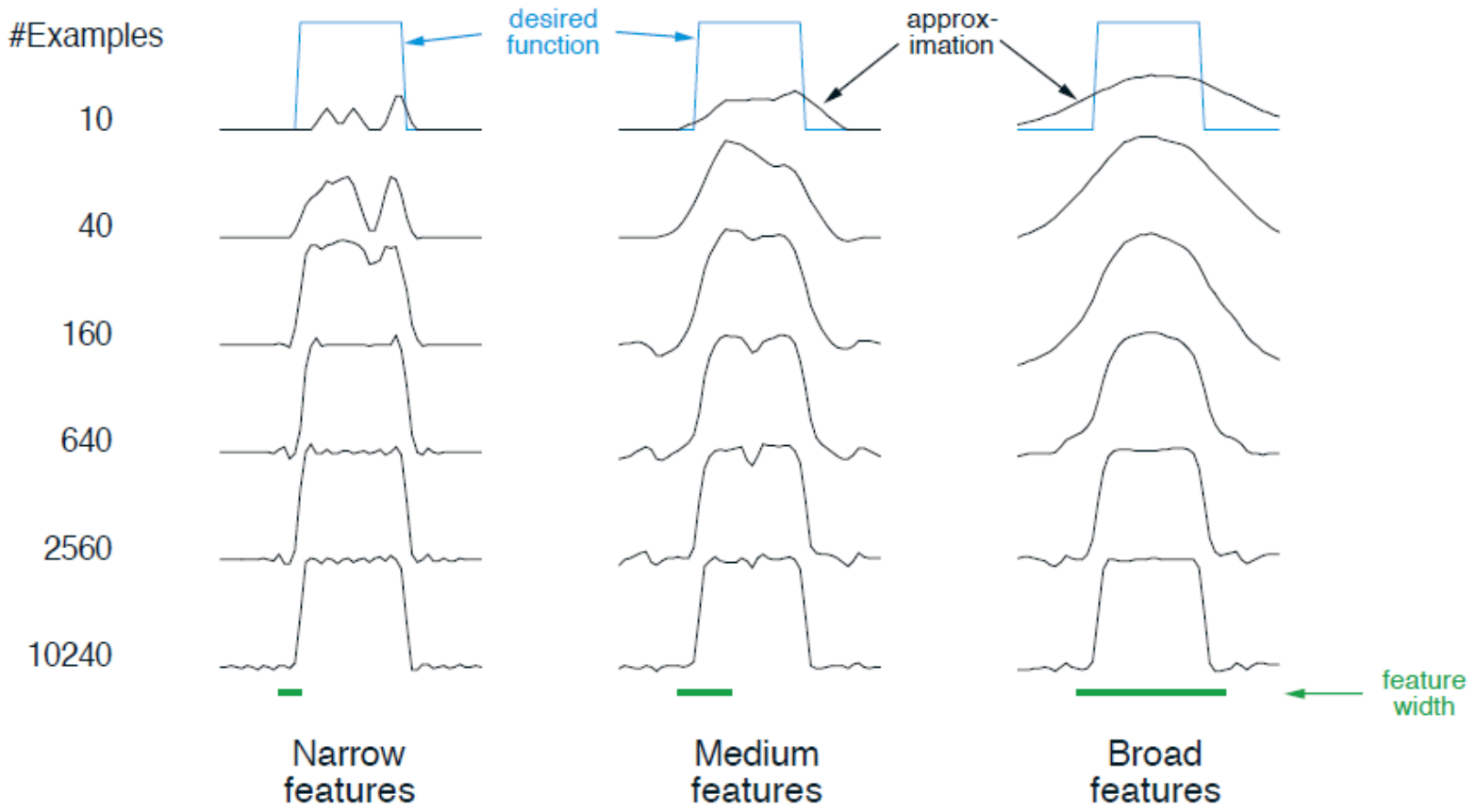


Broad generalization



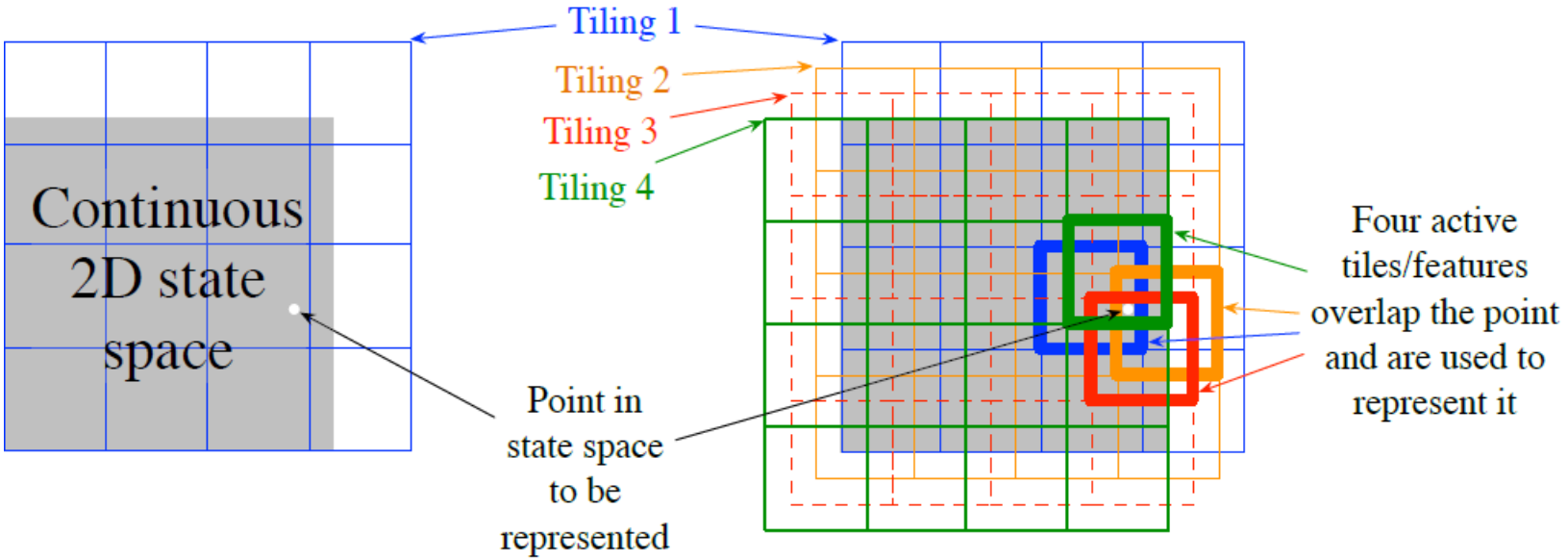
Asymmetric generalization

Density is what matters

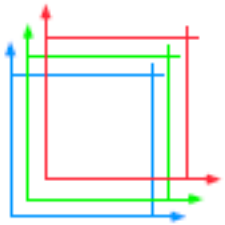


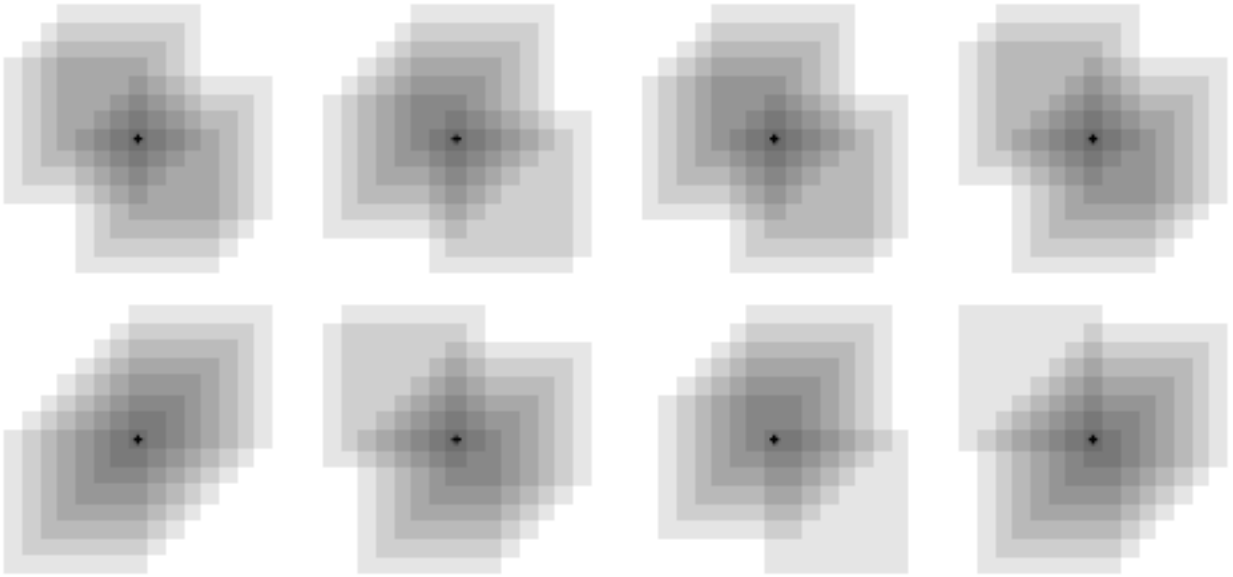
Tile Coding

- Tile coding is a form of coarse coding for multi-dimensional continuous spaces that is flexible and computationally efficient

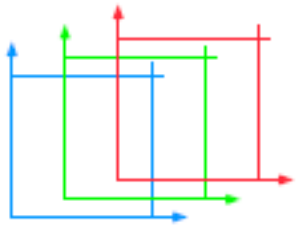


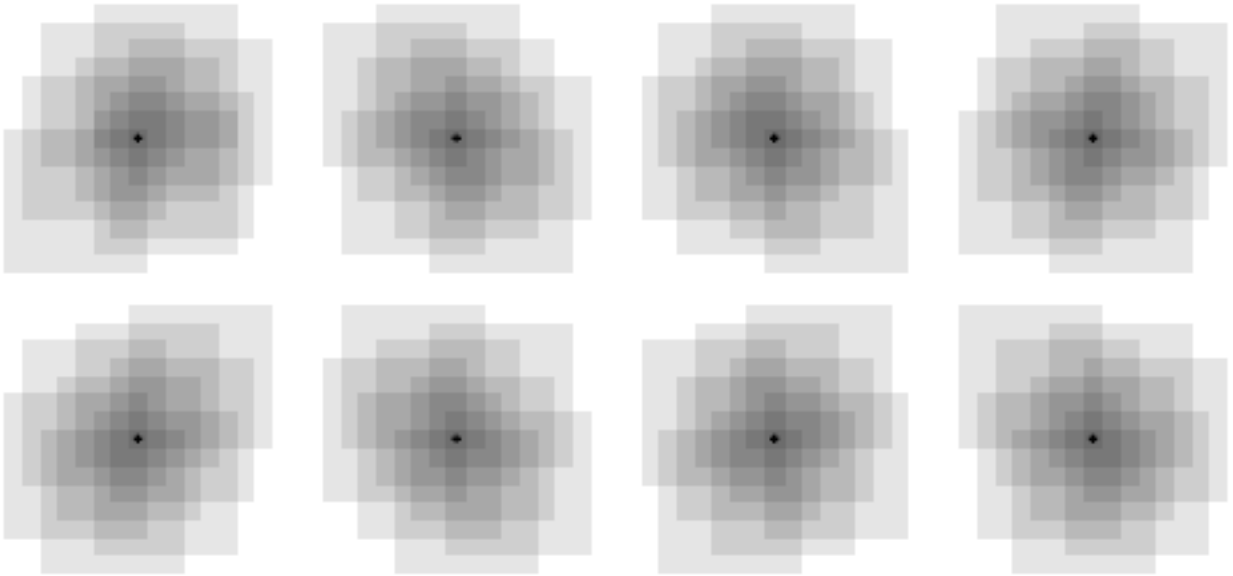
Possible generalizations for uniformly offset tilings



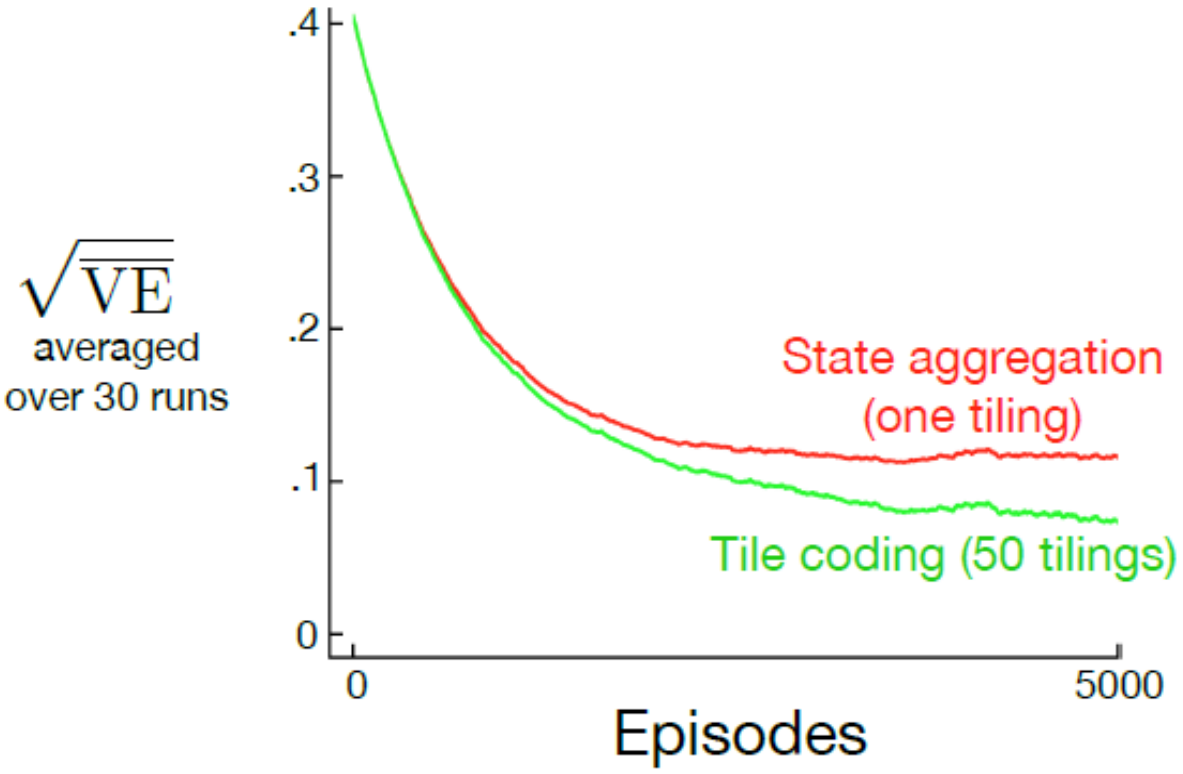


Possible generalizations for asymmetrically offset tilings





Tile coding vs State aggregation



- Coarse coding is always better than state aggregation method

Figure 8.8 Linear, gradient-descent Sarsa(λ) with binary features and ϵ-greedy policy.

Initialize $\vec{\theta}$ arbitrarily and $\vec{e} = \vec{0}$

Repeat (for each episode):

$s \leftarrow$ initial state of episode

For all $a \in \mathcal{A}(s)$:

$\mathcal{F}_a \leftarrow$ set of features present in s, a

$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$a \leftarrow \arg\max_a Q_a$

With probability ϵ: $a \leftarrow$ a random action $\in \mathcal{A}(s)$

Repeat (for each step of episode):

$\vec{e} \leftarrow \gamma \lambda \vec{e}$

For all $\bar{a} \neq a$: (optional block for replacing traces)

For all $i \in \mathcal{F}_{\bar{a}}$:

$e(i) \leftarrow 0$

For all $i \in \mathcal{F}_a$:

$e(i) \leftarrow e(i) + 1$ (accumulating traces)

or $e(i) \leftarrow 1$ (replacing traces)

Take action a, observe reward, r, and next state, s'

$\delta \leftarrow r - Q_a$

For all $a \in \mathcal{A}(s')$:

$\mathcal{F}_a \leftarrow$ set of features present in s', a

$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$a' \leftarrow \arg\max_a Q_a$

With probability ϵ: $a' \leftarrow$ a random action $\in \mathcal{A}(s)$

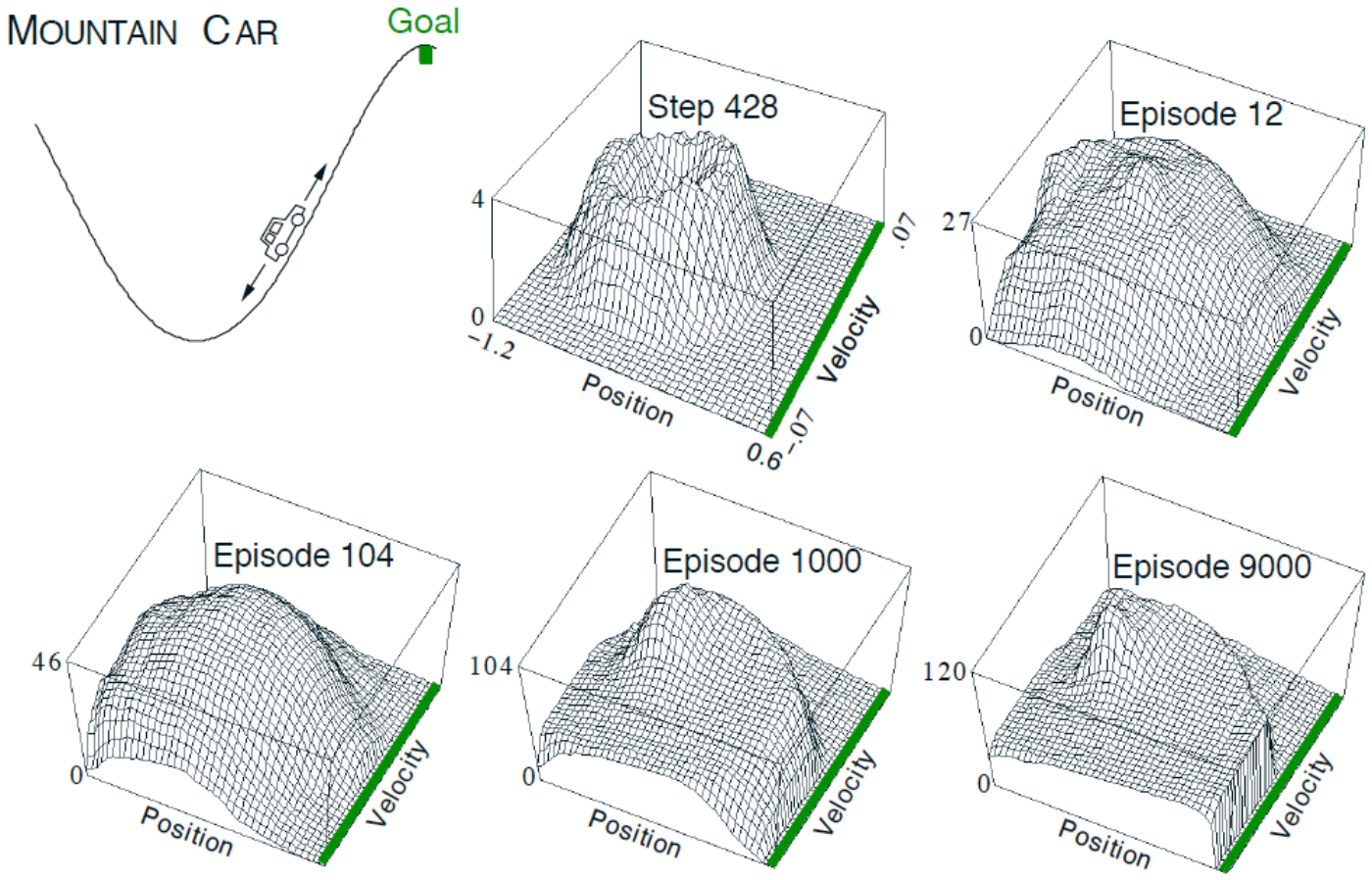
$\delta \leftarrow \delta + \gamma Q_{a'}$

$\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$

$a \leftarrow a'$

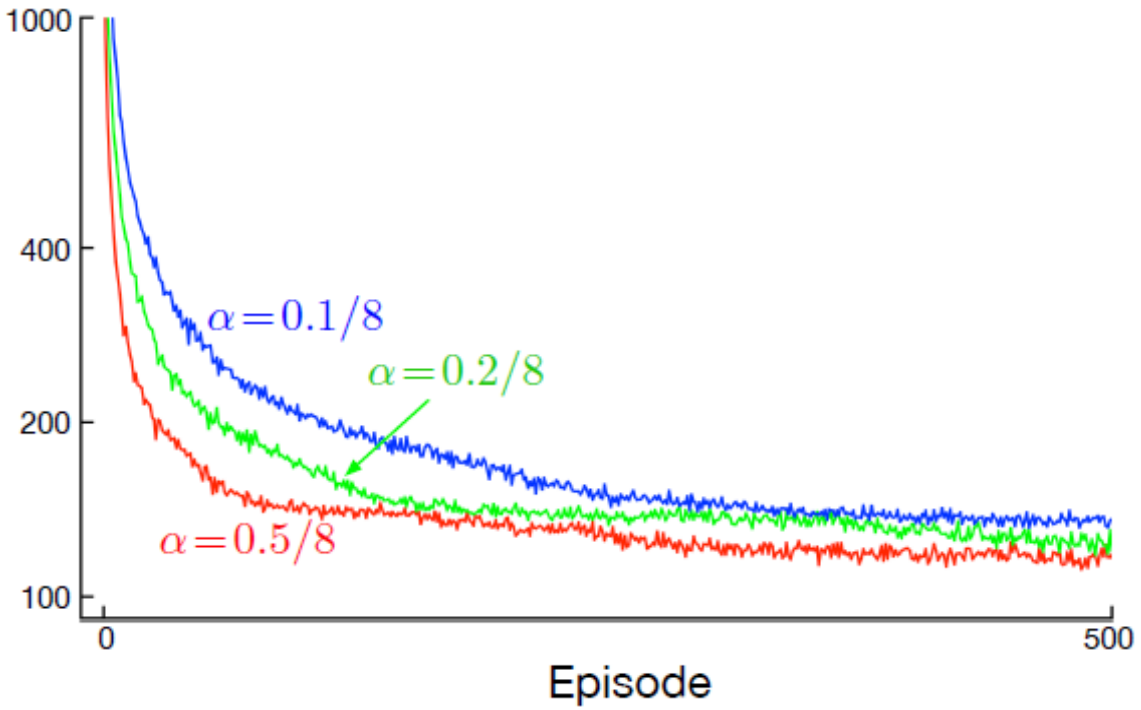
until s' is terminal

Linear Sarsa with tile-coding in [Mountain Car]



Learning curves of semi-gradient Sarsa

Mountain Car
Steps per episode
log scale
averaged over 100 runs



Watkin's Q(λ)

- Linear method
- Semi-gradient descent function approximation
- Binary features are used (e.g., Tile coding)
- Used epsilon-greedy policy for action selection

Initialize $\vec{\theta}$ arbitrarily and $\vec{e} = \vec{0}$

Repeat (for each episode):

  $s \leftarrow$ initial state of episode

  For all $a \in \mathcal{A}(s)$:

    $\mathcal{F}_a \leftarrow$ set of features present in s, a

    $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

  Repeat (for each step of episode):

    With probability $1 - \epsilon$:

      $a \leftarrow \arg\max_a Q_a$

      $\vec{e} \leftarrow \gamma \lambda \vec{e}$

    else

      $a \leftarrow$ a random action $\in \mathcal{A}(s)$

      $\vec{e} \leftarrow 0$

    For all $i \in \mathcal{F}_a$: $e(i) \leftarrow e(i) + 1$

    Take action a, observe reward, r, and next state, s'

    $\delta \leftarrow r - Q_a$

    For all $a \in \mathcal{A}(s')$:

      $\mathcal{F}_a \leftarrow$ set of features present in s', a

      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

    $a' \leftarrow \arg\max_a Q_a$

    $\delta \leftarrow \delta + \gamma Q_{a'}$

    $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$

  until s' is terminal

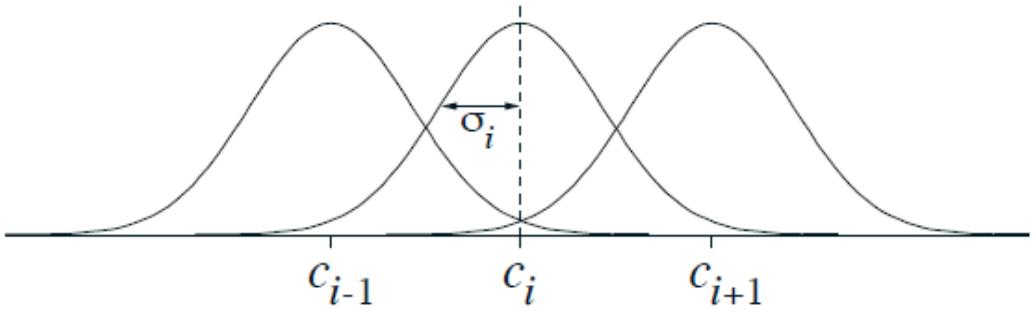
Figure 8.9 A linear, gradient-descent version of Watkins's Q(λ) with binary features, ϵ-greedy policy, and accumulating traces.

Radial Basis Functions

- RDFs are the natural generalization of coarse coding to continuous-valued features (example code)
- A typical RBF feature is defined as below:

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

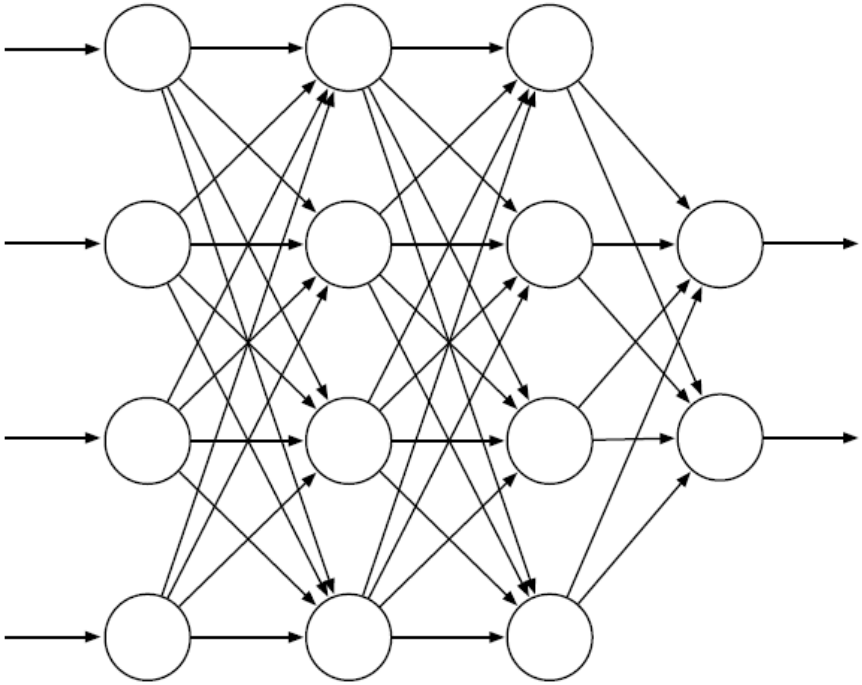
- The distance metric can be chosen appropriately



Non-linear methods

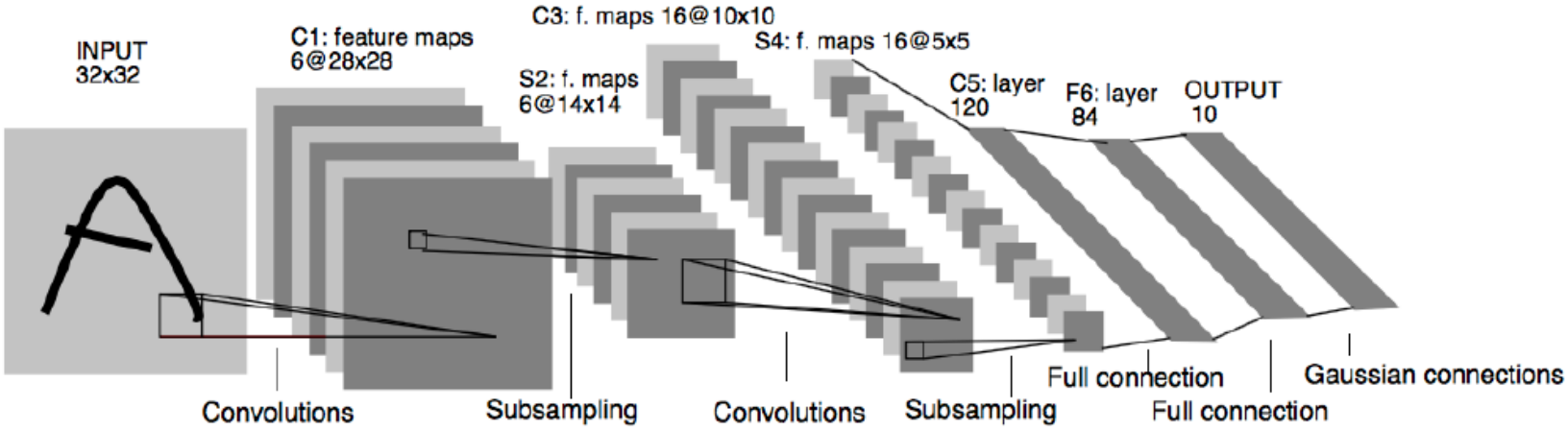
Artificial Neural Networks

- ANNs are widely used for nonlinear function approximation



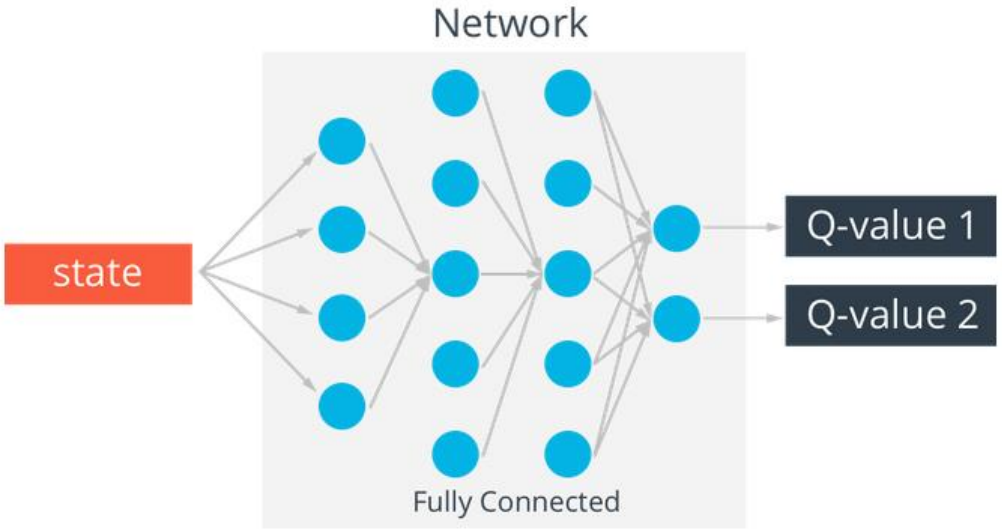
Deep convolution network

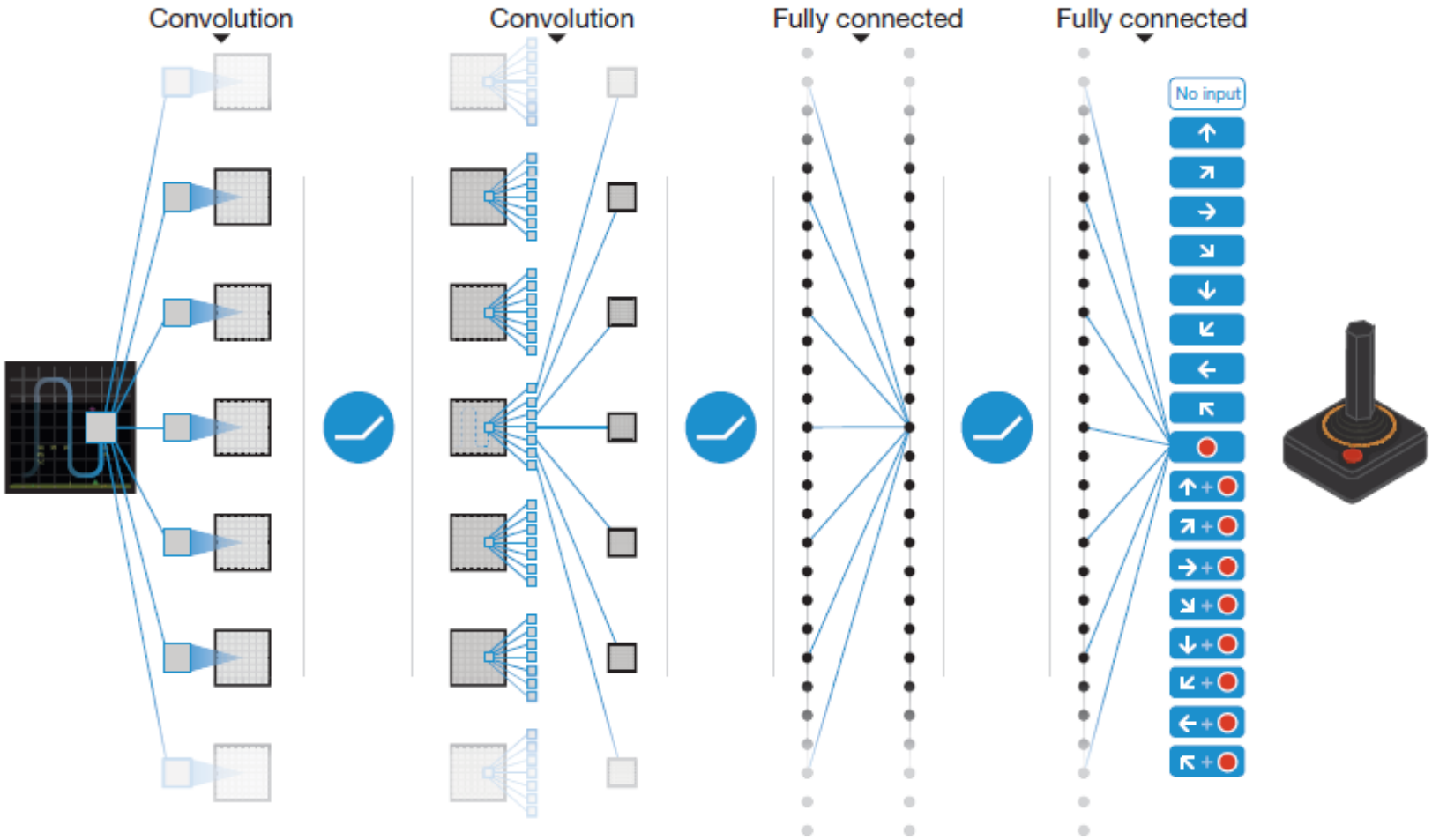
- Deep CNN has been successful in RL applications
- This network is specialized for processing high-dimensional data such as images

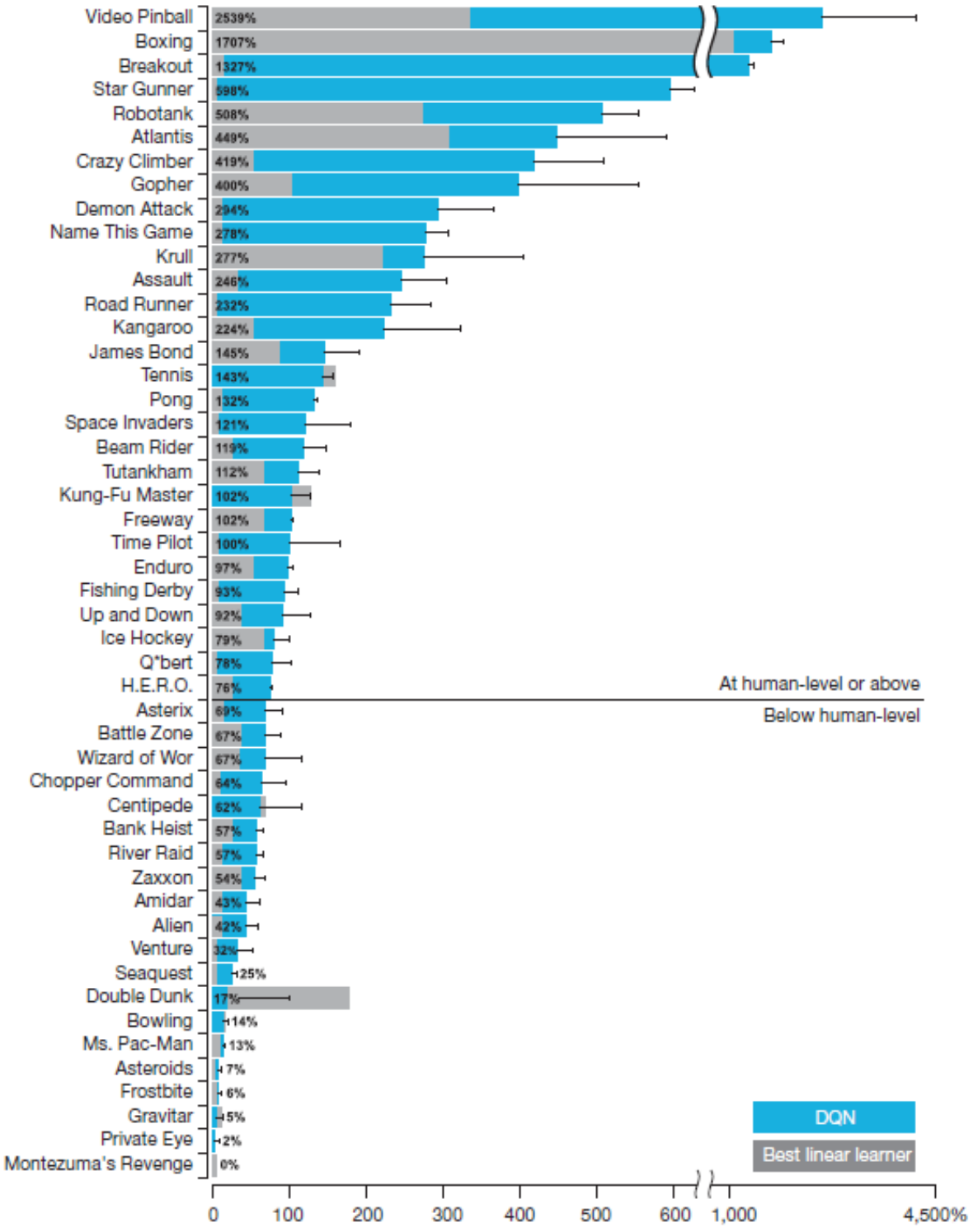


DQN for Atari games

- DQN (Mnih et al, 2015) learns Q(s, a) from streams of images
- Input state s is raw pixels from last 4 frames
- Output is 18 joystick/button positions
- Reward is change in score for the step




Convolution
Convolution
Fully connected
Fully connected
No input


Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
119%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

DQN with experience replay

- Experience replay mechanism is used to remove correlations in the observation data by random sampling
- It first sample <s, a, r, s'> sequences from experience buffer
- Then apply SGD update to the mini batch
  - Compute Q-learning targets w.r.t old weights (θ^-)
  - Optimize MSE between Q-network and Q-learning targets

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

DQN with target network

- Target network is used with parameters θ^-
- Target network is the same as the online network except that its parameters are copied every τ steps from the online network
- The target used by DQN is then

$$Y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

- The gradient of the loss is then

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a,r,s'}[(Y - Q(s, a; \theta)\nabla_\theta Q(s, a; \theta)]$$

Effects of replay and separating target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

The Deadly Triad

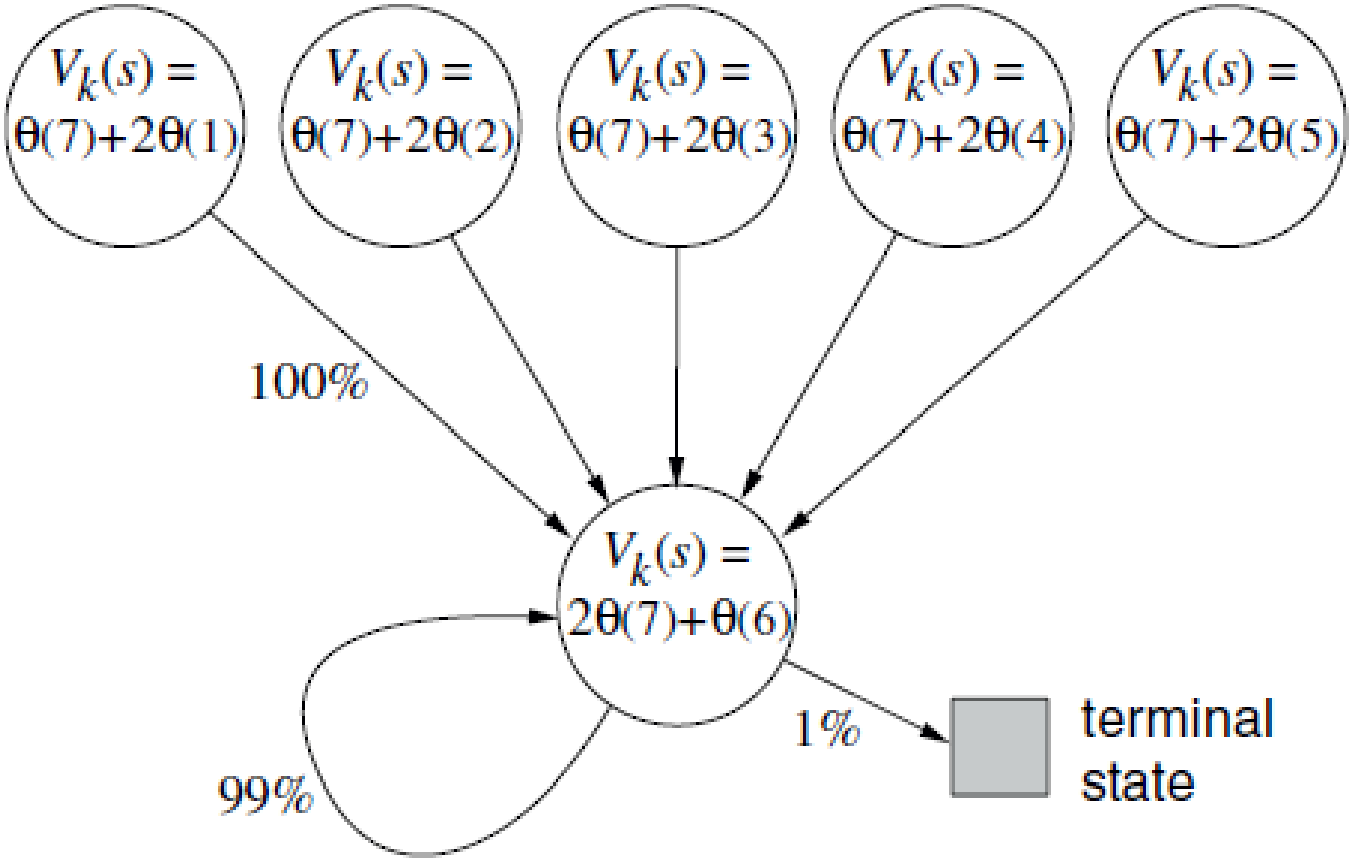
Challenges of off-policy learning

- Off-policy learning seeks to learn a value function for target policy, given data due to a behavior policy
- In prediction, both polices are static and given
- In control, action values are learned, and both policies typically change during learning
- Two challenges
  - How to do with the target of the update → importance sampling
  - How to do with the distribution of the updates → develop true gradient methods

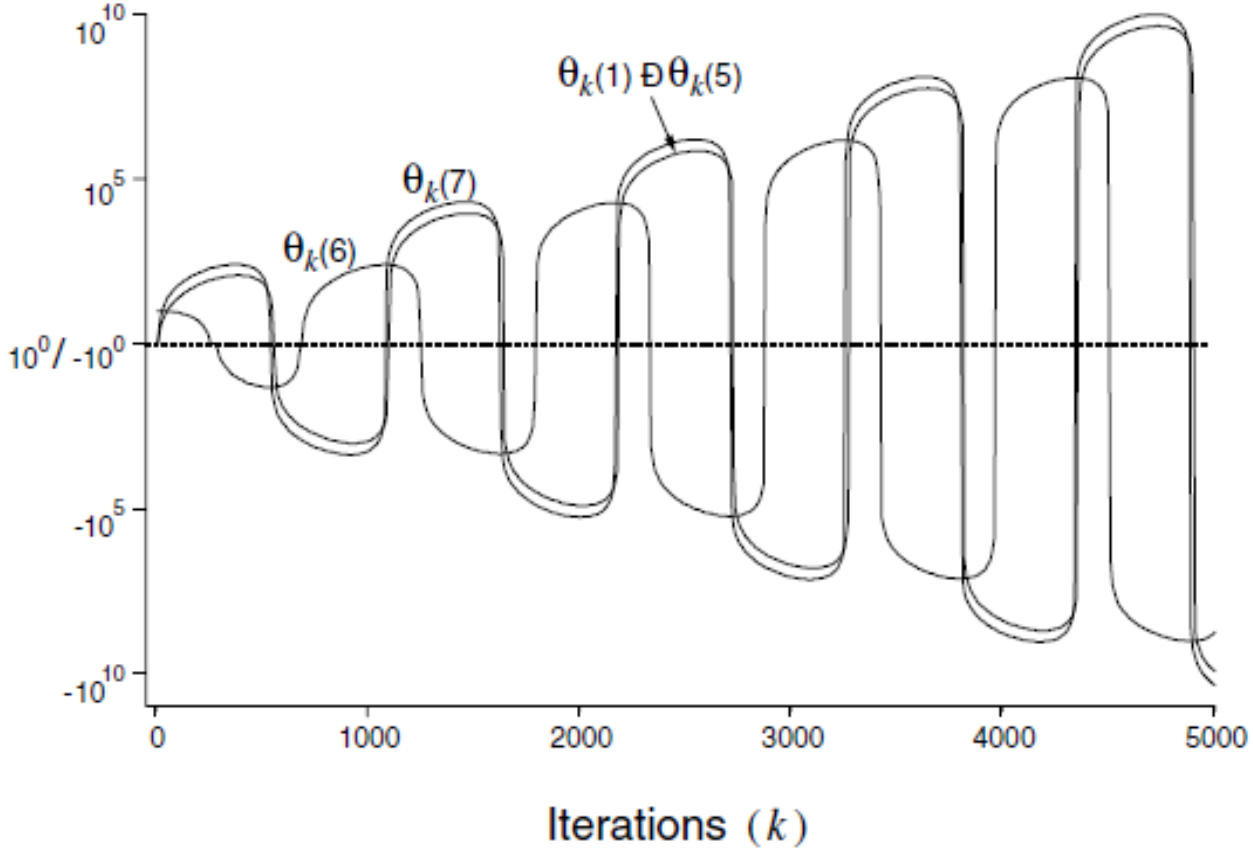
Off-policy divergence

- In off-policy learning, semi-gradient and other algorithms can be unstable and diverge
- Two famous examples on this problem is Baird's counter-example and Tsitsiklis and Van Roy's counter-example

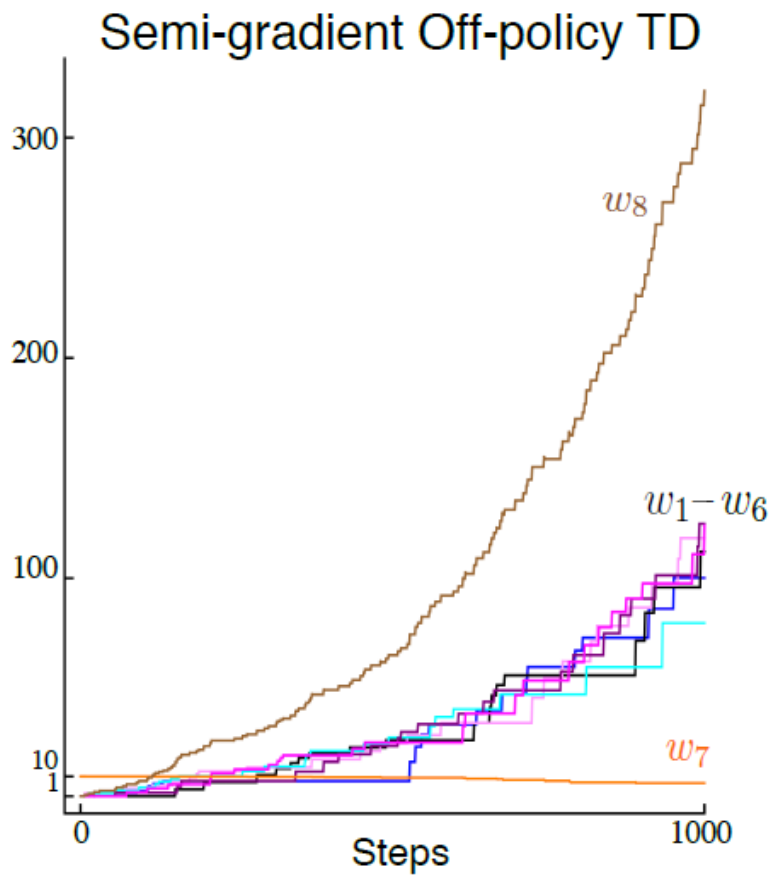
Baird's Counterexample

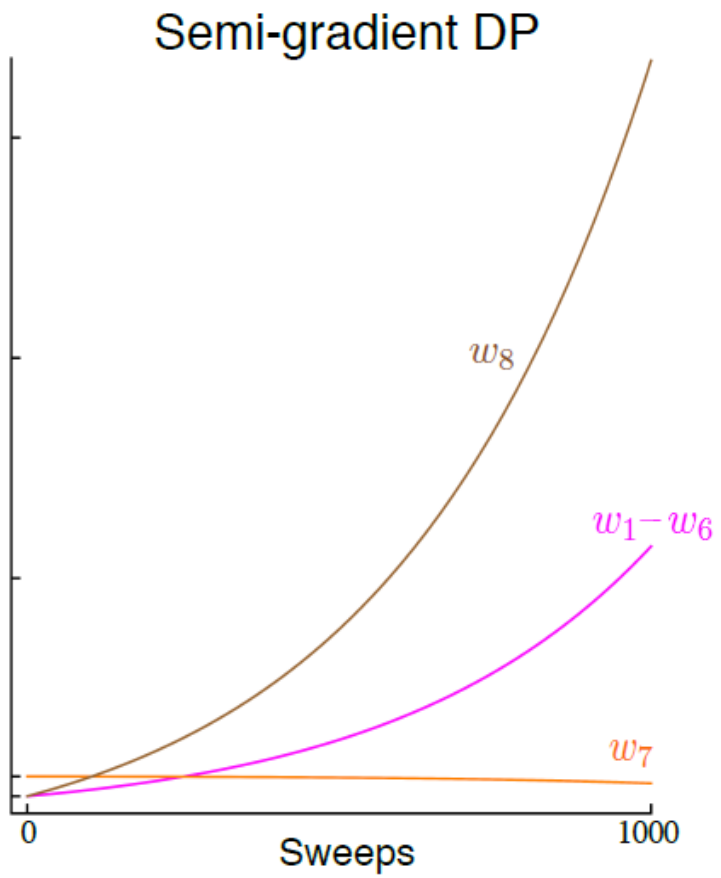


Parameter values, $\theta_k(i)$ (log scale, broken at ± 1)

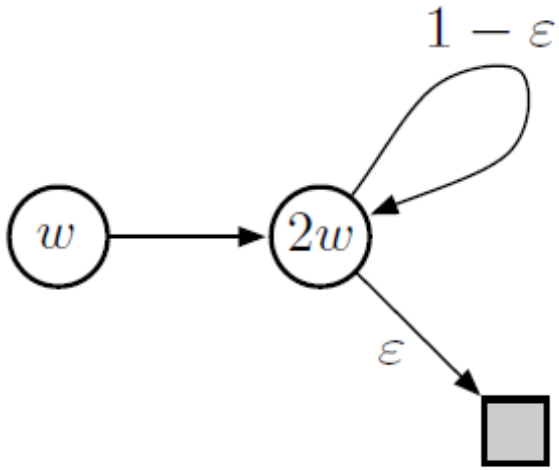


Instability of Baird's counterexample





Tsitsiklis and Van Roy's Counterexample



$$
\begin{aligned}
w_{k+1} &= \operatorname*{arg\,min}_{w\in\mathbb{R}} \sum_{s\in\mathcal{S}} \Big(\hat{v}(s,w) - \mathbb{E}_\pi\big[R_{t+1} + \gamma\hat{v}(S_{t+1},w_k) \;\big|\; S_t = s\big]\Big)^2 \\
&= \operatorname*{arg\,min}_{w\in\mathbb{R}} \big(w - \gamma 2w_k\big)^2 + \big(2w - (1-\varepsilon)\gamma 2w_k\big)^2 \\
&= \frac{6-4\varepsilon}{5}\gamma w_k.
\end{aligned}
$$

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

The Deadly Triad

- The danger of instability and divergence arises whenever we combine all of the following three elements
  - Function approximation
  - Bootstrapping
  - Off-policy training

Convergence of control algorithms

Algorithm	Tabular	Linear	Non-linear
Monte-Carlo	O	Δ	X
Sarsa	O	Δ	X
Q-learning	O	X	X

How to deal with the Triad

- First of all, function approximation cannot be given up
- Doing without bootstrapping is possible at the cost of computational and data efficiency
- Do we give up off-policy learning?
  - On-policy methods are often adequate
  - But off-policy learning is **essential** to the goal of creating a powerful intelligent agent
- van Hasselt (2018) argues that modern DQN models are safe from the problems caused by Triad