# Reinforcement Learning

## Lecture 8

### Value Function Approximation (v and q)

### DeepQ Learning

Robert Peharz

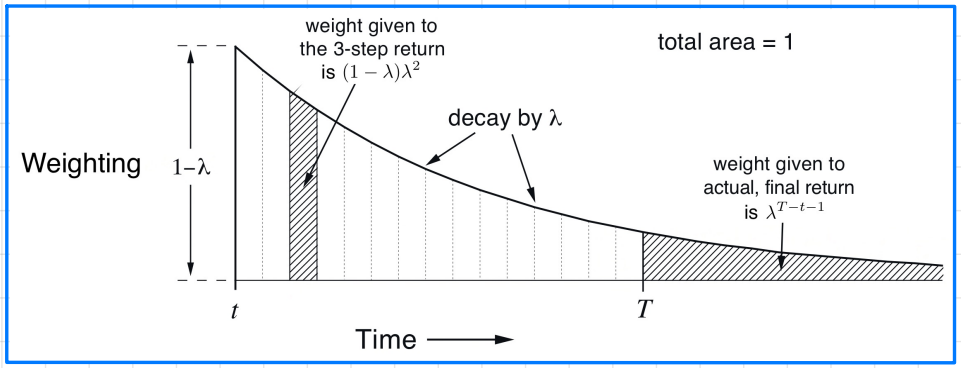
Institute of Theoretical Computer Science
Graz University of Technology
Winter Term 2023/24

# Recap: λ-Return

$$\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} = 1$$

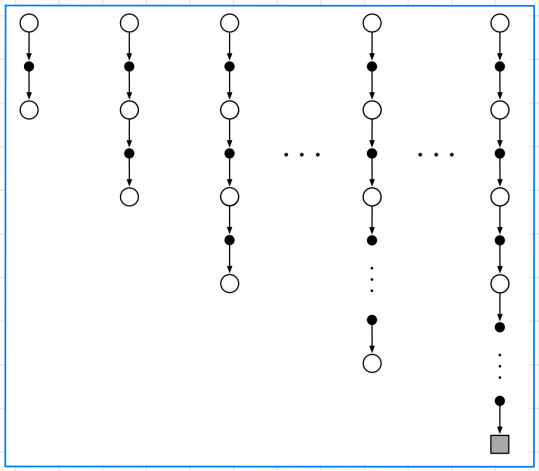
stick-breaking weights

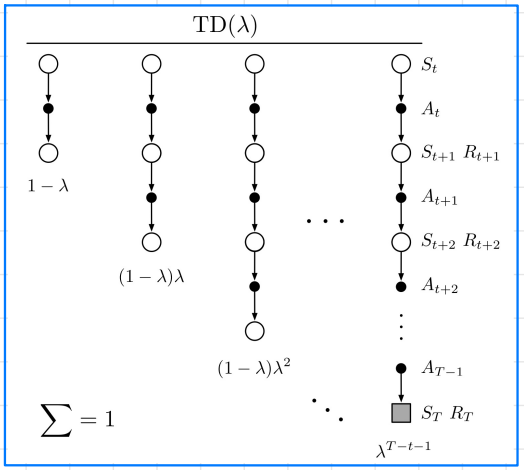
$0 \le \lambda \le 1$




## n-step TD

1 2 3 ... ∞ = MC






## n-step return

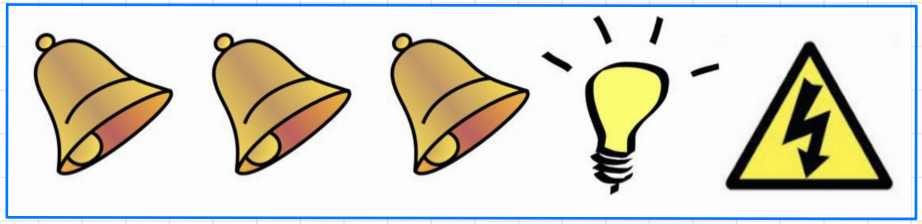
using v estimate = "bootstrapping"

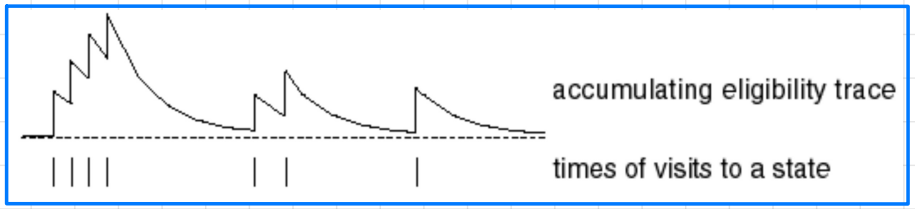
$$\hat{G}_t^n := \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n v(S_{t+n})$$

## λ-return

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{G}_t^n$$

# Recap: TD(λ)






$t=0$: $e(s) \leftarrow 0$

$t>0$: $e(s) \leftarrow \lambda \gamma e(s) + \mathbb{1}[S_t = s]$

## TD(λ)

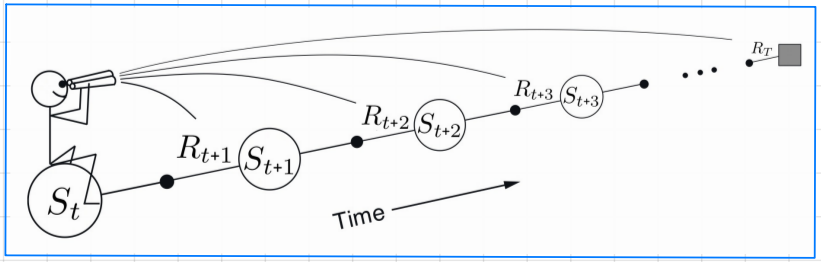
$e(s) \leftarrow \lambda \gamma e(s) + \mathbb{1}[S = s] \quad \forall s$

$\delta \leftarrow R + \gamma v(S') - v(S)$ // TD error

$v(s) \leftarrow v(s) + \alpha \delta e(s) \quad \forall s$

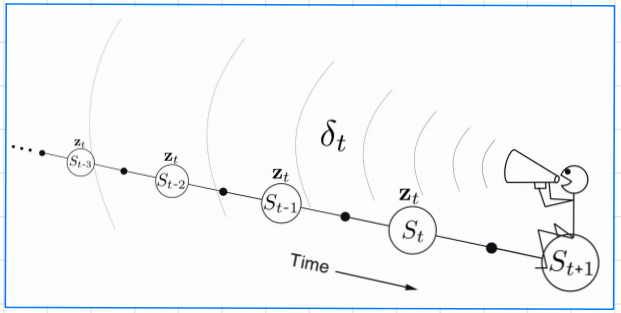
eligibility traces

## λ-Return




≈

## TD(λ)

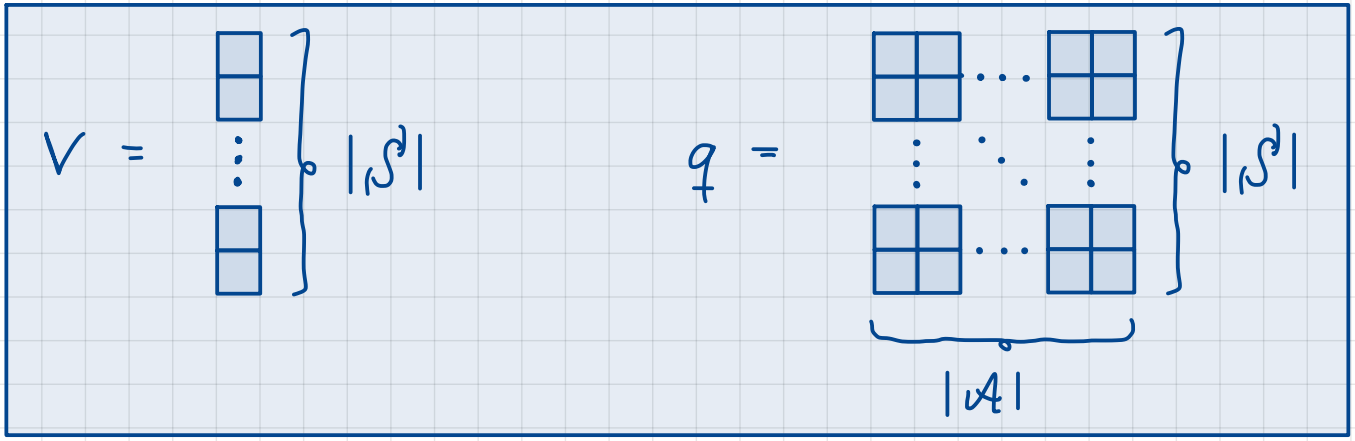



exactly equivalent, if TD(λ) performs accumulated update at end of episode

# Value Function Approximation

# Why Function Approximation?

- Reinforcement Learning = { learning $v_\pi$ (prediction) or learning $q^*$ (control)
- So far, we were considering tabular methods, i.e. we were explicitly storing $V(s)$ or $q(s,a)$



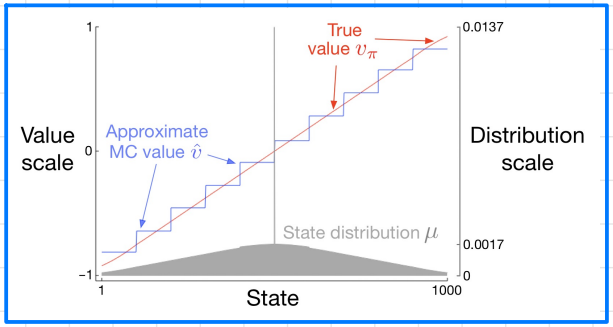

- For "interesting" problems, these are too many values

- Game of Go: $10^{170}$ states, continuous state spaces, ...
- Even when we can store the value functions, updating all states/actions might be too expensive
  → generalize over many states

# Function Approximators

Images: Sutton & Barto

- aggregated states




- Linear

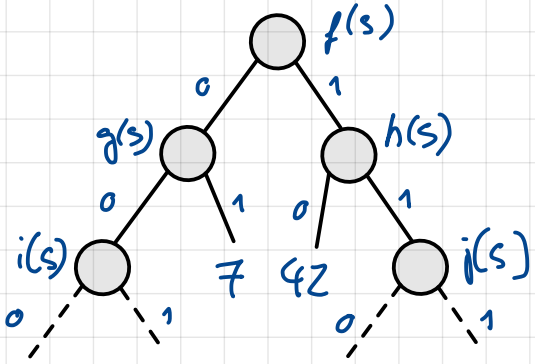
$$v(s) = \sum_{i=1}^{K} w_i \, x_i(s)$$

$$q(s,a) = \sum_{i=1}^{K} w_i \, x_i(s,a)$$

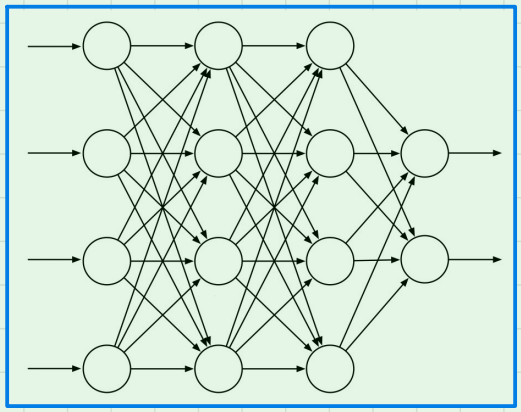
$x_i$ ... fixed non-linear features of $s$ or $s,a$ pair

$w_i$ ... tunable weights

- Decision Trees




- Neural Networks



Most prominent nowadays

# Differentiable Approximators

- we will consider approximators that are differentiable

$$v(s) \approx \hat{v}(s, \underline{w}) \quad \text{or} \quad q(s,a) \approx \hat{q}(s,a,\underline{w})$$

$$\underline{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix}$$

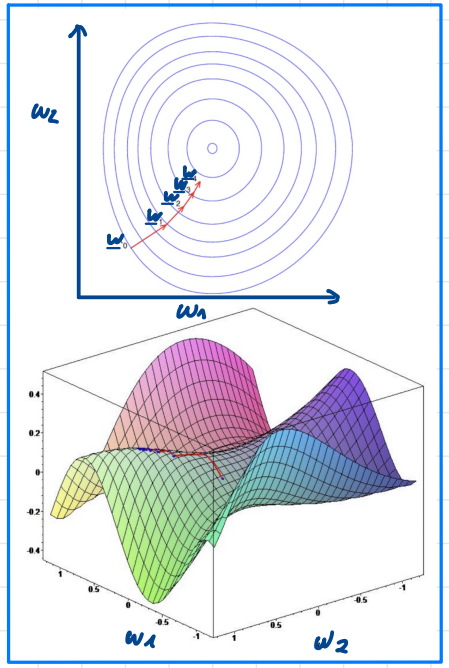
- where $\underline{w}$ is a weight vector
- usually $|\underline{w}| = d \ll |\mathcal{S}| \quad (|\mathcal{S}| \times |\mathcal{A}|)$
- we assume we can compute the gradient

$$\nabla_{\underline{w}} \hat{v}(s, \underline{w}) := \begin{pmatrix} \frac{\partial \hat{v}(s,\underline{w})}{\partial w_1} \\ \frac{\partial \hat{v}(s,\underline{w})}{\partial w_2} \\ \vdots \\ \frac{\partial \hat{v}(s,\underline{w})}{\partial w_d} \end{pmatrix} \qquad \nabla_{\underline{w}} \hat{q}(s,a,\underline{w}) := \begin{pmatrix} \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_1} \\ \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_2} \\ \vdots \\ \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_d} \end{pmatrix}$$

- in particular, neural networks and linear models
- neural nets allow for automatic differentiation (backprop)

# Gradient Descent for Supervised Learning

Image: D. Silver




- gradient is the direction of steepest increase
- to minimize a function $f(\underline{w})$, repeatedly take small step in direction of negative gradient

Gradient Descent:

init $\underline{w}$
repeat
$\underline{w} \leftarrow \underline{w} - \alpha \nabla_{\underline{w}} f(\underline{w})$ (stepsize $\alpha$)
until $\nabla_{\underline{w}} f(\underline{w}) \approx 0$

in supervised learning, we have a dataset $\mathcal{D} = \{(\underline{x}_1, y_1), (\underline{x}_2, y_2), \ldots, (\underline{x}_N, y_N)\}$ where $\underline{x}_i$ is an input vector and $y_i$ is a corresponding target.

$y_i$ might be a discrete value (classification)
or continuous value (regression)

goal: learn to predict $y$ from $\underline{x}$, i.e. $y \approx f(\underline{x}, \underline{w})$

$$\min_{\underline{w}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\underline{x}_i, \underline{w}), y_i)$$

training loss

where $\ell(\cdot, \cdot)$ is a loss function comparing $f(\underline{x}, \underline{w})$ and $y$

# Stochastic Gradient Descent (SGD)

- training loss $\mathcal{L}$ and its gradient need <u>full pass</u> over $N$ samples
- if $N$ is large, vanilla gradient descent becomes slow
- **<u>SGD</u>**: use a <u>small batch</u> of $K$ <u>randomly selected samples</u>:

$$\mathcal{L} := \frac{1}{N}\sum_{i=1}^{N} \ell(f(\underline{x}_i, \underline{w}), y_i) \approx \frac{1}{K}\sum_{i=1}^{K} \ell(f(\underline{x}_i, \underline{w}), y_i) =: \mathcal{L}_K$$

$$\nabla_{\underline{w}} \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\underline{w}} \ell(\dots) \approx \frac{1}{K}\sum_{i=1}^{K} \nabla_{\underline{w}} \ell(\dots) = \nabla_{\underline{w}} \mathcal{L}_K \qquad K \ll N$$

- <u>batch loss</u> $\mathcal{L}_K$ and <u>batch gradient</u> $\nabla_{\underline{w}} \mathcal{L}_K$ are noisy <u>but unbiased</u>

$$\mathbb{E}[\mathcal{L}_K] = \mathcal{L} \qquad \mathbb{E}[\nabla_{\underline{w}} \mathcal{L}_K] = \nabla_{\underline{w}} \mathcal{L}$$

let $\alpha_i$ be the SGD stepsize at iteration $i$

if $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$: SGD converges to a local minimum of $\mathcal{L}$.

(e.g. $\alpha = 1/i$)

# Value Function Approximation via SGD

- treat value function approximation as a supervised problem
- using quadratic loss (least squares)

$$\ell(\hat{v}(s,\underline{w}), v_\pi(s)) = \frac{1}{2}\left(v_\pi(s) - \hat{v}(s,\underline{w})\right)^2$$

$$\nabla_{\underline{w}} \ell = -\left(v_\pi(s) - \hat{v}(s,\underline{w})\right) \nabla_{\underline{w}} \hat{v}(s,\underline{w})$$

(P)rediction

$$\ell(\hat{q}(s,a,\underline{w}), q_\pi(s,a,\underline{w})) = \frac{1}{2}\left(q_\pi(s,a,\underline{w}) - \hat{q}(s,a,\underline{w})\right)^2$$

$$\nabla_{\underline{w}} \ell = -\left(q_\pi(s,a) - \hat{q}(s,a,\underline{w})\right) \nabla_{\underline{w}} \hat{q}(s,a,\underline{w})$$

(C)ontrol

- follow policy and perform SGD updates: $\underline{w} \leftarrow \underline{w} - \alpha \nabla_{\underline{w}} \ell$
- underlying objectives: $\mathbb{E}_\pi\left[\frac{1}{2}\left(v_\pi(s) - \hat{v}(s,\underline{w})\right)^2\right]$ $\mathbb{E}_\pi\left[\frac{1}{2}\left(q_\pi(s,a) - \hat{q}(s,a,\underline{w})\right)^2\right]$

- what is the problem with this algorithm?

# Monte Carlo Function Approximation

Recall and compare

**Tabular MC update:**

$$V(S) \leftarrow V(S) + \alpha (G_t - V(S))$$

$$\nabla_{\underline{w}} \ell = -\left(v_\pi(s) - \hat{v}(s, \underline{w})\right) \nabla_{\underline{w}} \hat{v}(s, \underline{w}) \qquad (P)$$

$$\nabla_{\underline{w}} \ell = -\left(q_\pi(s, a) - \hat{q}(s, a, \underline{w})\right) \nabla_{\underline{w}} \hat{q}(s, a, \underline{w}) \qquad (C)$$

since $G$ is unbiased, also the gradient will be unbiased, so this is just SGD noise

- the <u>target values</u> $v_\pi(s)$ $(q_\pi(s,a))$ are <u>not known</u>
- if they were, no need to approximate them...
- <u>idea</u>: use MC to get <u>unbiased estimate of gradients</u>

initialize $\underline{w}$

repeat // for each episode

generate episode with $\pi$ $(\varepsilon\text{-greedy}(q))$

$G \leftarrow 0$

for $t = T-1 \dots 0$:

$G \leftarrow R_{t+1} + \gamma G$

$\underline{w} \leftarrow \underline{w} + \alpha \left(G - \hat{v}(S_t, \underline{w})\right) \nabla_{\underline{w}} \hat{v}(S_t, \underline{w})$

$\left(\underline{w} \leftarrow \underline{w} + \alpha \left(G - \hat{q}(S_t, A_t, \underline{w})\right) \nabla_{\underline{w}} \hat{q}(S_t, A_t, \underline{w})\right)$

for control

# TD(0) and Sarsa Function Approximation

Instead of estimating $v_\pi(s)$ ($q_\pi(s,a)$) with MC, plug in TD estimates:

$$\nabla_{\underline{w}} \ell = -\left(v_\pi(s) - \hat{v}(s,\underline{w})\right) \nabla_{\underline{w}} \hat{v}(s,\underline{w})$$

$$\approx -\left(R_{t+1} + \gamma \underline{\hat{v}(s',\underline{w})} - \hat{v}(s,\underline{w})\right) \nabla_{\underline{w}} \hat{v}(s,\underline{w}) \qquad (P)$$

"ignored" in the gradient computation
→ "semi-gradient" method

$$\nabla_{\underline{w}} \ell = -\left(q_\pi(s,a) - \hat{q}(s,a,\underline{w})\right) \nabla_{\underline{w}} \hat{q}(s,a,\underline{w})$$

$$\approx -\left(R_{t+1} + \gamma \underline{\hat{q}(s',a',\underline{w})} - \hat{q}(s,a,\underline{w})\right) \nabla_{\underline{w}} \hat{q}(s,a,\underline{w}) \qquad (C)$$

→ "semi-gradient" method

- bootstrapping <=> "semi-gradient"
- TD does not work reliably with non-linear approximators !

# TD(0) and Sarsa Function Approximation

initialize $\underline{w}$

repeat // for each episode

  initialize $S$

  select $A \sim \hat{\pi}(a|S)$ $(\varepsilon\text{-greedy}(q))$

  repeat

    take action $A$, observe $R, S'$

    if $S'$ is terminal:

      $\underline{w} \leftarrow \underline{w} + \alpha\left[R - \hat{v}(S, \underline{w})\right] \nabla_{\underline{w}} \hat{v}(S, \underline{w})$

      $\left(\underline{w} \leftarrow \underline{w} + \alpha\left[R - \hat{q}(S, A, \underline{w})\right] \nabla_{\underline{w}} \hat{q}(S, A, \underline{w})\right)$

      break // start next episode

    select $A' \sim \hat{\pi}(a|S)$ $(\varepsilon\text{-greedy}(q))$

    $\underline{w} \leftarrow \underline{w} + \alpha\left[R + \gamma \hat{v}(S', \underline{w}) - \hat{v}(S, \underline{w})\right] \nabla_{\underline{w}} \hat{v}(S, \underline{w})$

    $\left(\underline{w} \leftarrow \underline{w} + \alpha\left[R + \gamma \hat{q}(S', A', \underline{w}) - \hat{q}(S, A, \underline{w})\right] \nabla_{\underline{w}} \hat{q}(S, A, \underline{w})\right)$

    $S, A \leftarrow S', A'$

(comand alternatives in color are for control, i.e. Sarsa)

# Linear Function Approximation

- assume a set of $d$ features $x_1, x_2, \ldots, x_d$ where

  $x_i : \mathcal{S} \to \mathbb{R}$ (for $\hat{v}$) $\quad x_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (for $\hat{q}$) $\quad \underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$

- the linear approximator is given as

$$\hat{v}(s, \underline{w}) = \underline{w}^T \underline{x} = \sum_{i=1}^{d} w_i \, x_i(s) \qquad \hat{q}(s, a, \underline{w}) = \underline{w}^T \underline{x} = \sum_{i=1}^{d} w_i \, x_i(s, a)$$

- gradients: $\nabla_{\underline{w}} \hat{v} = \underline{x}$ $\qquad \nabla_{\underline{w}} \hat{q} = \underline{x}$

- objective is expected quadratic loss

$$\mathbb{E}_{\pi}\left[\frac{1}{2}\left(\hat{v}(s, \underline{w}) - v_{\pi}(s)\right)^2\right] \qquad \mathbb{E}_{\pi}\left[\frac{1}{2}\left(\hat{q}(s, a, \underline{w}) - q_{\pi}(s, a)\right)^2\right]$$

- for linear approximations: quadratic in $\underline{w}$, hence convex
- any local minimum is also a global minimum

# Linear Function Approximation cont'd

Linear function approximators subsume tabular methods, by using indicator features:

$$x_i(s) := \mathbb{1}[s = s_i] \qquad s_i \in \mathcal{S} \qquad \text{(for } \hat{v}\text{)}$$

$$x_i(s,a) := \mathbb{1}[(s,a)_i = (s,a)] \qquad (s,a)_i \in \mathcal{S} \times \mathcal{A} \qquad \text{(for } \hat{q}\text{)}$$

Note how the previous two algorithms reduce to tabular methods!

## Monte Carlo

tabular: $V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$

gradient: $\underline{w} \leftarrow \underline{w} + \alpha (G - \hat{v}(S_t, \underline{w})) \nabla_{\underline{w}} \hat{v}(S_t, \underline{w})$

$$\underline{x} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \leftarrow S_t$$

## TD

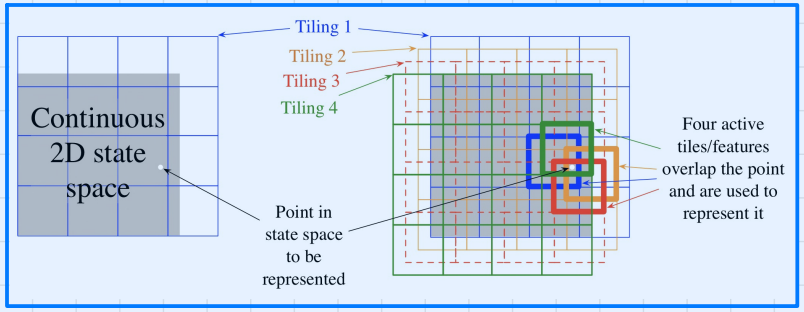
tabular: $V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S') - V(S_t))$

gradient: $\underline{w} \leftarrow \underline{w} + \alpha [R + \gamma \hat{v}(S', \underline{w}) - \hat{v}(S, \underline{w})] \nabla_{\underline{w}} \hat{v}(S, \underline{w})$

# Mountain Car with Linear Function

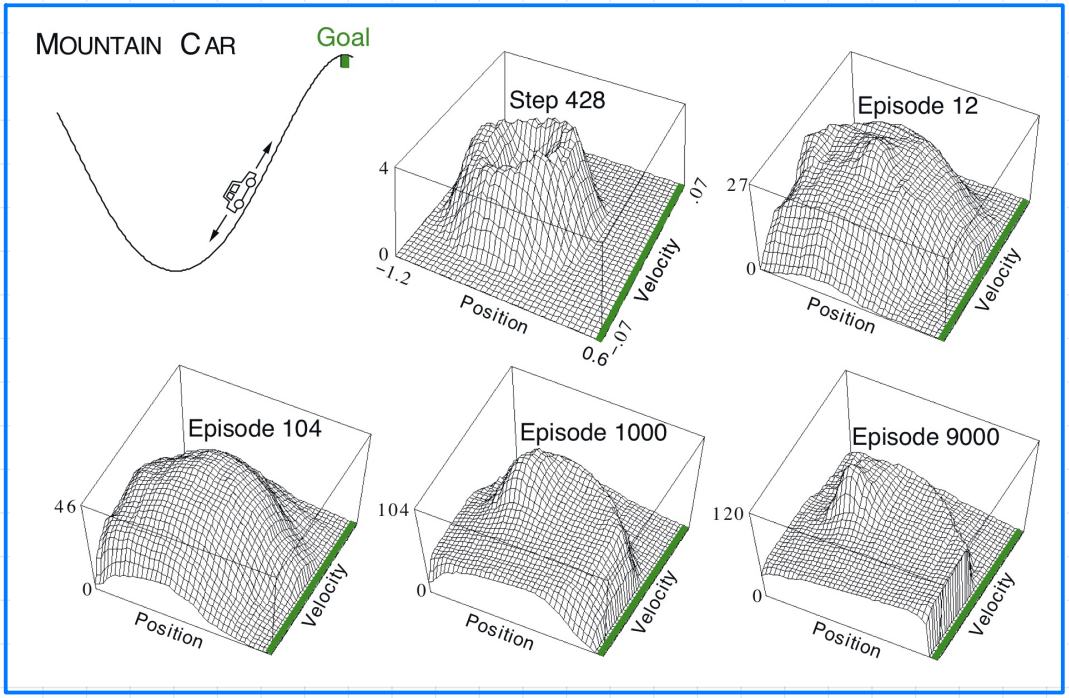
[Sutton & Barto, p. 245]

- car with a too weak engine shall go up a hill
  → needs to gather momentum on the other side
- continuous state space: position $x \in [-1.2, 0.6]$, velocity $\dot{x} \in [-0.7, 0.7]$
- actions $\mathcal{A} = \{-1, 1, 0\}$ (full throttle left/right, no throttle)
- (simplified) physics $\dot{X}_{t+1} = clip[\dot{X}_t + 0.001 A_t - 0.0025 \cos(3 X_t)]$
- features: "tile coding"
- negative value function learned by Sarsa (-1 reward per time step)




Space is discretized into several displaced tilings. For each tiling, we have a vector indicating the containing tile. Overal vector is concatenation of these vectors.

16x4 = 64 binary features



# TD(λ) and Sarsa(λ) Function Approximation

initialize $\underline{w}$

repeat // for each episode

$\underline{e} \leftarrow \underline{0}$ // vector of $d$ zeros

initialize $S$

$A \sim \hat{\pi}(a|S)$ [$\varepsilon$-greedy($q$)]

repeat

take action $A$, observe $R, S'$

$A' \sim \hat{\pi}(a|S')$ [$\varepsilon$-greedy($q$)]

$\underline{e} \leftarrow \gamma\lambda\underline{e} + \nabla_{\underline{w}}\, \hat{v}(S, \underline{w})$ — note that this is consistent with tabular case

[$\underline{e} \leftarrow \gamma\lambda\underline{e} + \nabla_{\underline{w}}\, \hat{q}(S, A, \underline{w})$]

$\delta \leftarrow R + \gamma\, \hat{v}(S', \underline{w}) - \hat{v}(S, \underline{w})$

[$\delta \leftarrow R + \gamma\, \hat{q}(S', A', \underline{w}) - \hat{q}(S, A, \underline{w})$]

$\underline{w} \leftarrow \underline{w} + \alpha\delta\underline{e}$

$S, A \leftarrow S', A'$

(comand alternatives in color are for control, i.e. Sarsa(λ))

# Q-Learning with Function Approximation

initialize $\underline{w}$

repeat // for each episode

  initialize $S$

  repeat

    select $A \sim \hat{\pi}(a|S)$ [$\varepsilon$-greedy($q$)]

    take action $A$, observe $R, S'$

    if $S'$ is terminal:

      $\underline{w} \leftarrow \underline{w} + \alpha[R - \hat{q}(S, A, \underline{w})] \nabla_{\underline{w}} \hat{q}(S, A, \underline{w})$

      break // start next episode

    $\underline{w} \leftarrow \underline{w} + \alpha[R + \gamma \max_{a'} \hat{q}(S', a', \underline{w}) - \hat{q}(S, A, \underline{w})] \nabla_{\underline{w}} \hat{q}(S, A, \underline{w})$

    $S \leftarrow S'$

... unstable algorithm

# Convergence of Algorithms

## Prediction

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

## Control

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

(✓) = chatters around near-optimal value function

* not covered here

Tables: D. Silver

# Deep Q-Learning

**Playing Atari with Deep Reinforcement Learning**

arXiv:1312.5602v1 [cs.LG] 19 Dec 2013

nature

LETTER

doi:10.1038/nature14236

## Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

# Deep Q-Learning

## Main Take-aways

- end-to-end Q-learning, involving a ConvNet
- critical component: experience replay on buffered states
- partially-observed MDP: stack last 4 frames
- close-to or super-human play on many Atari games
- however, long-term and symbolic dependencies are still to be learned ("Montezuma's revenge")