# Value Function Approximation

## CMPS 4660/6660: Reinforcement Learning

Acknowledgement: slides adapted from David Silver's RL course

# Agenda

- Introduction
- Incremental Methods
- Batch Methods

# 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?

# Value Function Approximation

- So far we have represented value function by a lookup table (tabular setting)
  - Every state $s$ has an entry $V(s)$
  - Or every state-action pair $s, a$ has an entry $Q(s, a)$
- Problems 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

# Value Function Approximation

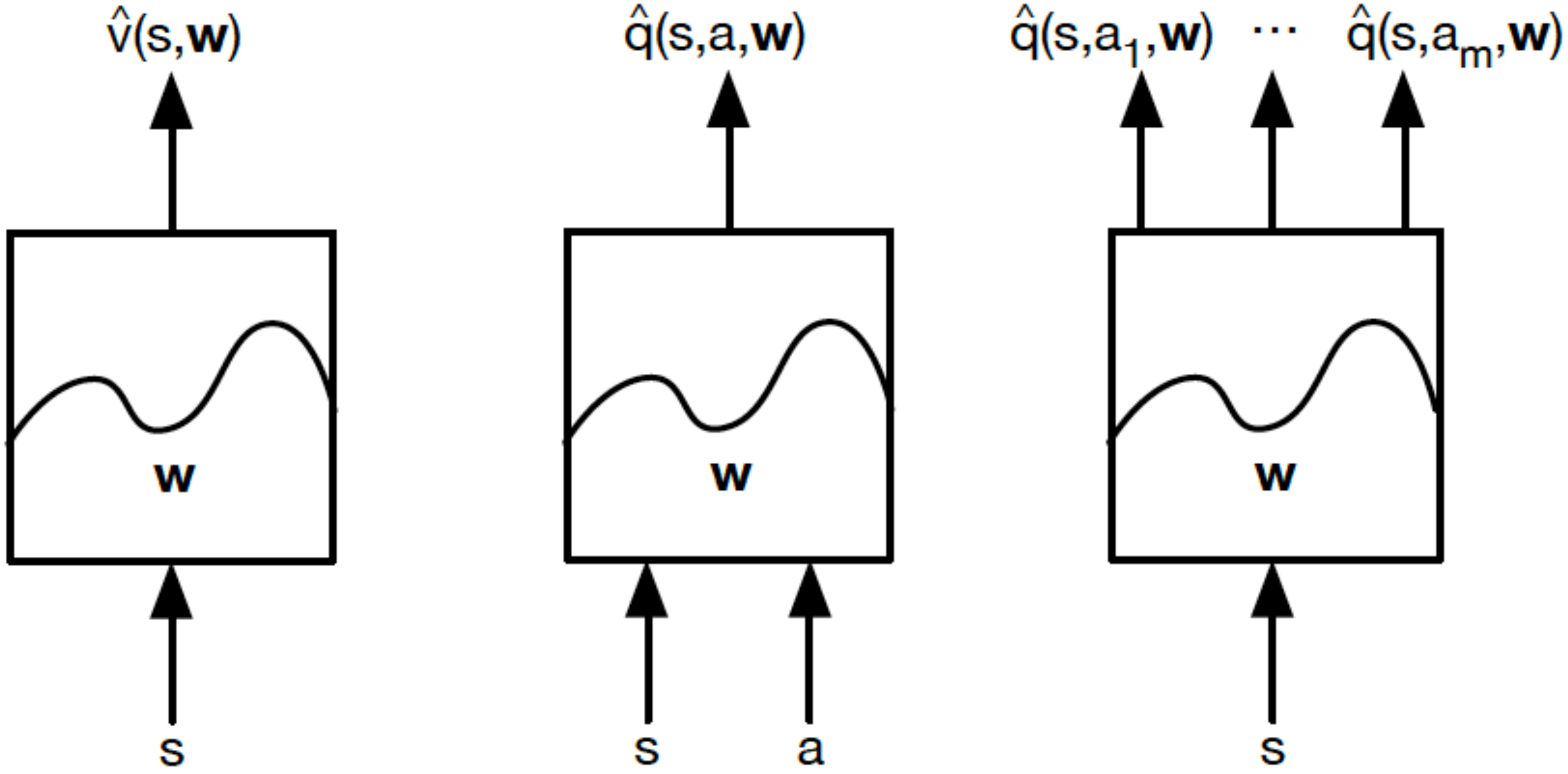
- Solution for large MDPs:
  - Estimate value function with *function approximation*

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

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

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

# Types of Value Function Approximation

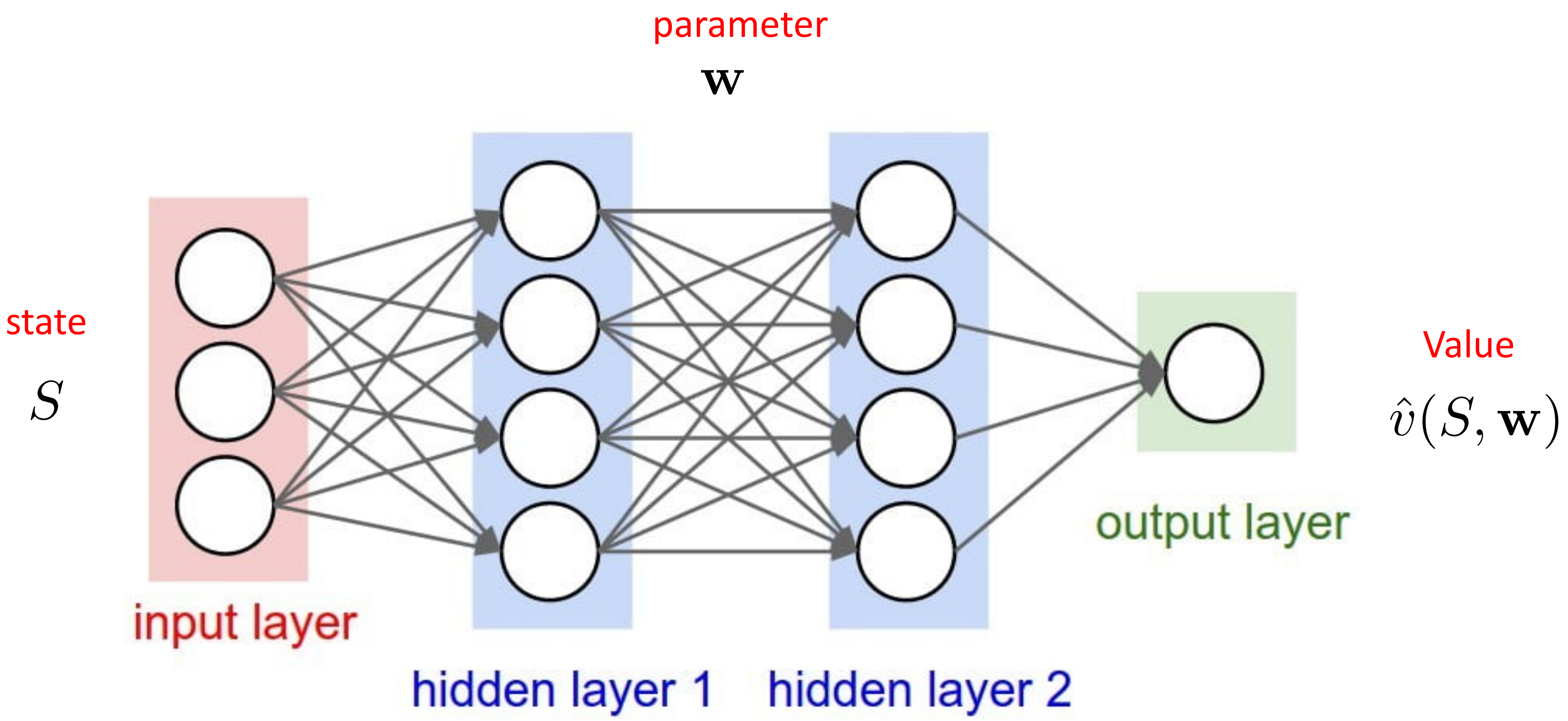


# Which Function Approximator?

- There are many function approximators, e.g.
    - Linear combinations of features
    - Neural network
    - Decision tree
    - Nearest neighbor
    - Fourier / wavelet bases
    - ...

# Nonlinear Value Function Approximation

- Use artificial neural networks



# Which Function Approximator?

- We consider differentiable function approximators, e.g.
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbor
  - Fourier / wavelet bases
  - ...
- Furthermore, we require a training method that is suitable for non-stationary, non-iid data

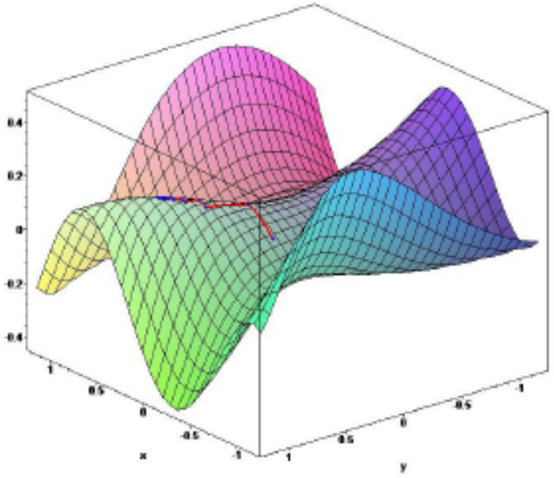
# Agenda

- Introduction
- Incremental Methods
  - On-policy prediction with value function approximation
  - On-policy control with value function approximation
  - Off-policy methods with approximation
- Batch Methods

# Gradient Descent

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

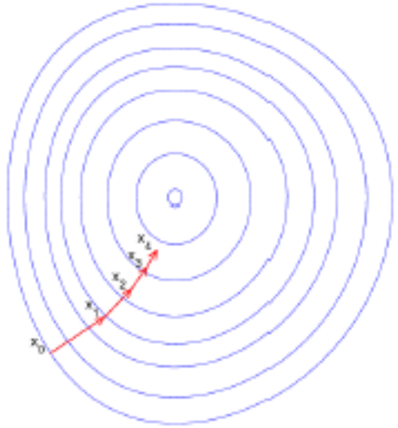
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$



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

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

where $\alpha$ is a step-size parameter



# Gradient Descent

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

$$\Leftrightarrow \mathbf{w}_{t+1} = \operatorname{argmin}_{\mathbf{w}} \frac{1}{2}\left\|\mathbf{w} - \left(\mathbf{w}_t - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}_t)\right)\right\|_2^2$$

$$= \operatorname{argmin}_{\mathbf{w}} \alpha \langle \mathbf{w} - \mathbf{w}_t, \nabla_{\mathbf{w}} J(\mathbf{w}_t), \rangle + \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|_2^2$$

$$= \boxed{\operatorname{argmin}_{\mathbf{w}} \alpha [J(\mathbf{w}_t) + \langle \mathbf{w} - \mathbf{w}_t, \nabla_{\mathbf{w}} J(\mathbf{w}_t) \rangle] + \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|_2^2}$$

$$= \operatorname{argmin}_{\mathbf{w}} \alpha \langle \mathbf{w}, \nabla_{\mathbf{w}} J(\mathbf{w}_t), \rangle + \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|_2^2$$

# 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 $v_\pi(s)$

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

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

- $\mu(s) \geq 0, \sum_{s \in \mathcal{S}} \mu(s) = 1$
- Often $\mu(s)$ is chosen to be the fraction of time spent in $s$, i.e., $\mu$ is the stationary distribution of states under policy $\pi$

# Stationary distribution of states under policy $\pi$

- Recall that given an MDP $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ and a stationary policy, the state sequence $S_0, S_1, \ldots$ is a Markov chain $\langle \mathcal{S}, P^{\pi} \rangle$, where

$$P^{\pi}_{ss'} = \sum_{a \in \mathcal{A}(s)} \pi(a|s)\, P_{ss'}(a)$$

- Given a Markov chain $\langle \mathcal{S}, P \rangle$, a probability distribution $\{\mu(s): s \in \mathcal{S}\}$ is called
  - a limiting distribution if $\mu_{s'} = \lim_{n \to \infty} P^{(n)}_{ss'}, \ \forall s, s'$
  - a stationary distribution if $\mu \cdot P = \mu$
- Theorem: If a finite state Markov chain is irreducible and aperiodic, it must have a limiting distribution, which is also stationary and is unique (such a chain is called ergodic).

# Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector **w** minimizing

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

- Gradient descent finds a local minimum

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

- Stochastic gradient descent samples 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 $n$ dimensional *feature vector*

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

- Typically, $d \leq |\mathcal{S}|$
- 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

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

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

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

- 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

# Linear Value Function Approximation

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

$$\mathrm{X} = \begin{pmatrix} - & \mathrm{x}(s_1) & - \\ - & \mathrm{x}(s_2) & - \\ & \vdots & \end{pmatrix} = \begin{pmatrix} \mathrm{x}_1(s_1) & \mathrm{x}_2(s_1) \ldots & \mathrm{x}_d\,(s_1) \\ \mathrm{x}_1(s_2) & \mathrm{x}_2(s_2) \ldots & \mathrm{x}_d\,(s_2) \\ & \vdots & \end{pmatrix} = \begin{pmatrix} | & & | \\ \mathrm{x}_1 & \ldots & \mathrm{x}_d \\ | & & | \end{pmatrix}$$

$$\hat{v}(\mathbf{w}) = \begin{pmatrix} \hat{v}(s_1, \mathbf{w}) \\ \hat{v}(s_2, \mathbf{w}) \\ \vdots \end{pmatrix} = \mathrm{X}\mathbf{w}$$

# Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using table lookup features, where $d = |\mathcal{S}|$

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

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

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_d) \end{pmatrix}^{\mathsf{T}} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_d \end{pmatrix}$$

# Incremental Prediction Algorithms

- 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 $G_t$

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

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

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

  - For TD($\lambda$), the target is the return $G_t^\lambda$ $\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$

# Monte-Carlo with Value Function Approximation

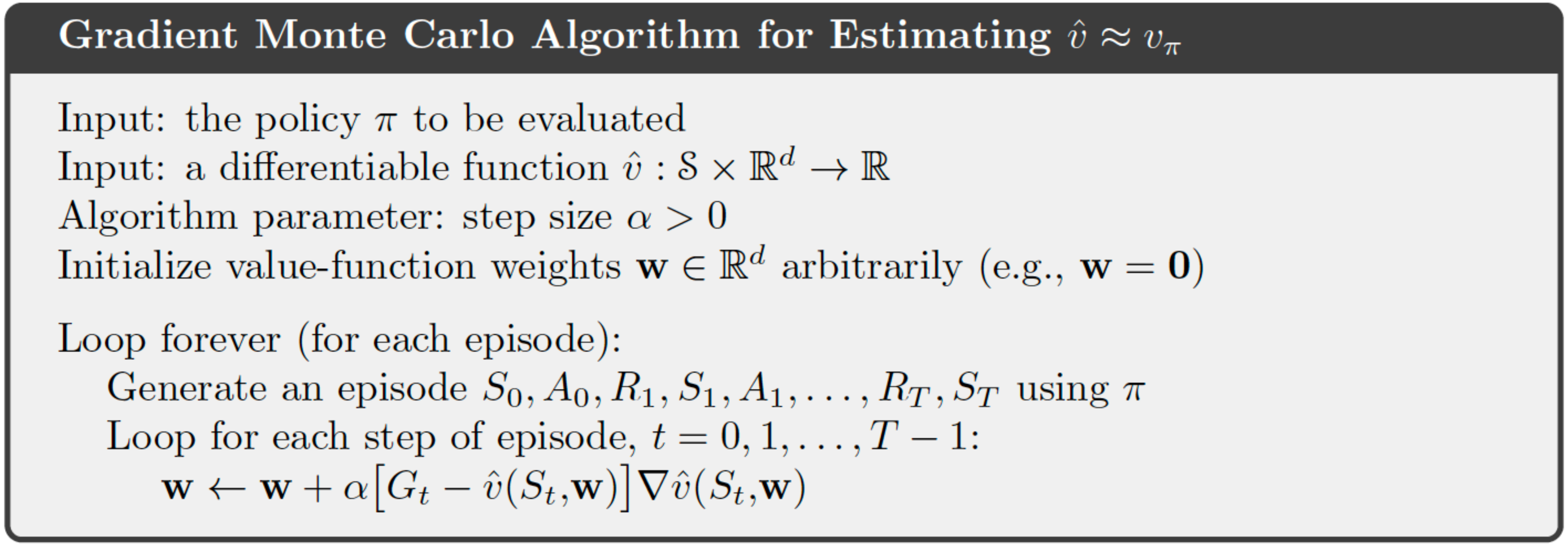
- Return $G_t$ is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to “training data”:

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

- For example, using linear Monte-Carlo policy evaluation

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

- Monte-Carlo evaluation converges to (using a decreasing $\alpha$)
  - a global optimum using linear function approximation
  - a local optimum when using non-linear value function approximation



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

```
Input: the policy $\pi$ 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 $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \dots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$
```

# 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”:

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

- For example, using linear TD(0)

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

- A semi-gradient method: $\mathbf{w}$ in $\hat{v}(S', \mathbf{w})$ is ignored when taking the gradient

# TD Learning with Value Function Approximation

- TD(0) converges to $\mathbf{w}_{\text{TD}}$ such that $J(\mathbf{w}_{\text{TD}}) \leq \frac{1}{(1-\gamma)^2} \min_{\mathbf{w}} J(\mathbf{w})$ (w.p.1) when
  - linear function approximation is adopted
  - the Markov chain $\langle \mathcal{S}, P^{\pi} \rangle$ is ergodic
  - step size follows the Robbins-Monro's conditions

See Tsitsiklis and van Roy, "An Analysis of Temporal-Difference Learning with Function Approximation", 1997

# Convergence of Linear TD(0)

- Update at time $t$:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\Big)\mathbf{x}_t \quad \text{where} \quad \mathbf{x}_t = \mathbf{x}(S_t)$$

$$= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\big)^\top \mathbf{w}_t\Big),$$

- Once the system reaches steady state at time $t$:

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\Big[\mathbf{x}_t\big(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\big)^\top\Big] \in \mathbb{R}^d \times \mathbb{R}^d$$

- If the system converges, it must converge to $\mathbf{w}_{\mathrm{TD}}$ where

$$\begin{aligned} & \mathbf{b} - \mathbf{A}\mathbf{w}_{\mathrm{TD}} = \mathbf{0} \\ \Rightarrow \quad & \mathbf{b} = \mathbf{A}\mathbf{w}_{\mathrm{TD}} \\ \Rightarrow \quad & \mathbf{w}_{\mathrm{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}. \end{aligned}$$

TD fixed point

# Convergence of Linear TD(0)

- Once the system reaches steady state at time $t$:

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$$

$$\Leftrightarrow \mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = (\mathbf{I} - \alpha\mathbf{A})\mathbf{w}_t + \alpha\mathbf{b}$$

$\Rightarrow \mathbf{w}_t$ converges if $\mathbf{A}$ is positive definite and $\alpha$ is small enough

# TD Fixed Point

- Recall

$$\mathrm{X} = \begin{pmatrix} - & \mathrm{x}(s_1) & - \\ - & \mathrm{x}(s_2) & - \\ & \vdots & \end{pmatrix} = \begin{pmatrix} | & & | \\ \mathrm{x}_1 & \cdots & \mathrm{x}_d \\ | & & | \end{pmatrix}$$

- Assume that $\mathrm{x}_1, \mathrm{x}_1, \dots, \mathrm{x}_d$ are linearly independent
- Define $\Pi$ as the projection of $v$ on $\{\mathrm{X}\mathbf{w} : \mathbf{w} \in \mathbb{R}^{\boldsymbol{d}}\}$

$$\Pi v = \operatorname{argmin}_{\bar{v} \in \{\mathrm{X}\mathbf{w} : \mathbf{w} \in \mathbb{R}^{\boldsymbol{d}}\}} \sum_{s \in \mathcal{S}} \mu(s) \left[v(s) - \bar{v}(s)\right]^2$$

- Then $v_0 \doteq \mathrm{X}\mathbf{w}_{\mathbf{TD}}$ is a fixed point of $\underline{\Pi T^{\pi}(\cdot)}$, i.e., $\Pi T^{\pi}(v_0) = v_0$

Projected Bellman Operator

# Least-Squares TD (LSTD)

- Recall TD fixed point: $\mathbf{w}_{\text{TD}} = \mathbf{A}^{-1}\mathbf{b},$

$$\mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\right] \quad \text{and} \quad \mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] .$$

- Least-squares TD computing estimates of $\mathrm{A}$ and $\mathrm{b}$, and then directly computing the TD fixed point

$$\widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^\top + \varepsilon\mathbf{I} \quad \text{and} \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k, \quad \mathbf{w}_t \doteq \widehat{\mathbf{A}}_t^{-1}\widehat{\mathbf{b}}_t.$$

- Most data efficient form of linear TD(0)
- $O(d^2)$ computational and memory complexity per time step

# TD($\lambda$) with Value Function Approximation

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

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

- Forward view linear TD($\lambda$)

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

- Backward view linear TD($\lambda$)

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

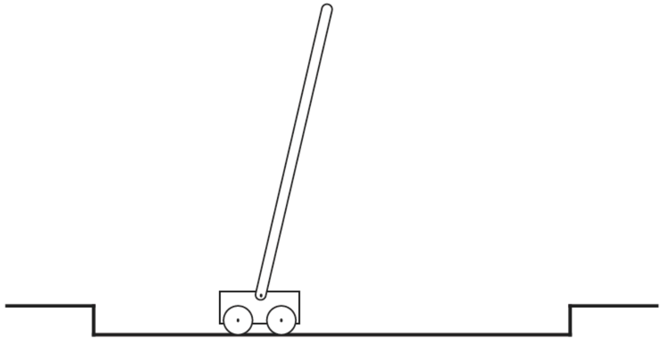
- Forward view and backward view linear TD($\lambda$) (with offline updates) are equivalent

# How to Design Features?

- Use prior domain knowledge to design features for RL
  - Graphic objects: shape, color, size
  - Mobile Robot: location, battery level, sensing reading
- Linear value function approximation

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

  - i.e., pole-balancing:
  - High angular velocity can be good or bad depending on angle
  - Linear value function could not represent this if angle and angular velocity are two different features
  - One should design features to combine state dimensions

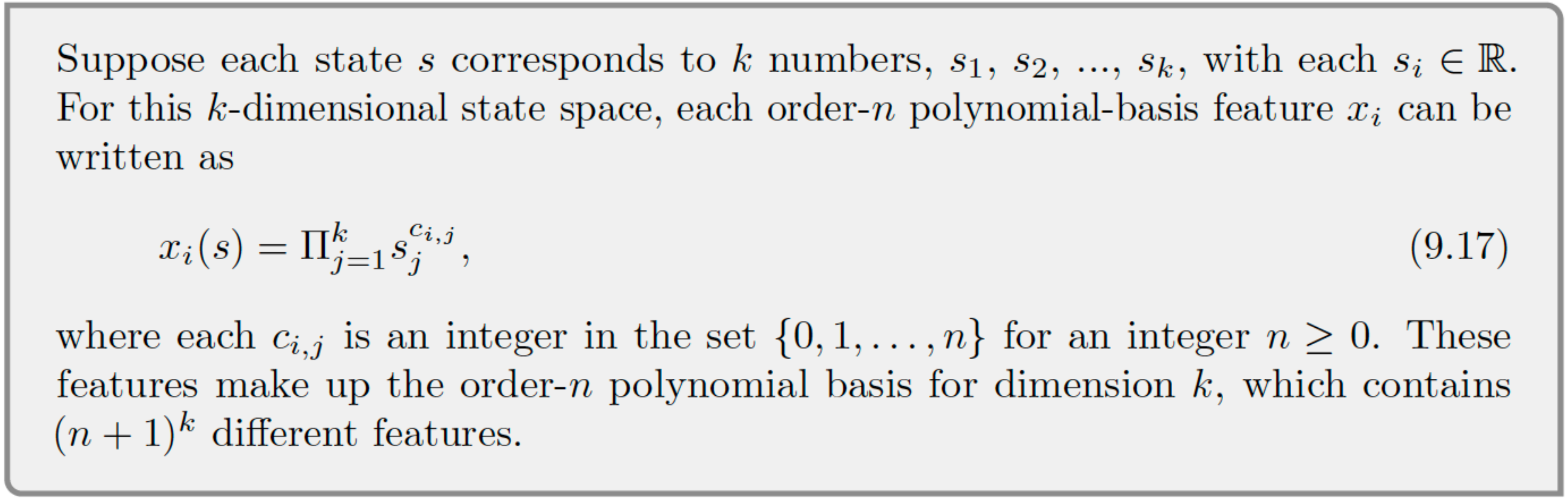


# Feature Construction for Linear Methods

- Polynomials
- Fourier Basis
- Coarse Coding
- Tile Coding
- Radial Basis Functions
- …

# Polynomials

- Let $s_1$ and $s_2$ be the two dimensions of state $s$
  - Is $\mathbf{x}(s) = (s_1, s_2)^T$ a good feature representation?
- Design a feature vector $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2)^T$
  - Allow the value to be non-zero if $s_1$ and $s_2$ are zero
  - Consider the interaction between dimensions



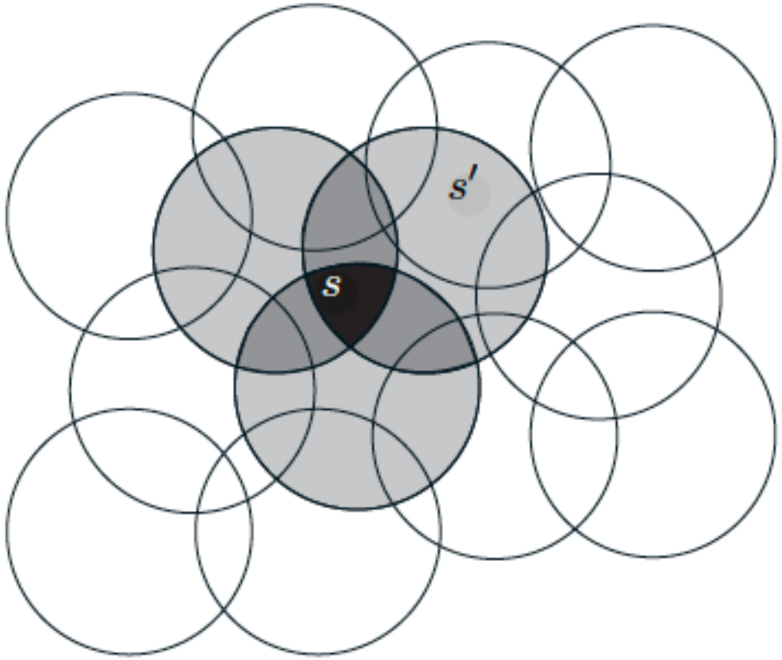
Suppose each state $s$ corresponds to $k$ numbers, $s_1$, $s_2$, ..., $s_k$, with each $s_i \in \mathbb{R}$. For this $k$-dimensional state space, each order-$n$ polynomial-basis feature $x_i$ can be written as

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}, \tag{9.17}$$

where each $c_{i,j}$ is an integer in the set $\{0, 1, \ldots, n\}$ for an integer $n \geq 0$. These features make up the order-$n$ polynomial basis for dimension $k$, which contains $(n+1)^k$ different features.

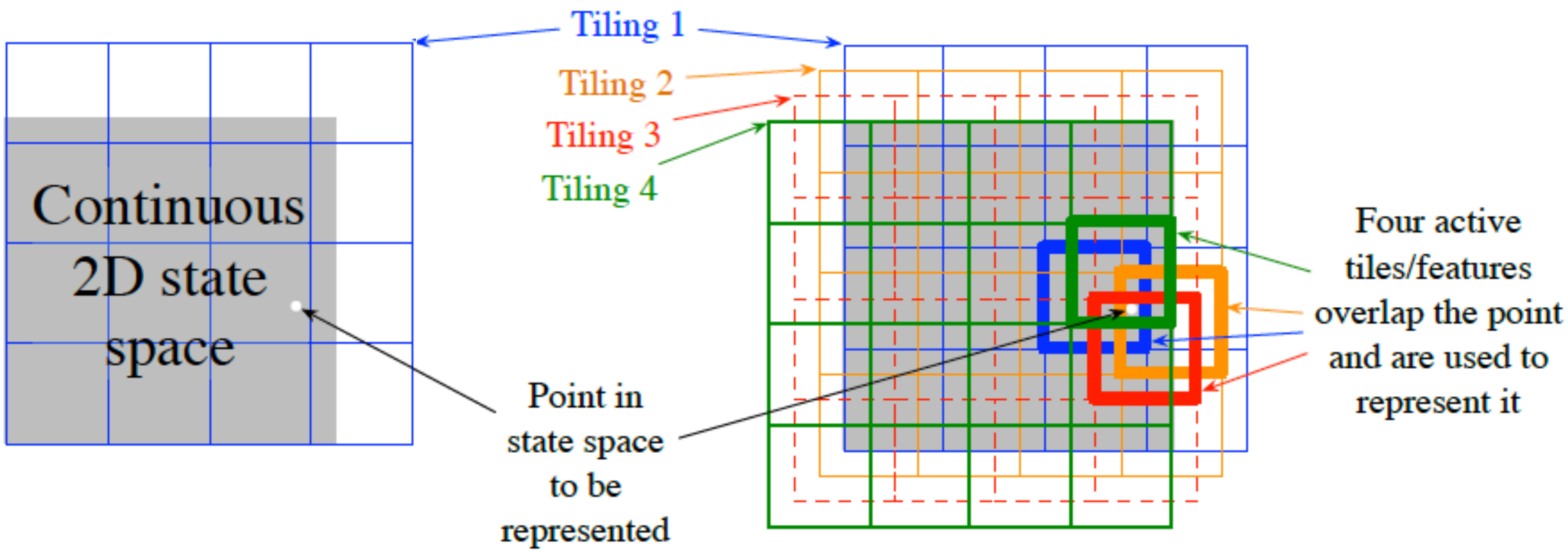
# Coarse Coding

- A set of binary features
- Given a state, which binary features are present indicate within which circles the state lies
- Generalization from state $s$ to state $s'$ depends on the number of their features whose receptive fields (i.e., circles) overlap
    - generalization determined by size, shape, and density



# Tile Coding

- a form of coarse coding for multi-dimensional continuous spaces



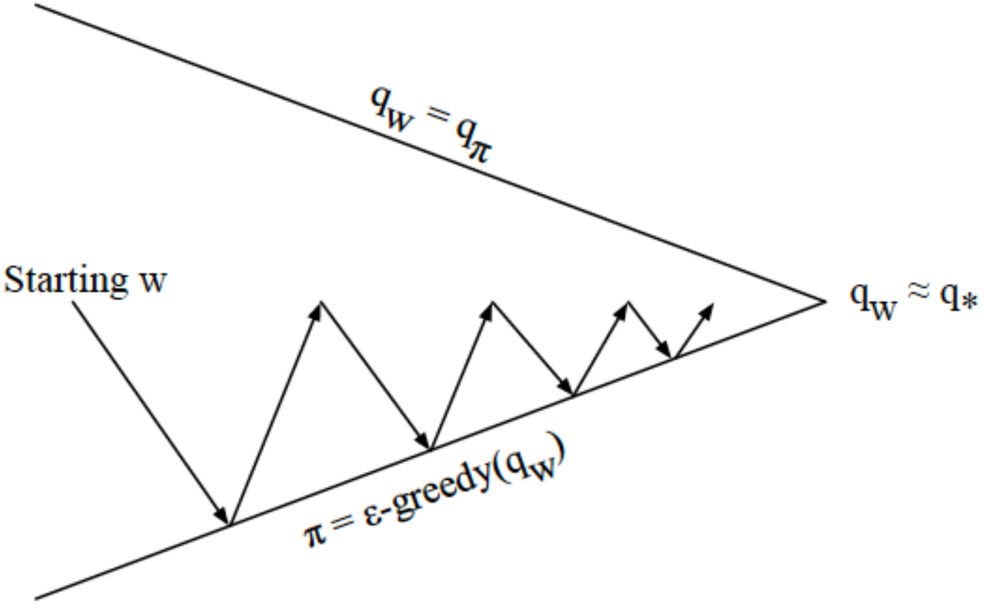
1 feature (with 16 values)

4 features (each with 16 values)

# Agenda

- Introduction
- Incremental Methods
  - On-policy prediction with value function approximation
  - On-policy control with value function approximation
  - Off-policy methods with approximation
- Batch Methods

# Control with Value Function Approximation



- Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
- Policy improvement $\epsilon$-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 value function $q_\pi(s, a)$

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

  - Is minimizing mean-squared error the right performance objective for policy optimization?

- Use stochastic gradient descent to find a local minimum

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

# Linear Action-Value Function Approximation

- Represent state and action by a feature vector

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

- Represent action-value function by linear combination of features

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

- 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 $G_t$

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

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

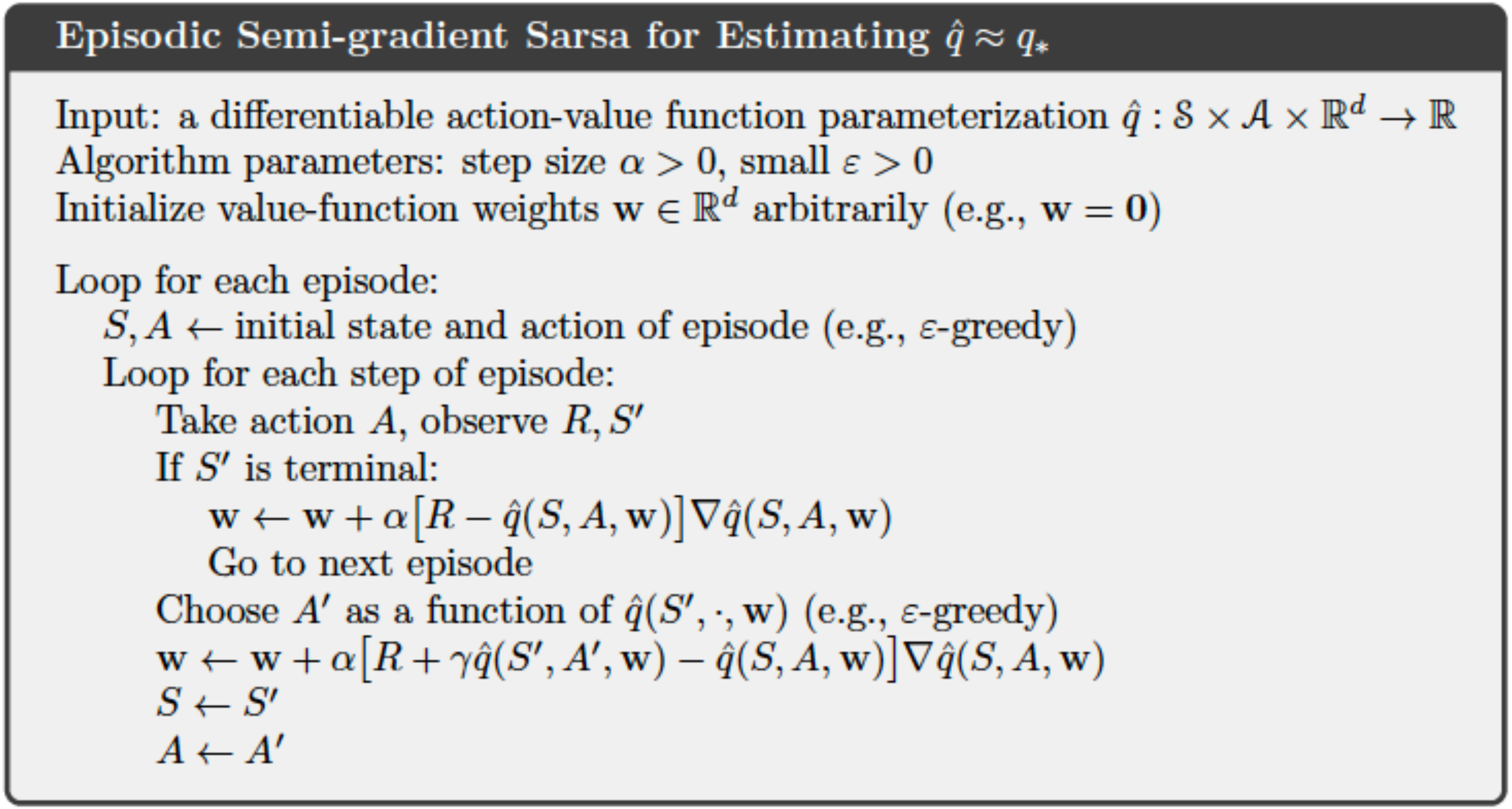
$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- For forward-view TD($\lambda$), the target is the action-value $\lambda$-return

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

- For backward-view TD($\lambda$), the target is the action-value $\lambda$-return

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



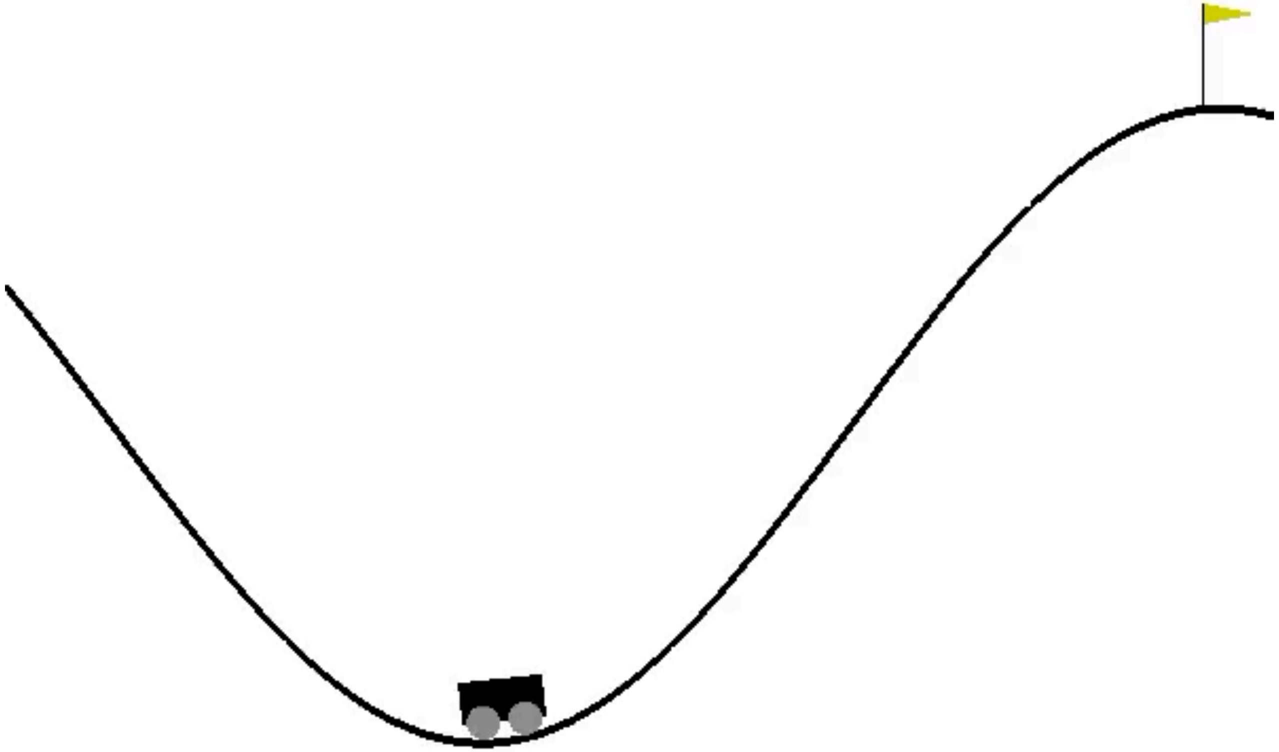
**Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$**

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

# Mountain Car Example

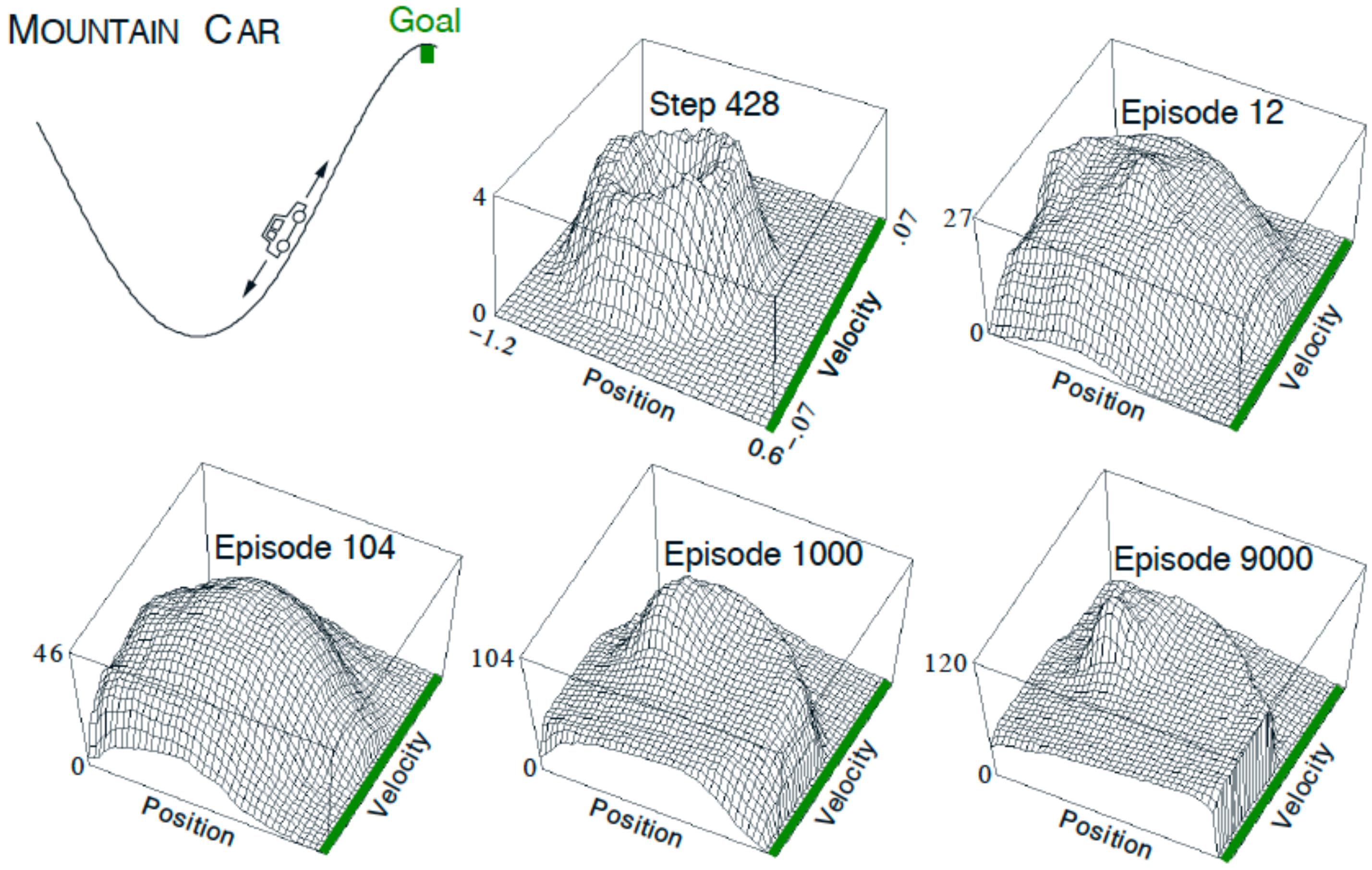
- Continuous states $(x_t, \dot{x}_t)$
- Three actions:
  - full throttle forward (+1),
  - full throttle reverse (−1),
  - zero throttle (0)
- Reward is -1 per unit time until reaching the goal



$$x_{t+1} \doteq bound\big[x_t + \dot{x}_{t+1}\big]$$

$$\dot{x}_{t+1} \doteq bound\big[\dot{x}_t + 0.001 A_t - 0.0025\cos(3x_t)\big],$$

# Linear Sarsa($\lambda$) with Tile Coding in Mountain Car



- 8 tilings
- Initially, $\boldsymbol{q} = \boldsymbol{0}$
- $\epsilon = 0$

# Agenda

- Introduction
- Incremental Methods
  - On-policy prediction with value function approximation
  - On-policy control with value function approximation
  - Off-policy methods with approximation
- Batch Methods

# Semi-Gradient Methods with Importance Sampling

- $\pi$: target policy, $\mu$: behavior policy
- per-step importance sampling ratio: $\rho_t = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
- Off-policy TD(0) update:

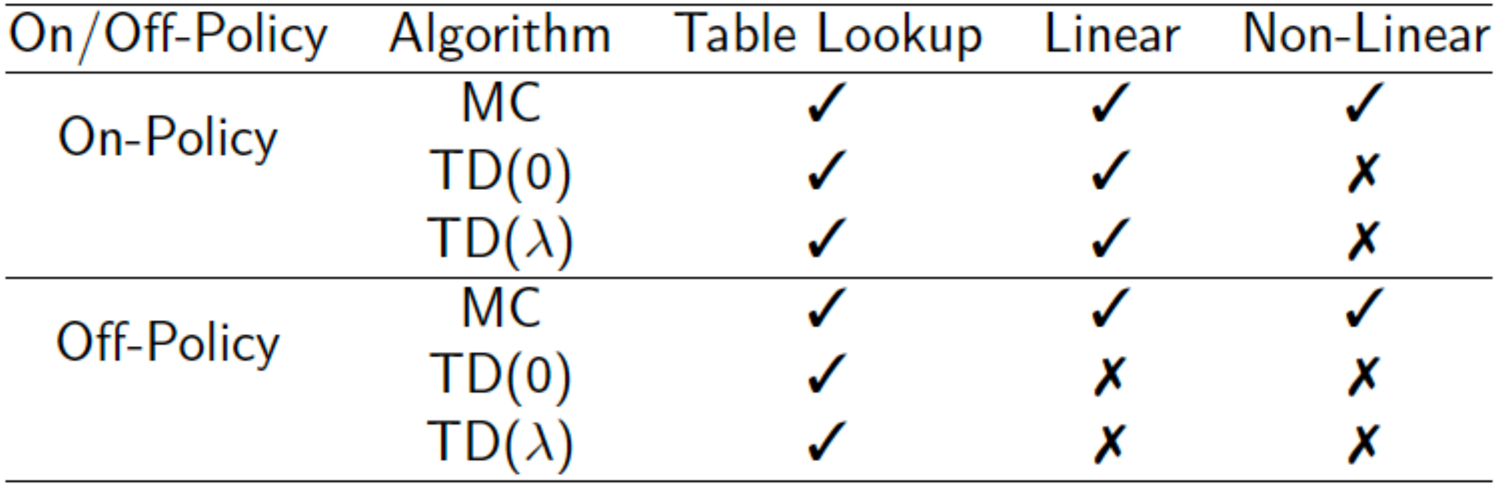
$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t),$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \rho_t \delta_t \nabla \hat{v}(S_t,\mathbf{w}_t),$$

# The Deadly Triad

- The danger of instability and divergence arises whenever we combine all of the following:
    - Function approximation
    - Bootstrapping
    - Off-policy training
- Instability arises even in the simpler prediction case, such as off-policy TD(0)

# Convergence of Prediction Algorithms

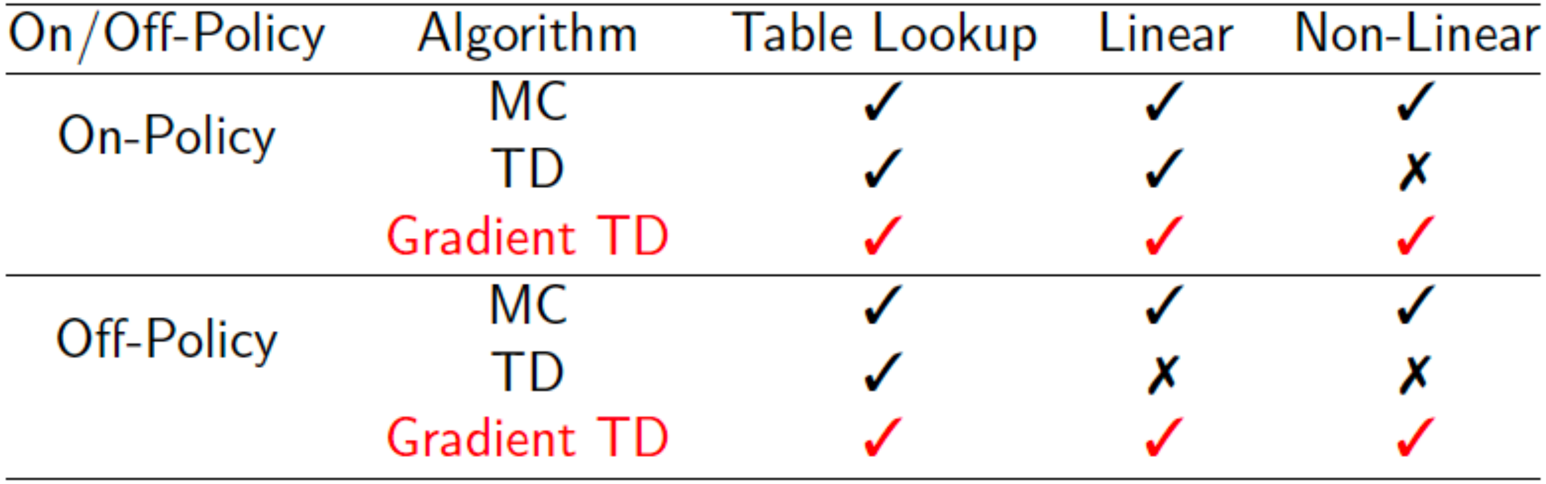


| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✓ | ✗ |
| | TD($\lambda$) | ✓ | ✓ | ✗ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✗ | ✗ |
| | TD($\lambda$) | ✓ | ✗ | ✗ |

# Gradient Temporal-Difference Learning

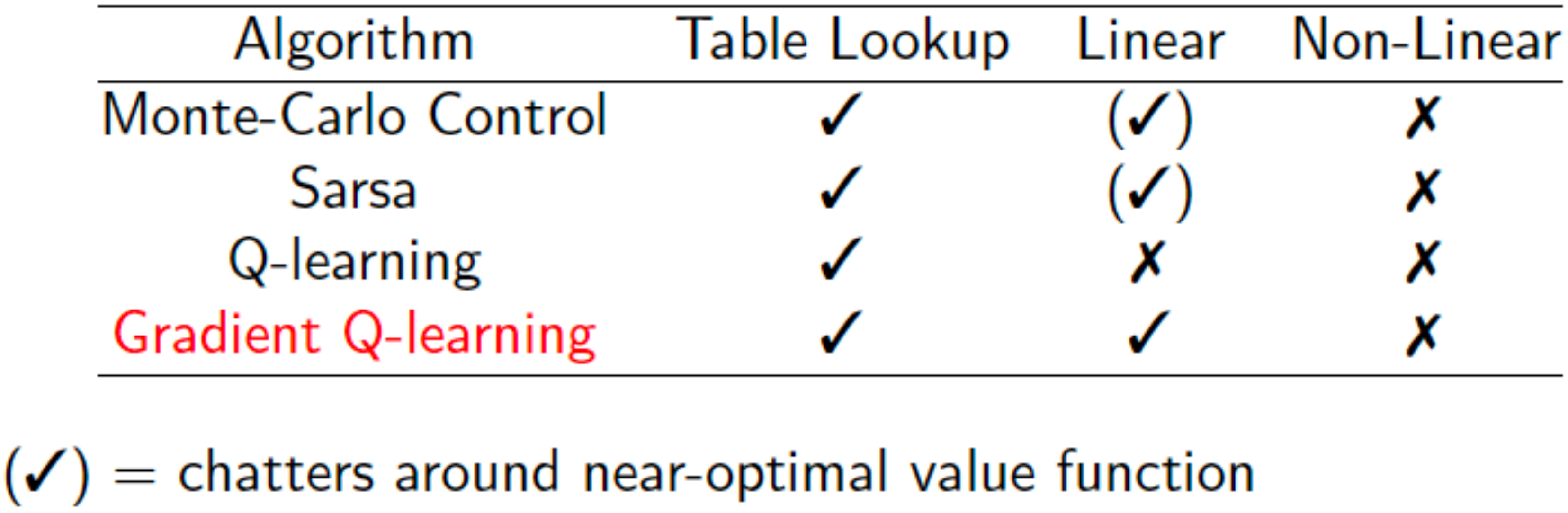
- TD does not follow the gradient of any objective function
- This is why TD can diverge when off-policy or using non-linear function approximation
- Gradient TD follows true gradient of projected Bellman error

$$\bar{\delta}(\mathbf{w}) = T^{\pi}\hat{v}(\mathbf{w}) - \hat{v}(\mathbf{w}) \qquad \overline{\text{PBE}}\,(\mathbf{w}) = \left\|\Pi\bar{\delta}(\mathbf{w})\right\|_{\mu}$$



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

# 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

# Agenda

- Introduction
- Incremental Methods
- 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 $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* D consisting of ⟨state, value⟩ pairs

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

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

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

# Stochastic Gradient Descent with Experience Replay

Given experience consisting of ⟨state, value⟩ pairs

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

Repeat:

1. Sample state, value from experience

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

2. Apply stochastic gradient descent update

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

Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}} \; LS(\mathbf{w})$$

# Experience Replay in Deep Q-Networks (DQN)

DQN uses experience replay and fixed Q-targets

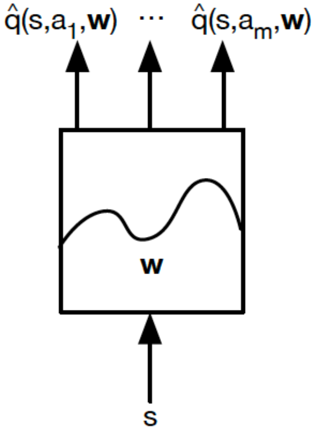
- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions $(s, a, r, s')$ from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $w^-$
- Optimize MSE between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

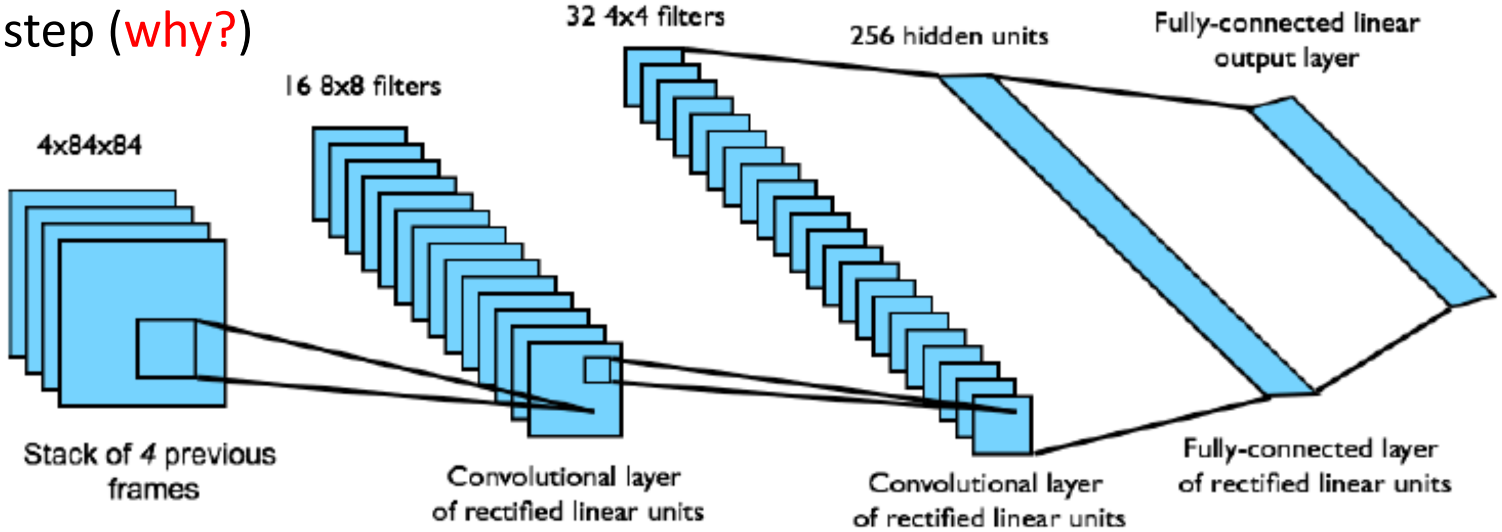
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

- Using variant of stochastic gradient descent

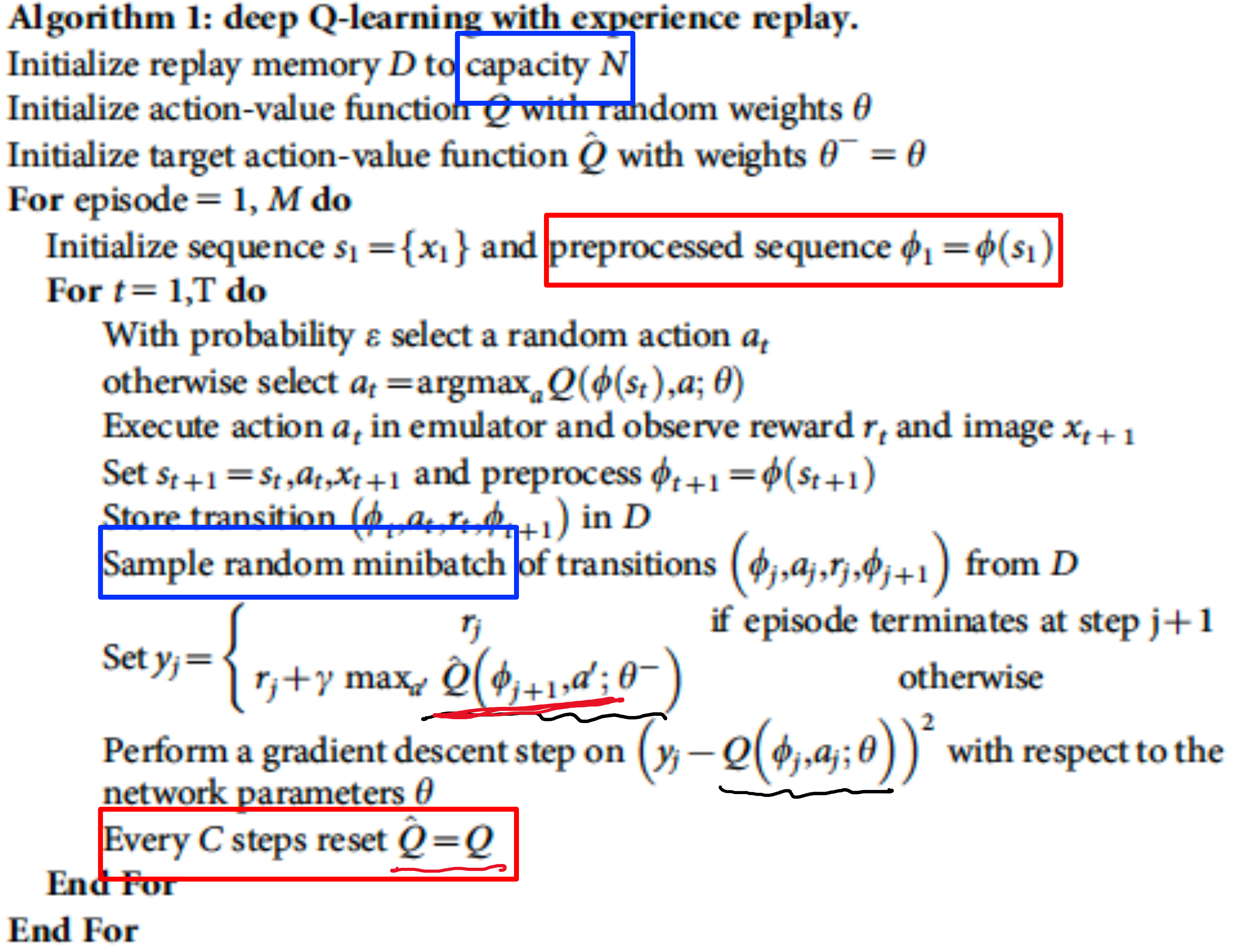
# DQN in Atari

- 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
  - Compute Q-values for all possible actions in a given state with only a single forward pass through the network.
- Reward is change in score for that step (why?)





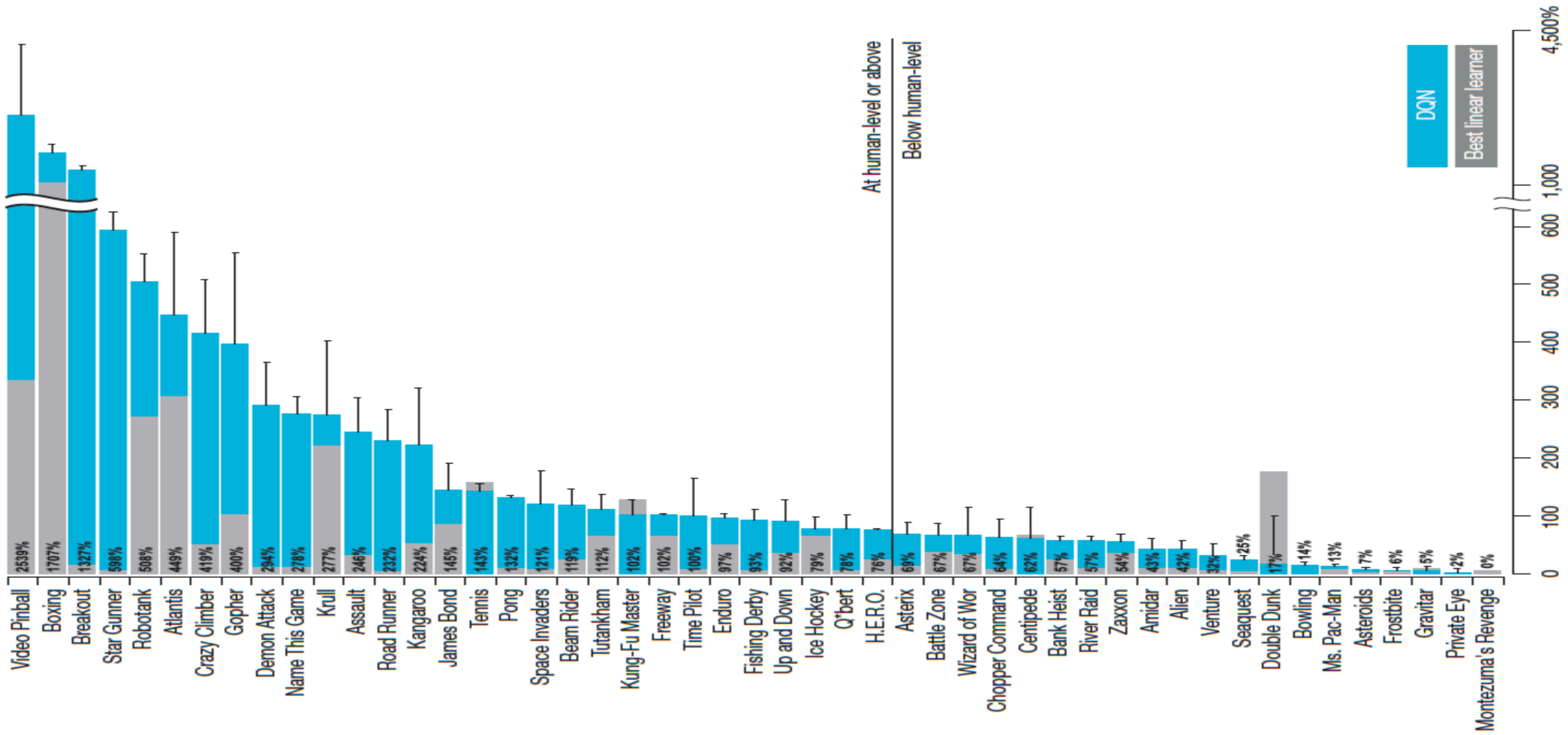
Network architecture and hyperparameters fixed across all games



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1$,T do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

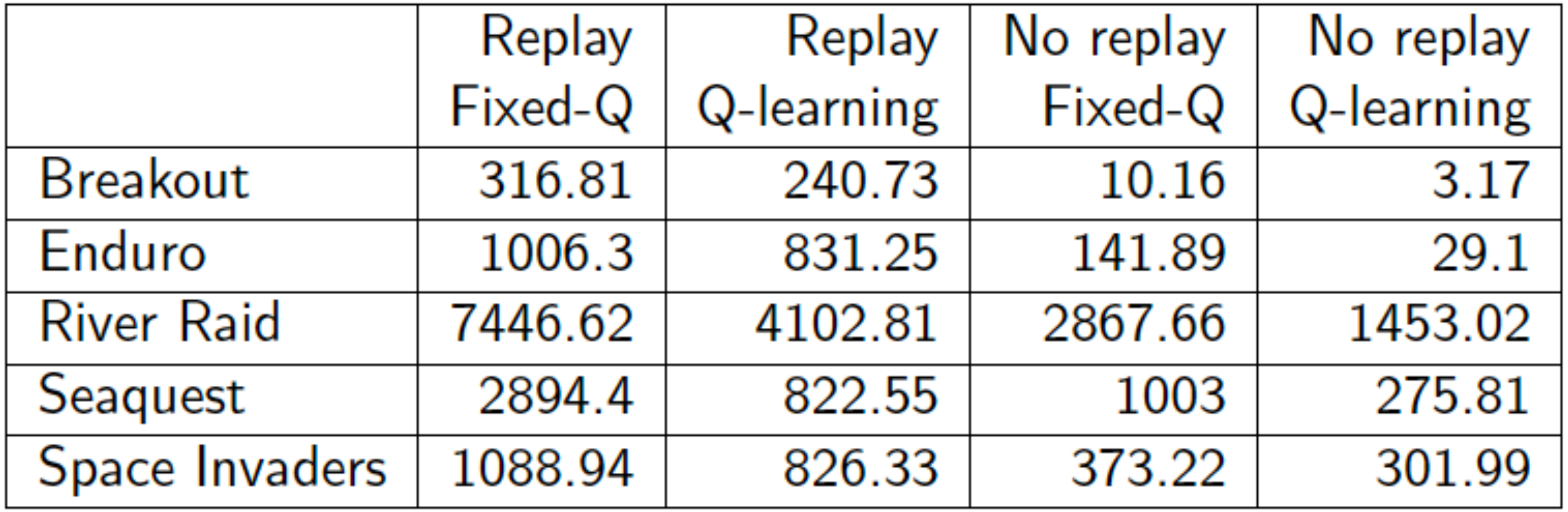
- Mnih, et al., "Human-level control through deep reinforcement learning", Nature, 2015

# DQN Results in Atari



$$\text{Normalized performance of DQN} = 100\times\frac{\text{DQN score}-\text{random play score}}{\text{human score}-\text{random play score}}$$

# 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 |

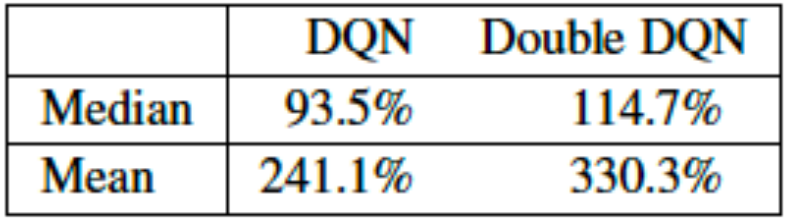
# Double DQN

- Double Q-learning target

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

- Double DQN target

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax} Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^{-})$$



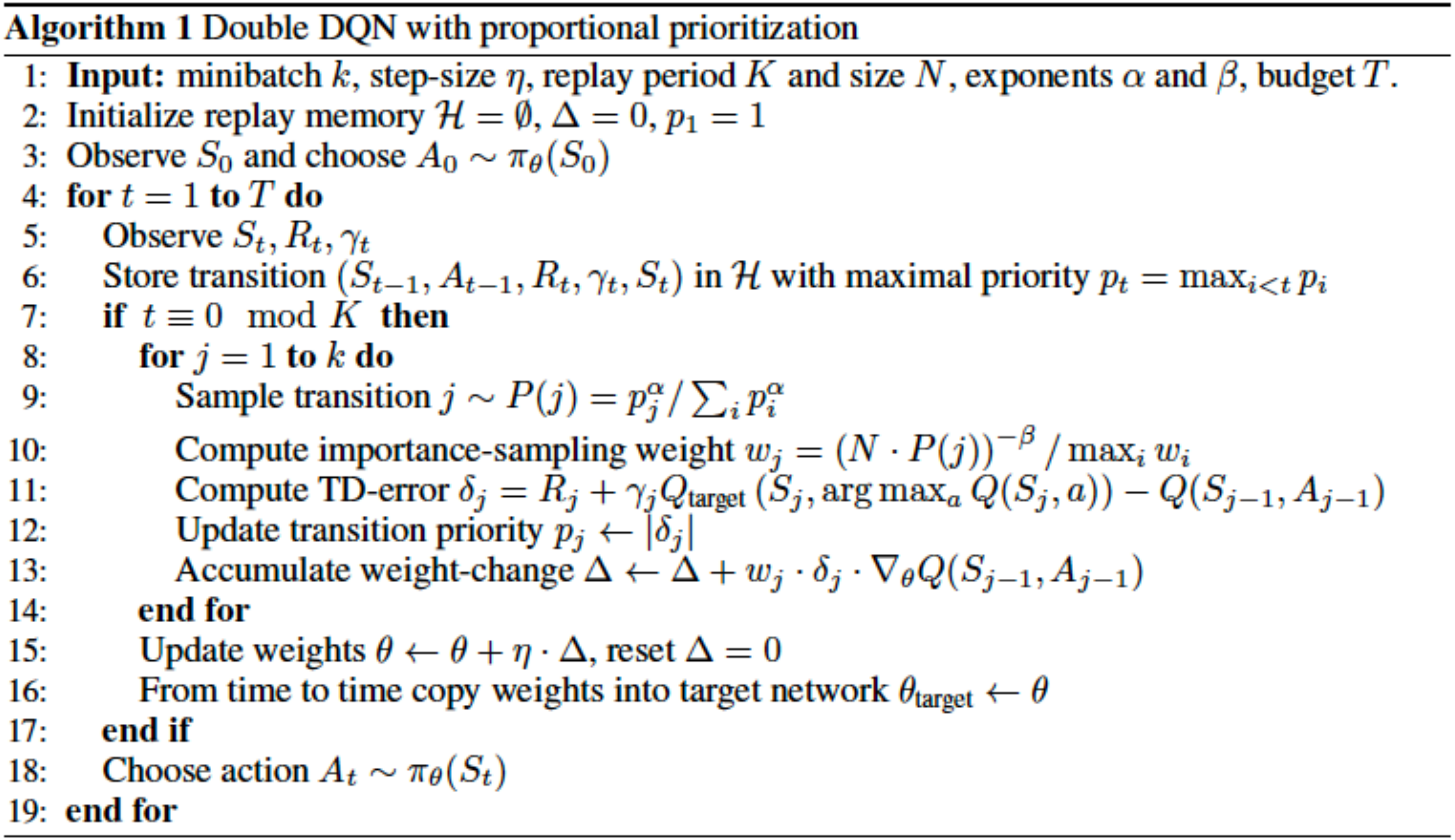
| | DQN | Double DQN |
|---|---|---|
| Median | 93.5% | 114.7% |
| Mean | 241.1% | 330.3% |

Normalized performance up to 5 minutes of play on 49 Atari games

- van Hasselt, et al., “Deep Reinforcement Learning with Double Q-learning”, AAAI, 2016

# DQN with Prioritized Experience Replay

- An RL agent can learn more effectively from some transitions than from others.
  - Transitions may be more or less surprising, redundant, or task-relevant
  - Some transitions may not be immediately useful, but might become so when the agent competence increases
- TD-error prioritization: transitions with higher TD-error $\delta_i$ are replayed more frequently
  - $p_i$ - priority of transition $i$: (a) $p_i = |\delta_i|$ ; (b) $p_i = |\delta_i| + \epsilon$; (c) $p_i = \frac{1}{\mathrm{rank}(i)}$
- Stochastic Prioritization
  - Probability of sampling transition $i$: $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$
- Annealing the Bias
  - Q-learning update with weighted importance-sampling with weight $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$
  - Linearly anneal $\beta$ from $\beta_0$ to 1: the unbiased nature of the updates is most important near convergence at the end of training



**Algorithm 1** Double DQN with proportional prioritization

```
Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to $T$ do
    Observe $S_t, R_t, \gamma_t$
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ then
        for $j = 1$ to $k$ do
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
        From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for
```

# Least Squares Prediction

- And *experience* $\mathcal{D}$ consisting of ⟨state, value⟩ pairs

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

- Least squares algorithms find parameter vector $\mathbf{w}$ minimizing

$$LS(\mathbf{w}) = \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2$$

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

# Linear Least Squares Prediction (2)

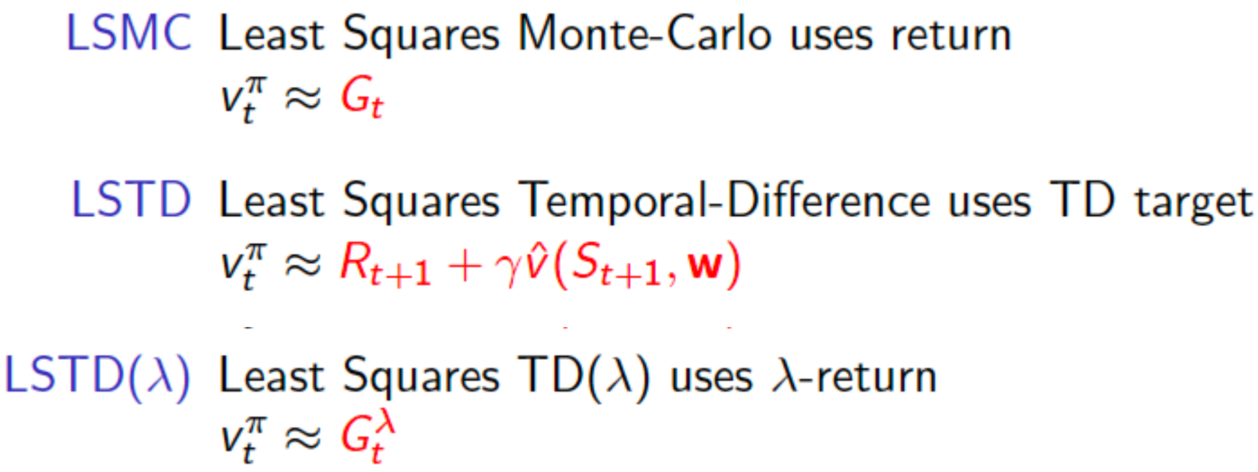
- At minimum of $LS(\mathbf{w})$, the expected update must be zero

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[\Delta \mathbf{w}\right] &= 0 \\
\alpha \sum_{t=1} \mathbf{x}(s_t)(v_t^{\pi} - \mathbf{x}(s_t)^{\top}\mathbf{w}) &= 0 \\
\sum^{T} \mathbf{x}(s_t) v_t^{\pi} &= \sum^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\mathbf{w} \\
\mathbf{w} &= \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}
\end{aligned}
$$

- For $N$ features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

# Linear Least Squares Prediction Algorithms

- We do not know true values $v_t^\pi$
- In practice, our “training data” must use noisy or biased samples of $v$



LSMC Least Squares Monte-Carlo uses return
$v_t^\pi \approx G_t$

LSTD Least Squares Temporal-Difference uses TD target
$v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

LSTD($\lambda$) Least Squares TD($\lambda$) uses $\lambda$-return
$v_t^\pi \approx G_t^\lambda$

- In each case solve directly for fixed point of MC / TD / TD($\lambda$)

# Linear Least Squares Prediction Algorithms (2)

LSMC

$$0 = \sum_{t=1}^{T} \alpha (G_t - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)$$

$$\mathbf{w} = \left( \sum_{t=1}^{T} \mathbf{x}(S_t) \mathbf{x}(S_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

LSTD

$$0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)$$

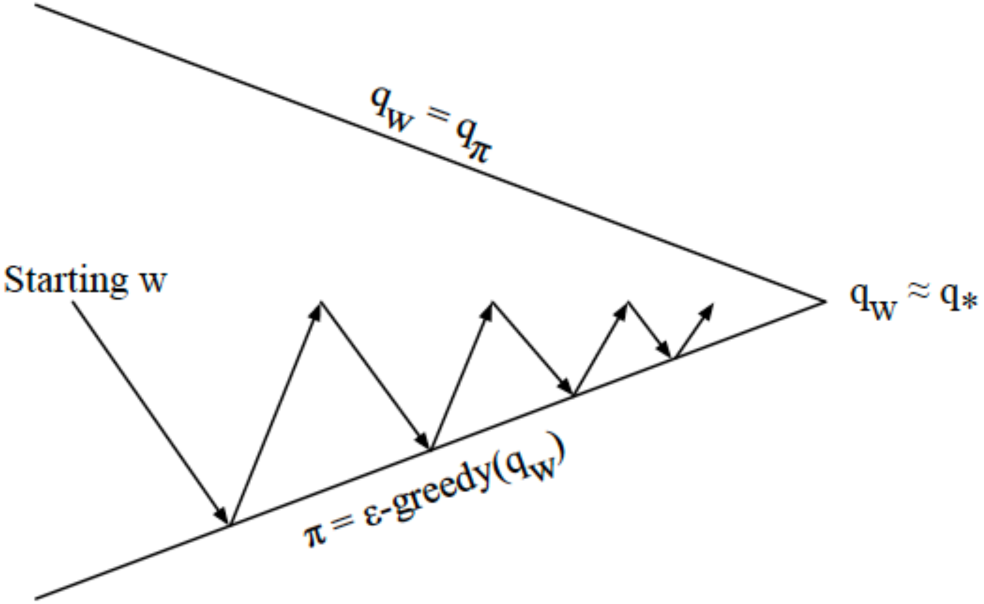
$$\mathbf{w} = \left( \sum_{t=1}^{T} \mathbf{x}(S_t) (\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) R$$

LSTD($\lambda$)

$$0 = \sum_{t=1}^{T} \alpha \delta_t E_t$$

$$\mathbf{w} = \left( \sum_{t=1}^{T} E_t (\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

# Least Squares Policy Iteration



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

# Least Squares Action-Value Function Approximation

- Approximate action-value function $q_\pi(s,a)$
- using linear combinations of features $\mathbf{x}(s,a)$

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

- Minimize least squares error between $\hat{q}(s,a,\mathbf{w})$ and $q_\pi(s,a)$
- from experience generated using policy $\pi$
- consisting of ⟨(state,action), value⟩ pairs

$$\mathcal{D} = \{\langle (s_1, a_1), v_1^\pi \rangle, \langle (s_2, a_2), v_2^\pi \rangle, \ldots, \langle (s_T, a_T), v_T^\pi \rangle\}$$

# Least Squares Q-Learning

- Consider the following linear Q-learning update

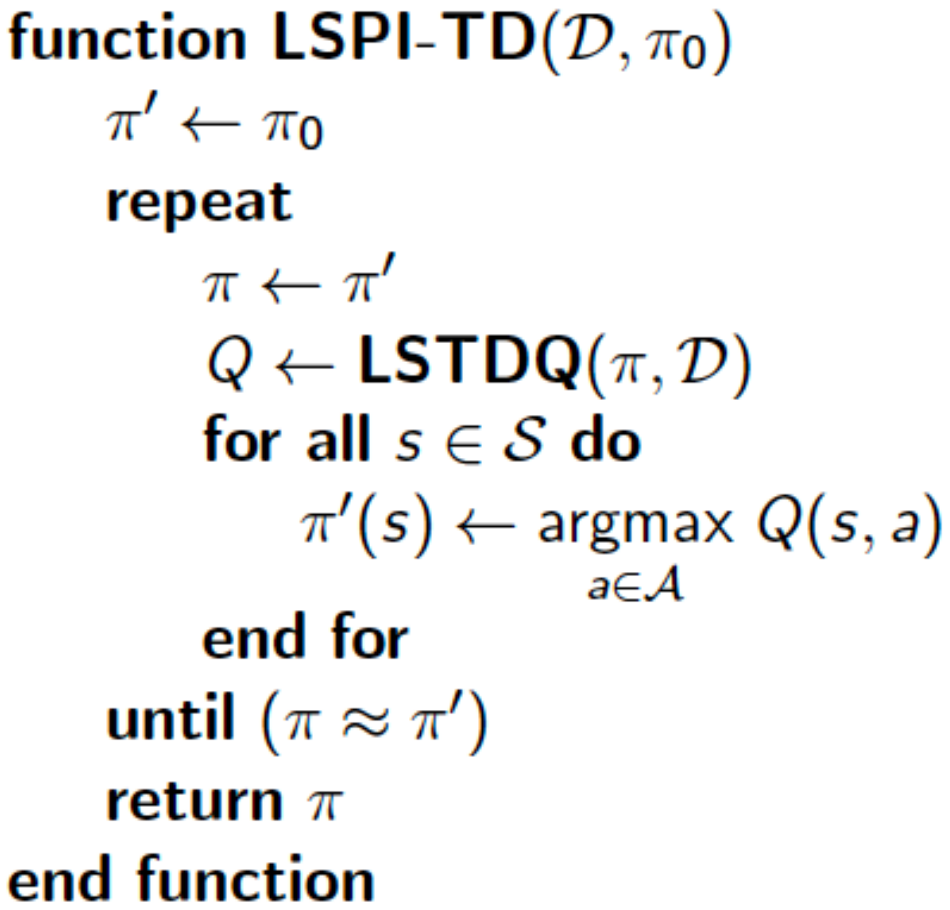
$$\begin{aligned} \delta &= R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\ \Delta \mathbf{w} &= \alpha \delta \mathbf{x}(S_t, A_t) \end{aligned}$$

- LSTDQ algorithm: solve for total update = zero

$$0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \mathbf{x}(S_t, A_t)$$

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

# Least Squares Policy Iteration (LSPI)



```
function LSPI-TD($\mathcal{D}, \pi_0$)
    $\pi' \leftarrow \pi_0$
    repeat
        $\pi \leftarrow \pi'$
        $Q \leftarrow$ LSTDQ($\pi, \mathcal{D}$)
        for all $s \in \mathcal{S}$ do
            $\pi'(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a)$
        end for
    until ($\pi \approx \pi'$)
    return $\pi$
end function
```