Reinforcement Learning

Lecture 6
Value Function Approximation

Stergios Christodoulidis

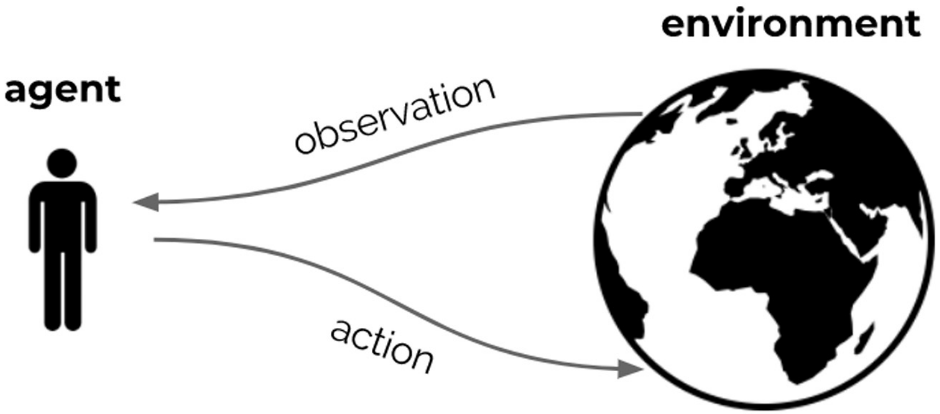
MICS Laboratory
CentraleSupélec
Université Paris-Saclay

https://stergioc.github.io/

CentraleSupélec

Last Lectures

The Agent and the Environment



- At each step t the agent:
    - Receives observation O_t (and reward R_t)
    - Executes action A_t
- The environment:
    - Receives action A_t
    - Emits observation O_{t+1} (and reward R_{t+1})

[Hado van Hasselt, 2021]

Markov Decision Process

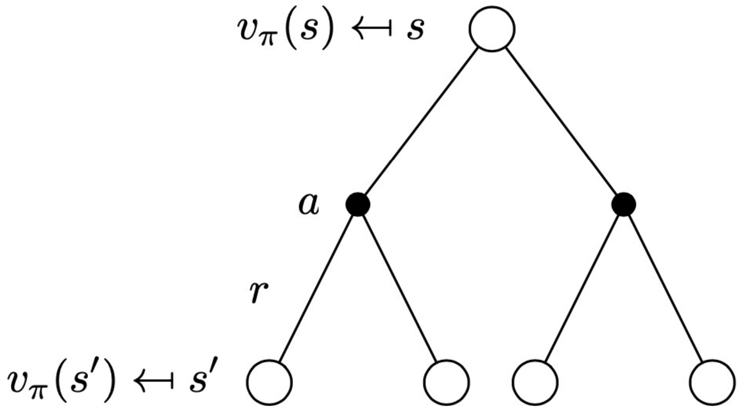
- A Markov decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov

Definition

A Markov Process (or Markov Chain) is a 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{A}$ is a (finite) set of actions
- $\mathcal{P}$ is a state transition matrix, i.e., $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s,\ A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s,\ A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

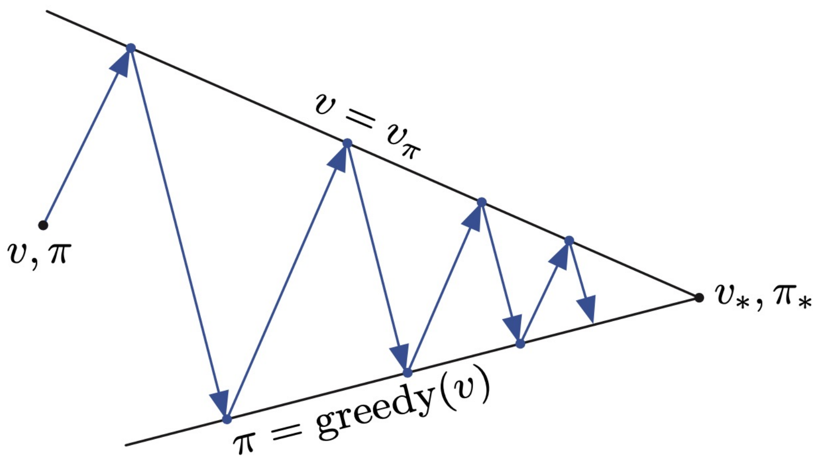
Bellman Equation and Policy Evaluation

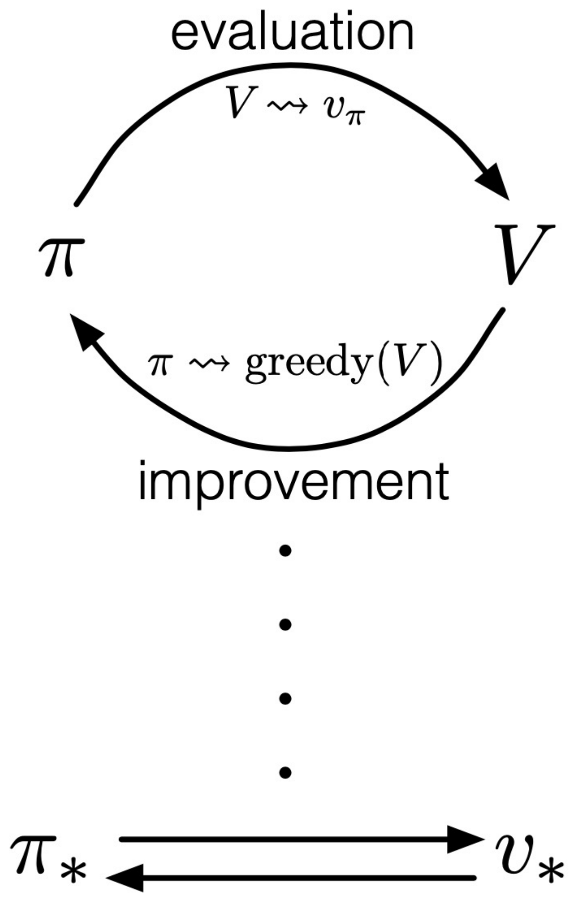


$$\mathrm{v}_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \mathrm{v}_k(s') \right)$$

[An Introduction to Reinforcement Learning, Sutton and Barto]

Generalized Policy Iteration





- Policy evaluation Estimate v_π
  - Any policy evaluation algorithm
- Policy improvement Generate $\pi' \geq \pi$
  - Any policy improvement algorithm

[An Introduction to Reinforcement Learning, Sutton and Barto]

MC and TD

- Goal: learn v_π from episodes of experience under policy π
- Incremental Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

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

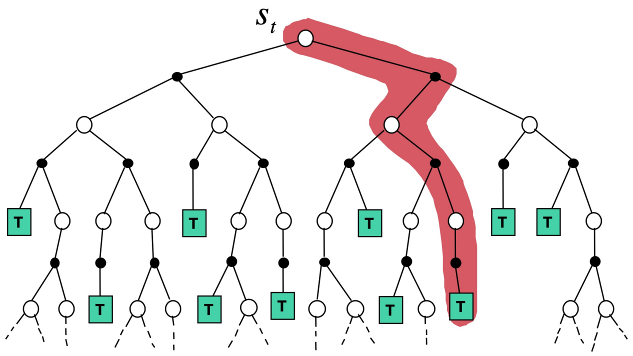
- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

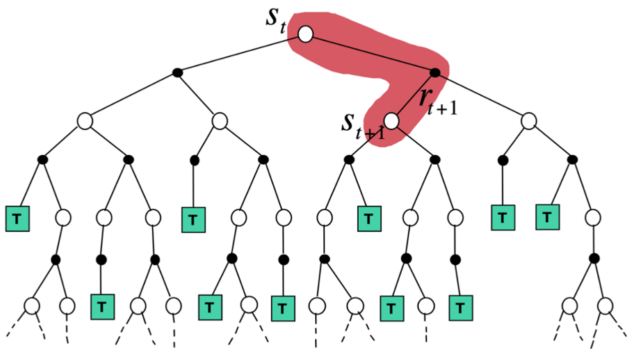
  - $R_{t+1} + \gamma V(S_{t+1})$ is called TD target
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

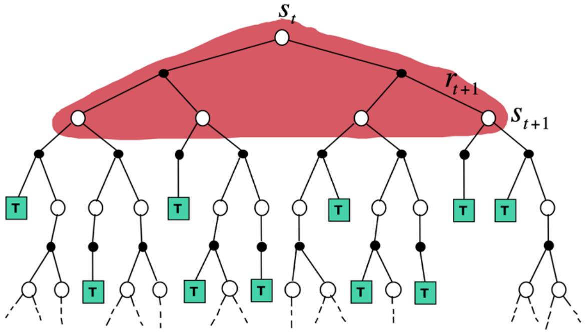
Monte-Carlo Update

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



$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

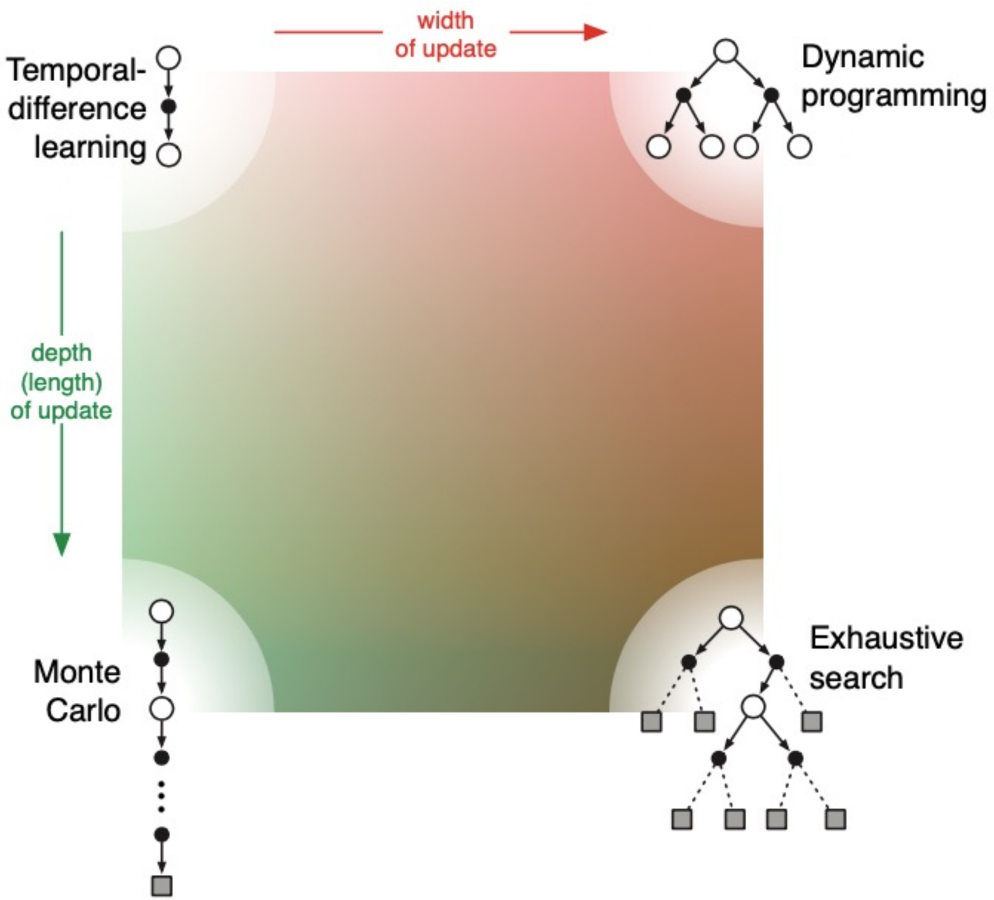




$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$

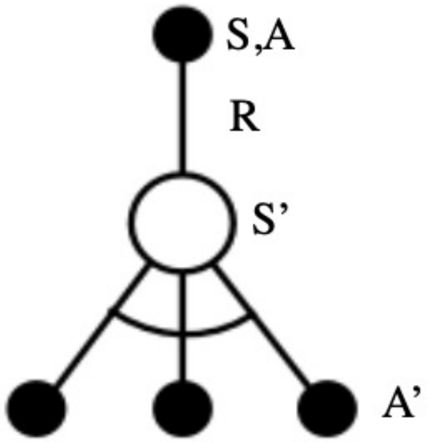
[David Silver, IRL, UCL 2015]

Unified View of Reinforcement Learning



[An Introduction to Reinforcement Learning, Sutton and Barto]

Q-Learning Control Algorithm



$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R_{t+1} + \gamma \max_{a'} Q(S', a') - Q(S, A)\right)$$

[David Silver, IRL, UCL 2015]

Today’s Lecture

Today’s Lecture

- Motivation
- Value Function Approximation
- Tile Coding
- Batch Methods

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Go: 10^{170}
  - Robots: continuous state space
- How can we scale up the model-free methods for prediction and control from the last two lectures?

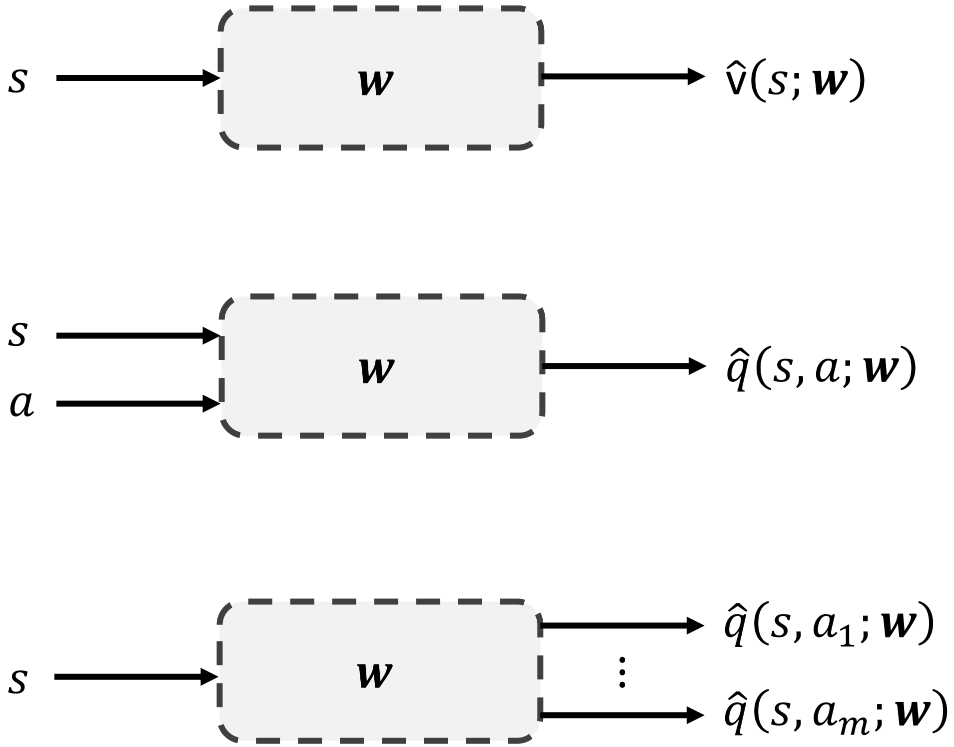
Value Function Approximation

- So far, we have represented value function by a lookup table
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem 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
- Solution for large MDPs:
  - Estimate value function with function approximation

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

  - Generalize from seen states to unseen states
  - Update parameter **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
  - …

Which Function Approximator?

- We will consider differentiable function approximators, e.g.,
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbor
  - Fourier/wavelet bases
  - …

Value Function Approximation

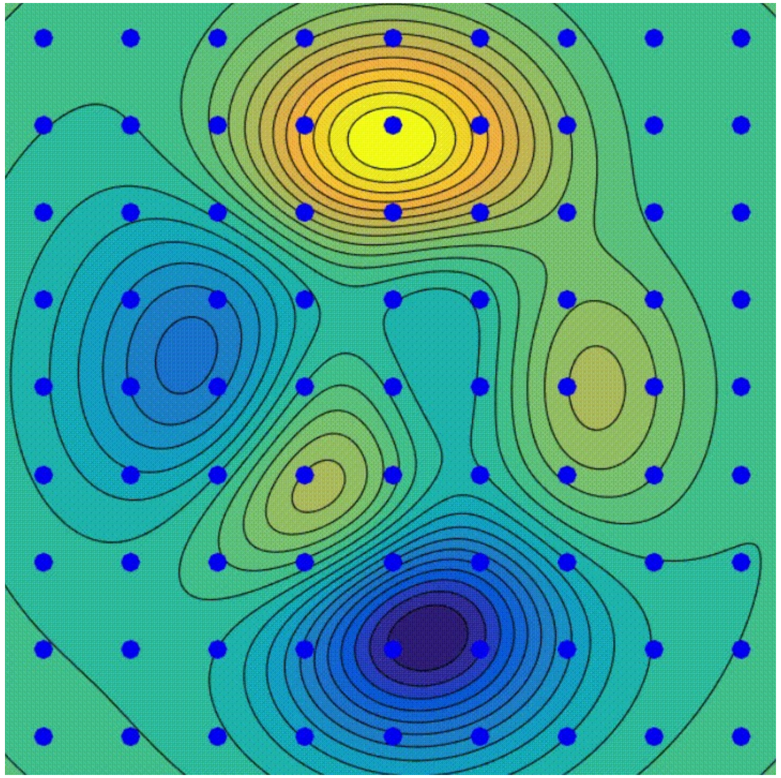
Gradient Descent

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

$$\nabla_w J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{w})}{\partial w_n} \end{pmatrix}$$

- To find a local minimum of $J(\boldsymbol{w})$
- Adjust w in direction of negative gradient

$$\Delta \boldsymbol{w} = -\frac{1}{2} a \nabla_w J(\boldsymbol{w})$$



Value Function Approx. By Stochastic Gradient Descent

- **Goal:** find parameter vector $\boldsymbol{w}$ minimizing mean-squared error between approximate value function $\hat{\mathrm{v}}(s;\boldsymbol{w})$ and true value function $\mathrm{v}_\pi(s)$

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\boldsymbol{w} &= -\frac{1}{2}a\nabla_w J(\boldsymbol{w}) \\ &= \alpha\mathbb{E}_\pi[(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))\nabla_w\hat{\mathrm{v}}(s;\boldsymbol{w})]\end{aligned}$$

- Stochastic gradient descent samples the gradient

$$\Delta\boldsymbol{w} = \alpha(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))\nabla_w\hat{\mathrm{v}}(s;\boldsymbol{w})$$

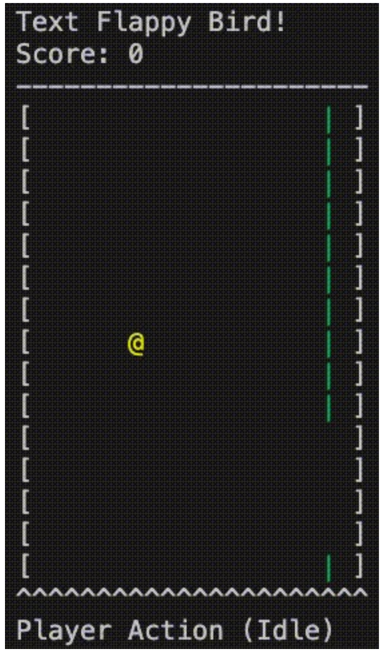
- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a feature vector

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- E.g. Polynomials, Fourier Basis
- 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

$$\hat{\mathrm{v}}(s;\boldsymbol{w}) = x(s)^T\boldsymbol{w} = \sum_{i=0}^{n} x_i(s)w_i$$

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

$$J(w) = \mathbb{E}_\pi[(\mathrm{v}_\pi(s) - x(s)^T\boldsymbol{w})^2]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_w\hat{\mathrm{v}}(s;\boldsymbol{w}) = x(s)$$
$$\Delta\boldsymbol{w} = \alpha\big(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w})\big)x(s)$$

- Update = step-size × prediction error × feature value

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using table lookup features

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

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

$$\hat{\mathrm{v}}(s;\boldsymbol{w}) = \begin{pmatrix} \mathbf{1}(s=s_1) \\ \vdots \\ \mathbf{1}(s=s_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Incremental Prediction Algorithms

- Have assumed true value $\mathrm{v}_\pi(s)$ function given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for $\mathrm{v}_\pi(s)$
  - For MC, the target is the return G_t

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

  - For TD(0), the target is the TD target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{\mathrm{v}}(s_{t+1}; \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha\big(R_{t+1} + \gamma\, \hat{\mathrm{v}}(S_{t+1}; \boldsymbol{w}) - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t; \boldsymbol{w})$$

  - For TD(n), the target is the TD target $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^n \hat{\mathrm{v}}(s_{t+n}; \boldsymbol{w})$

Monte-Carlo with Value Function Approximation (1/2)

- Return G_t is an unbiased, noisy sample of true value $\mathrm{v}_\pi(s_t)$
- Can therefore apply supervised learning to “training data”:

$$< S_1, G_1 >, < S_2, G_2 >, \dots, < S_T, G_T >$$

- For example, using linear Monte-Carlo policy evaluation

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha\big(G_t - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t; \boldsymbol{w}) \\ &= \alpha\big(G_t - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\boldsymbol{x}(S_t)\end{aligned}$$

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

Monte-Carlo with Value Function Approximation (2/2)

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

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

[An Introduction to Reinforcement Learning, Sutton and Barto]

TD Learning with Value Function Approximation (1/2)

- The TD-target $G_{t:t+1} = \; R_{t+1} + \gamma\, \hat{\mathrm{v}}(s_{t+1}; \boldsymbol{w})$ is a biased sample of true value $\mathrm{v}_\pi(s_t)$
- Can still apply supervised learning to "training data":

$$< S_1, R_1 + \gamma\, \hat{\mathrm{v}}(S_2; \boldsymbol{w}) >, < S_2, R_3 + \gamma\, \hat{\mathrm{v}}(S_3; \boldsymbol{w}) >, \dots, < S_{T-1}, R_T >$$

- For example, using linear TD(0)

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha\big(R + \gamma\, \hat{\mathrm{v}}(S'; \boldsymbol{w}) - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t; \boldsymbol{w}) \\ &= \alpha \delta \boldsymbol{x}(S) \end{aligned}$$

- (Semi-Gradient) We do not consider the effect of changing $\boldsymbol{w}$ on the target
- Linear TD(0) converges (close) to global optimum

TD Learning with Value Function Approximation (2/2)

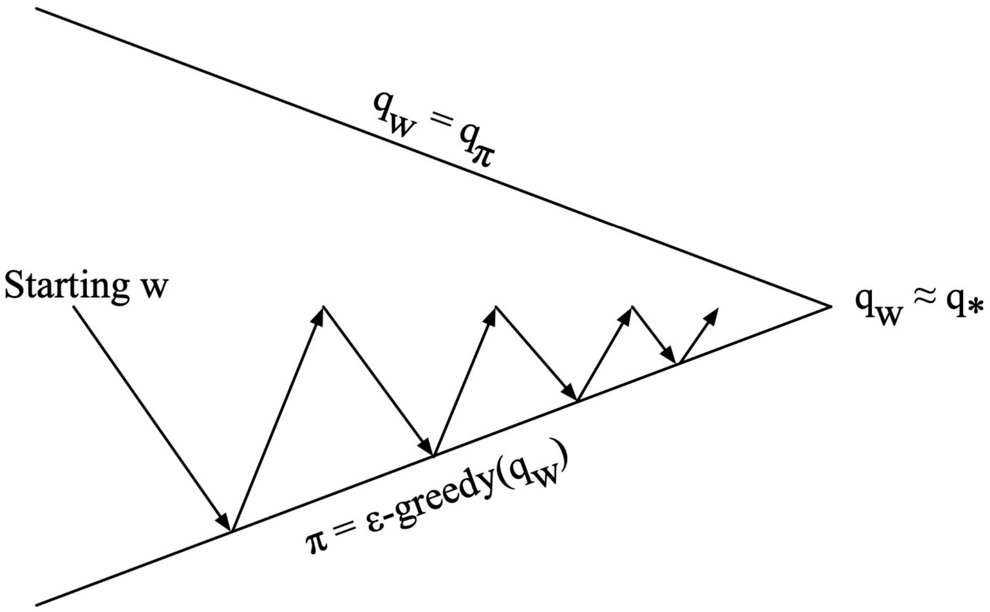
Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

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

[An Introduction to Reinforcement Learning, Sutton and Barto]

Control with Value Function Approximation



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

[David Silver, IRL, UCL 2015]

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A; \boldsymbol{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value function $\hat{q}(S, A, \boldsymbol{w})$ and true action-value function $q_\pi(S, A)$

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

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_w J(\boldsymbol{w}) = \big(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\big)\nabla_w \hat{q}(S, A; \boldsymbol{w})$$
$$\Delta\mathbf{w} = \alpha\big(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\big)\nabla_w \hat{q}(S, A; \boldsymbol{w})$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

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

- Represent action-value function by linear combination of features

$$\hat{q}(S,A,\boldsymbol{w}) = x(S,A)^T \boldsymbol{w} = \sum_{i=0}^{n} x_i(S,A) w_i$$

- Stochastic gradient descent update

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

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S, A)$
  - For MC, the target is the return G_t

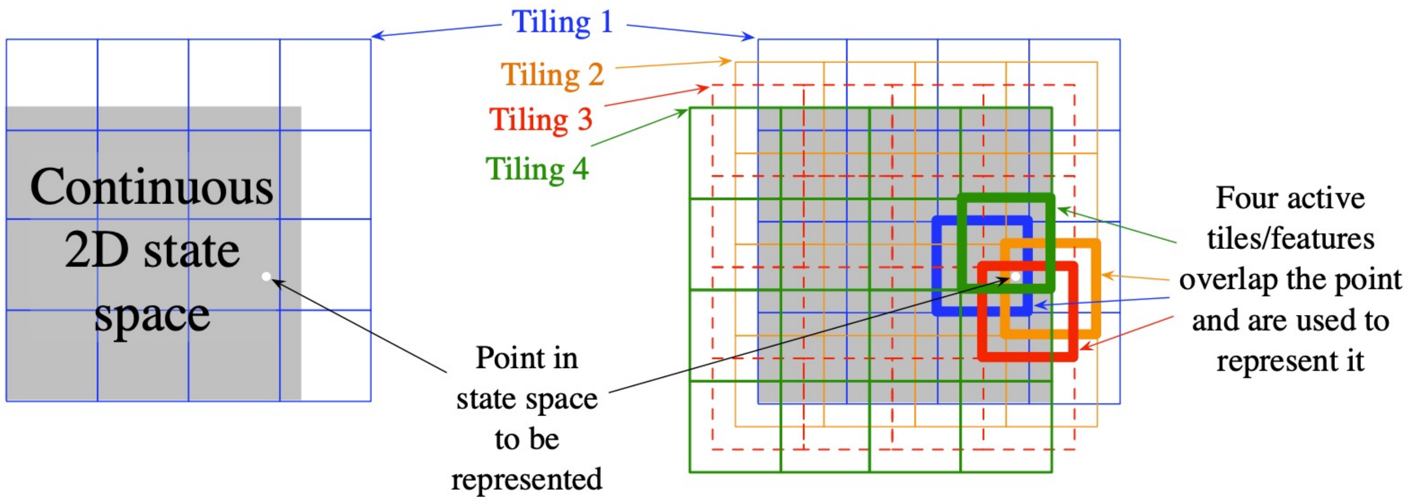
$$\Delta \boldsymbol{w} = \alpha\big(G_t - \hat{q}(S_t, A_t; \boldsymbol{w})\big)\nabla_w \hat{q}(S_t A_t; \boldsymbol{w})$$

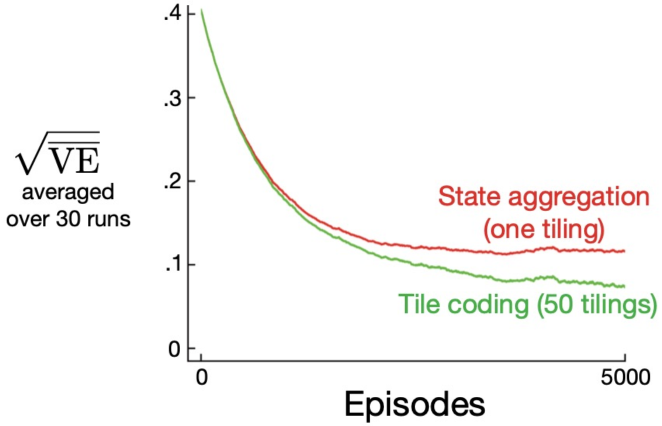
  - For TD(0), the target is the TD target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{\mathrm{v}}(s_{t+1}; \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha\big(R_{t+1} + \gamma\, \hat{\mathrm{v}}(S_{t+1}; \boldsymbol{w}) - \hat{q}(S_t, A_t; \boldsymbol{w})\big)\nabla_w \hat{q}(S_t A_t; \boldsymbol{w})$$

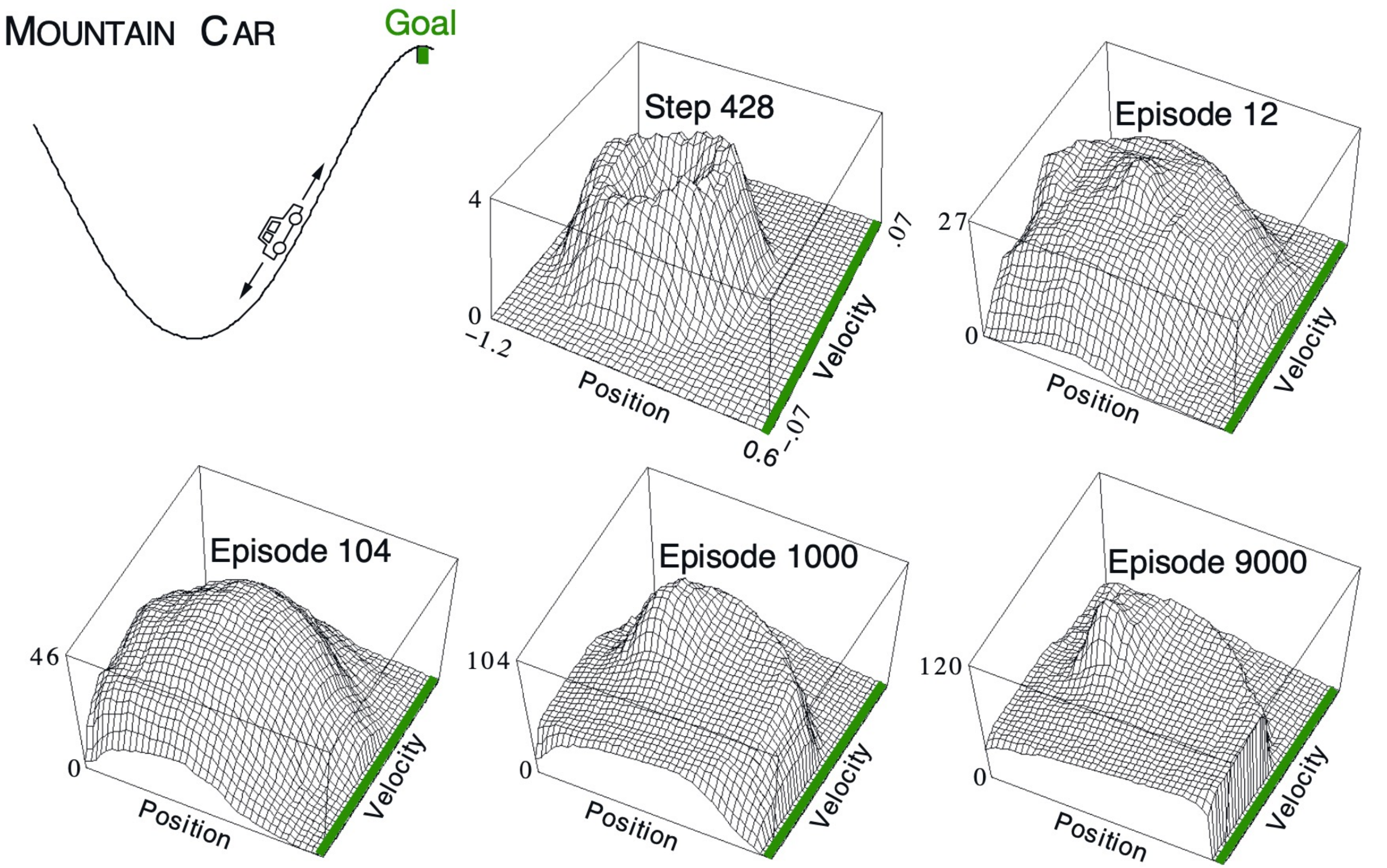
Tile Coding

- If the state is inside a tile, then the corresponding feature has the value 1
- These tilings are offset from one another by a uniform amount in each dimension.
- Overlapping tiles can provide more granularity





Semi-Gradient n-step* Sarsa for Mountain Car



[An Introduction to Reinforcement Learning, Sutton and Barto]

Convergence of Prediction Algorithms

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

Gradient Temporal-Difference Learning for Prediction

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

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

Convergence of Control Algorithms

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

(✓) = oscillates to a near-optimal value function

Batch Methods

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is not sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent’s experience (“training data”)

Least Squares Prediction

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

$$\mathcal{D} = \left\{\langle s_1, \mathrm{v}_1^\pi \rangle, \left\langle\langle s_2, \mathrm{v}_2^\pi \rangle\right\rangle, \dots, \left\langle\langle s_\mathrm{T}, \mathrm{v}_\mathrm{T}^\pi \rangle\right\rangle\right\}$$

- Which parameters **w** give the best fitting value function $\hat{\mathrm{v}}(s;\boldsymbol{w})$
- Least squares algorithms find parameter vector w minimizing sum-squared error between $\hat{\mathrm{v}}(s;\boldsymbol{w})$ and target values $\mathrm{v}_\mathrm{t}^\pi$

$$\begin{aligned} LS(\boldsymbol{w}) &= \sum_{t=0}^{T} (\mathrm{v}_1^\pi - \hat{\mathrm{v}}(s;\boldsymbol{w}))^2 \\ &= \mathbb{E}_\mathcal{D}[(\mathrm{v}_1^\pi - \hat{\mathrm{v}}(s;\boldsymbol{w}))^2] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

- Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, \mathrm{v}_1^\pi \rangle, \langle\langle s_2, \mathrm{v}_2^\pi \rangle\rangle, \dots, \langle\langle s_\mathrm{T}, \mathrm{v}_\mathrm{T}^\pi \rangle\rangle\}$$

- Repeat:
  1. Sample state, value from experience

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

  2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s; \boldsymbol{w}))\nabla_w \hat{\mathrm{v}}(s; \boldsymbol{w})$$

- Converges to least squares solution

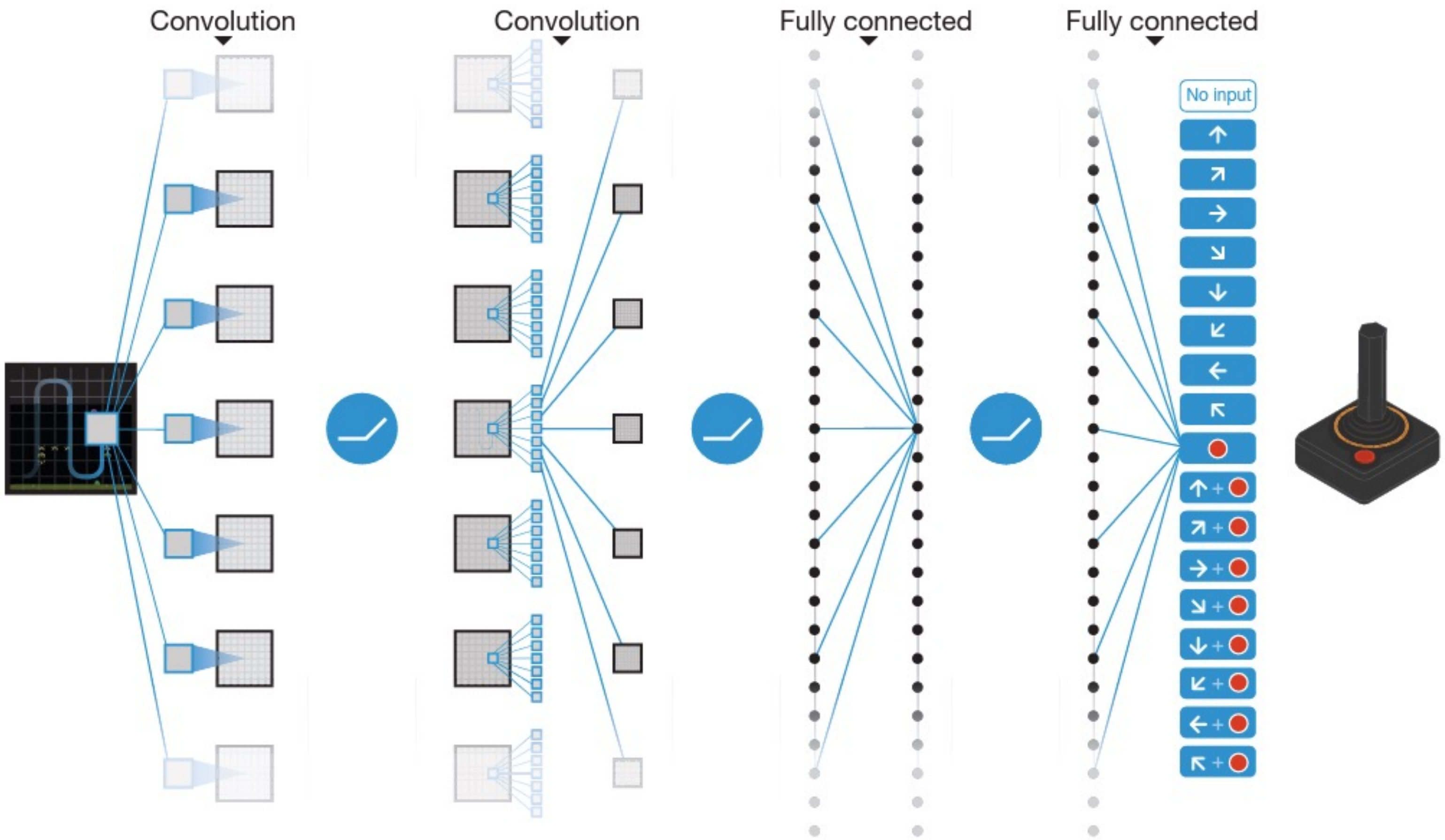
$$\boldsymbol{w}^\pi = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, LS(\boldsymbol{w})$$

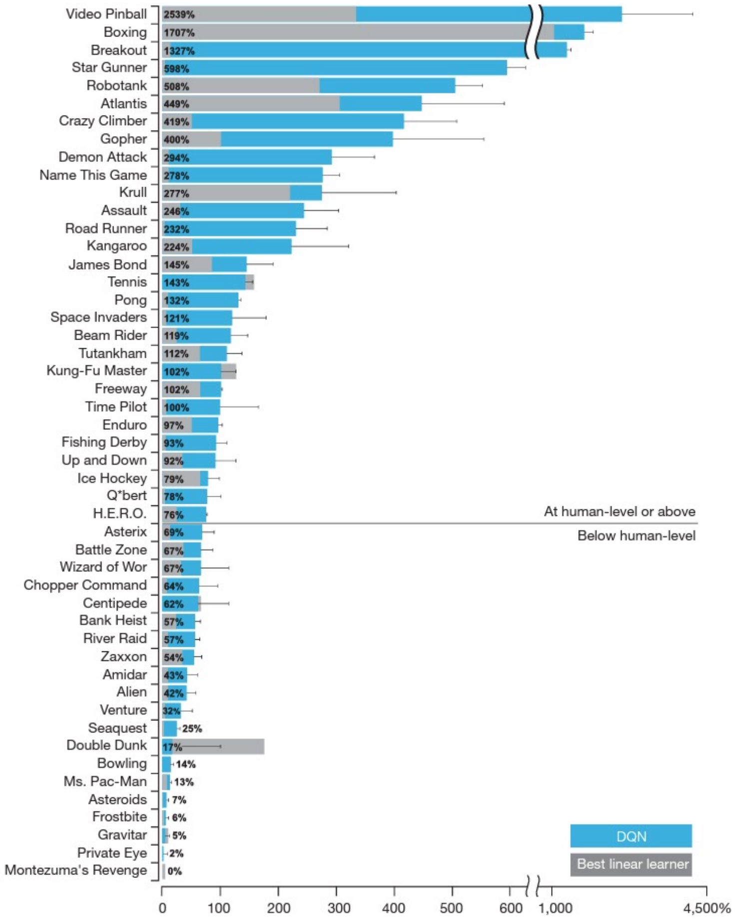
Experience Replay in Deep Q-Networks (DQN)

- Example of DQN that uses experience replay and fixed Q-targets
  - Take action a_t according to ε-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. the parameters of a DQN w^-
  - Optimize MSE between Q-network and Q-learning targets

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

DQN in Atari





[doi:10.1038/nature14236]

Summary

- We discussed how we can extend the tabular methods for building a value function using value function approximators
- We showed how to use gradient descent in order to fit the parameters of the approximators
- We showed how to apply these under the schemes of MC and TD.
- We discussed on the different convergence properties of the different algorithms.
- We discussed about batch methods and tile coding.