ST449 Artificial Intelligence and Deep Learning

Lecture 9

Generalization and function approximation

Milan Vojnovic

https://github.com/lse-st449/lectures

Topics of this lecture

- Introduction to value function approximation
- Gradient-descent methods
- Linear methods
- Control with function approximation
- DQN example

Introduction

Large reinforcement learning problems

- So far we studied reinforcement learning methods using tabular representation
  - Value function represented by a table
  - Ex each state s has an entry for value $V(s)$
  - Ex each state-action pair s, a has an entry for action value $Q(s, a)$
- Limitations of the tabular approach:
  - Scalability: computation time and storage needed to maintain estimates
  - Slow learning: learning the value of each state individually

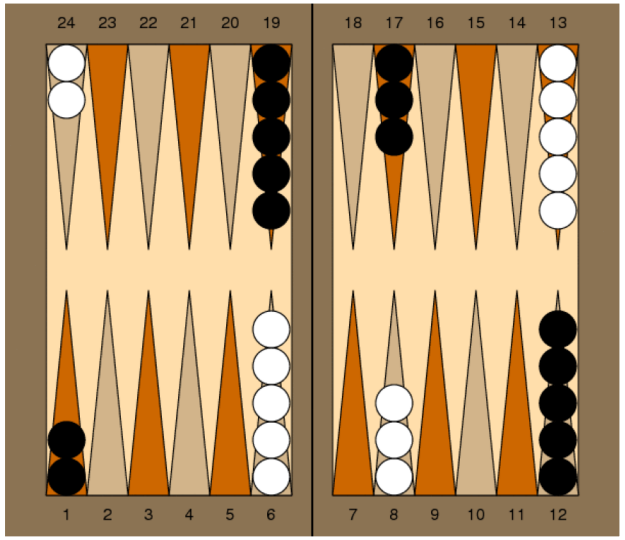
Large reinforcement learning problems (cont'd)

- Large problem examples:
  - Backgammon: $\mathbf{10^{20}}$ states
  - Computer Go: $\mathbf{10^{170}}$ states
  - Continuous state space

- Problem: how can we scale up the model-free reinforcement learning methods for prediction and control to large problems?

- Solution: use generalization to approximate function values from experience, where experience covers only a subset of the state space

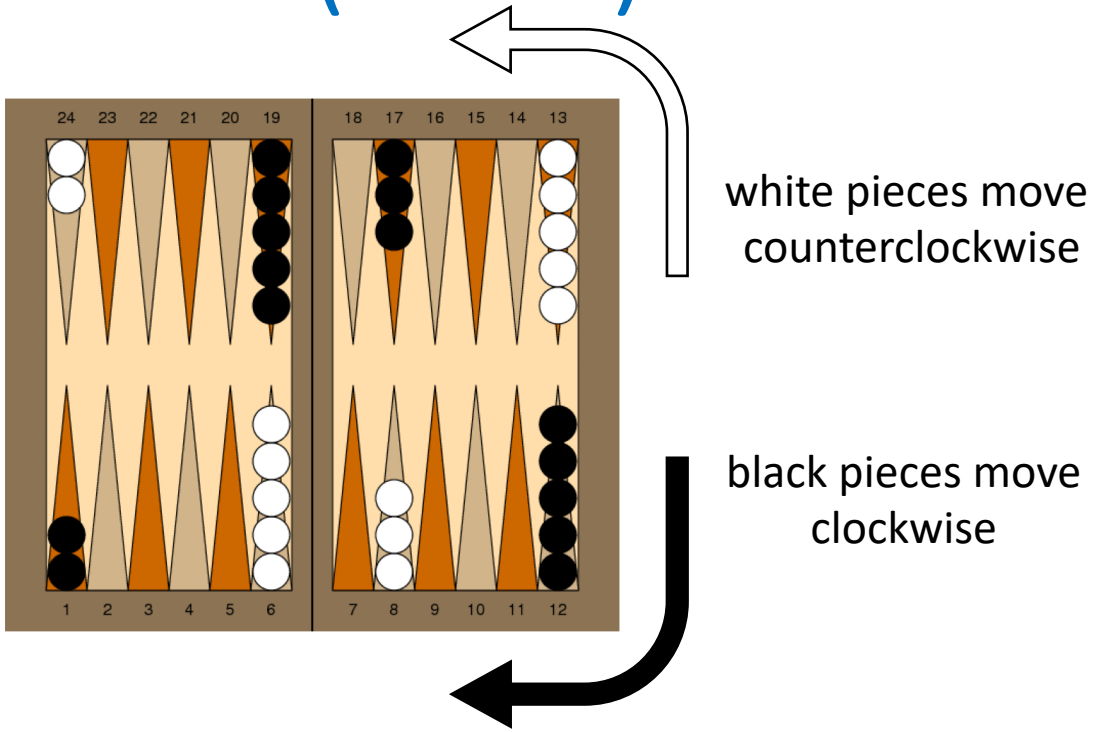
Example: backgammon



An early solution by reinforcement learning using function approximation:

- Teasuro (1992), Practical issues in temporal difference learning, Machine Learning, 8(3-4):257-277
- Teasuro (1994), TD-Gammon, a self-teaching backgammon program, achieves master-level play, Neural Computation, 6(2), 215-219
- Teasuro (1995), Temporal difference learning and TD-Gammon, Communications of the ACM, 38(3),58-68
- Teasuro (2002), Programming backgammon using self-teaching neural nets, Artificial Intelligence, 134(3):58-68

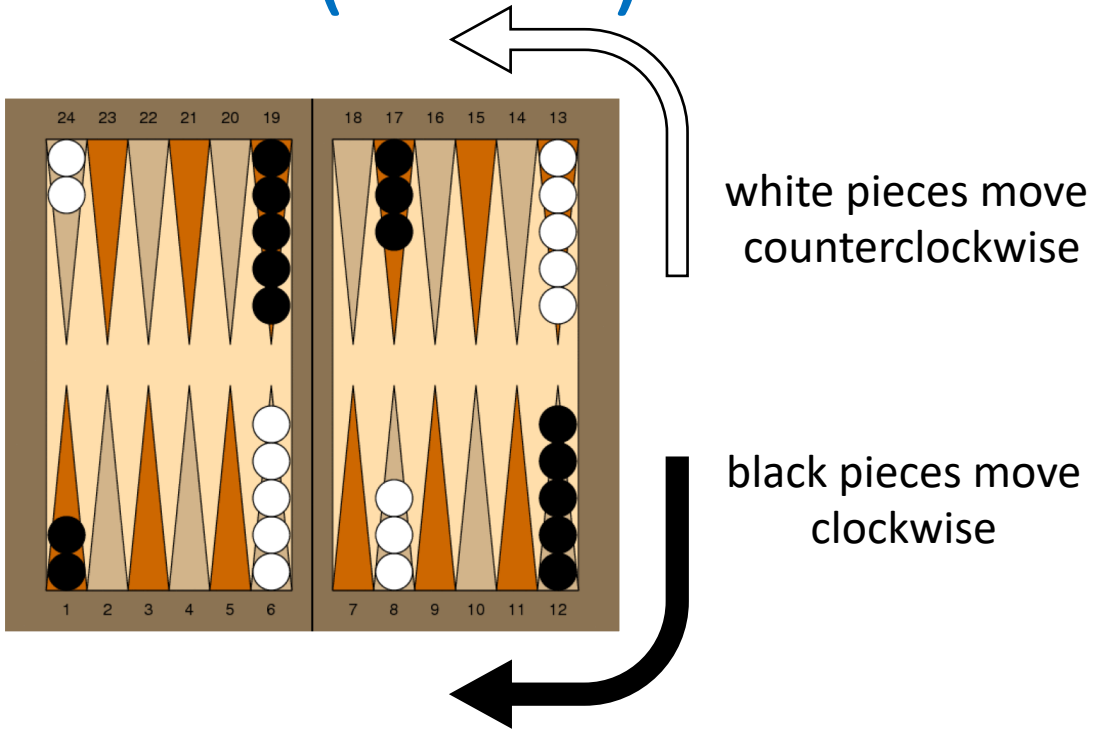
Example: backgammon (cont'd)



- 15 white and 15 black **pieces**
- 24 locations (**points**)
- Two players taking turns
- In each turn a player rolls dice: if the outcome is (x,y) she can move one of the pieces x steps and one (possibly the same piece) y steps

- For each player, the goal is to advance all of her pieces to the last quadrant and then off the board
- The first player to remove all her pieces wins

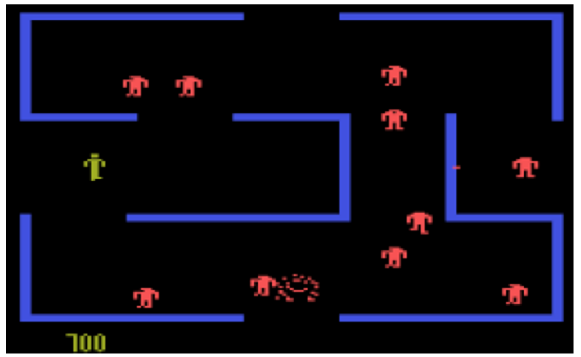
Example: backgammon (cont'd)



Some basic rules:

- If the player moves a piece to a point occupied by a single piece of the opponent, the opponent's piece is put **on the bar**, from which it reenters the game from the start
- If a player has two or more pieces on a point, then the opponent cannot move to that point

Example: Atari 2600 games

...

- Atari 2600 home video game console sold by Atari Inc. from 1977 to 1992
- Some Atari games: Pong, Breakout, Space Invaders, and Asteroids
- Bellemare et al (2015): The Arcade Learning Environment (ALE)
  - Developed and made publicly available the ALE to use Atari 2600 games for studying learning and planning algorithms

Back to generalization and function approximation

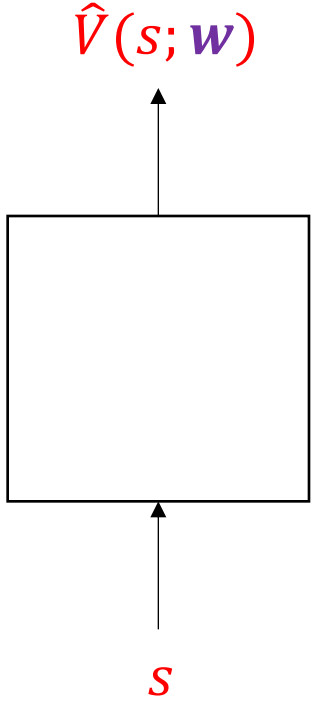
- Function approximation: special type of generalization
  - Estimate a value function using a parametric approximator function
  - Ex. $\hat{V}(s; \boldsymbol{w})$ as approximator for value function $V^{\pi}(s)$
  - Ex. $\hat{Q}(s, a; \boldsymbol{w})$ as approximator for action-value function $Q^{\pi}(s, a)$

- Dimensionality of $\boldsymbol{w}$ much smaller than the state space size

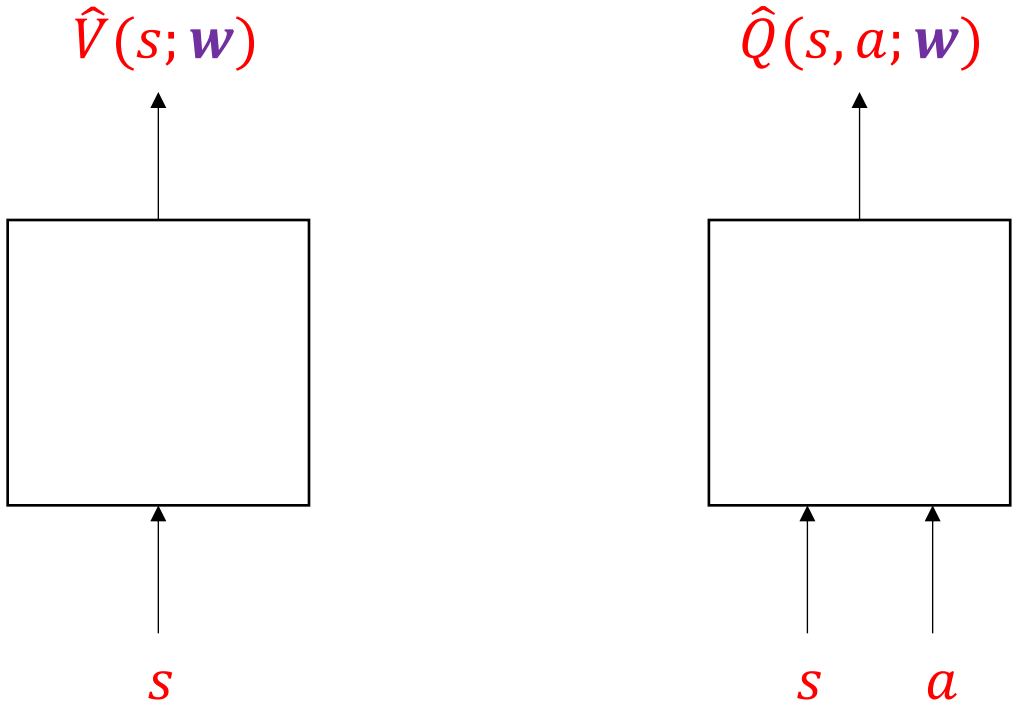
- Update parameter $\boldsymbol{w}$ to find a good approximation using a learning method
  - Ex. MC or TD learning method

- Generalization by function approximation studied in supervised learning
  - Integrate known methods with reinforcement learning !

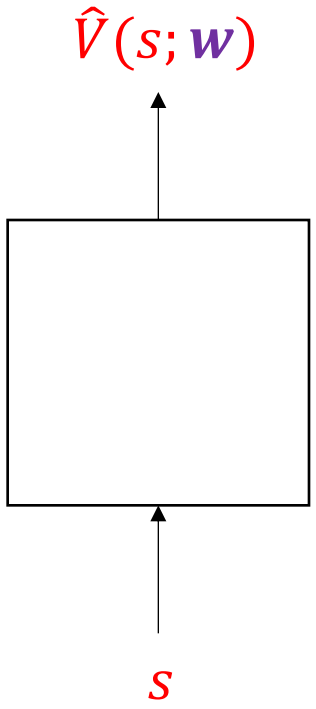
Types of value function approximators

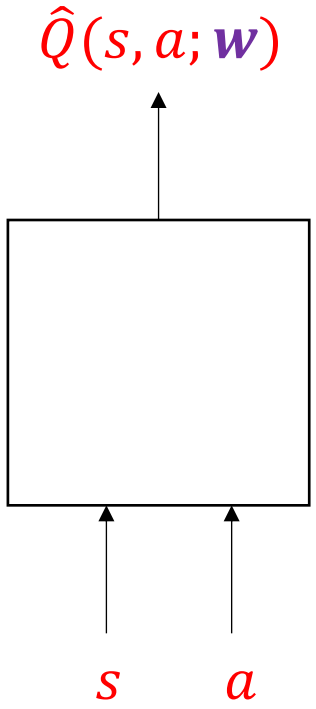


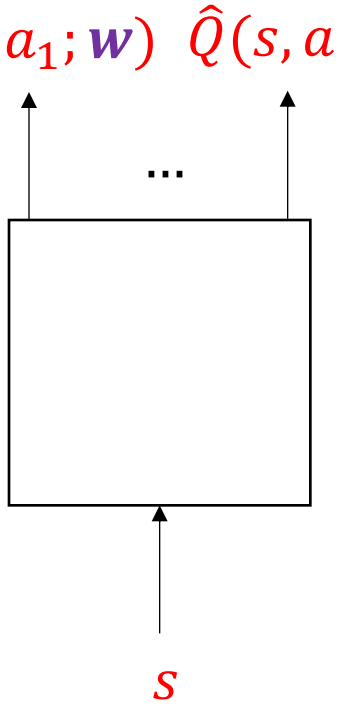
Types of value function approximators



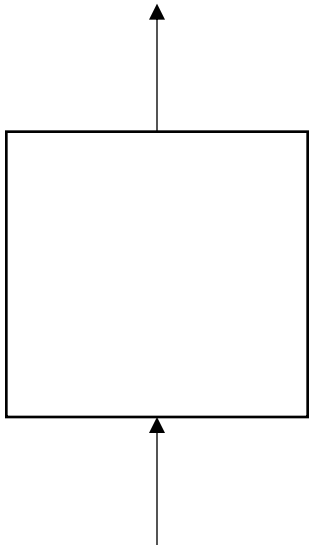
Types of value function approximators







Types of value function approximators (cont'd)



- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbor
- …

} Our focus

Gradient-descent based methods

Function approximation by reinforcement learning

- Observations are individual backups:
  (state being backed up) $s \mapsto v$ (backed up value)

- Example backups:
  - DP: $s \mapsto \mathbf{E}_\pi[r_{t+1} + \gamma V_t(s_{t+1}) \mid s_t = s]$
  - MC: $s_t \mapsto R_t$
  - TD(0): $s_t \mapsto r_{t+1} + \gamma V_t(s_{t+1})$
  - TD(λ): $s_t \mapsto R_t^\lambda$

- Similar to supervised learning: feature vector ↦ label

- Unique characteristics of reinforcement learning: online learning, nonstationary target functions (value functions change as policy changes)

How to chose weight parameters?

- Goal: find a parameter $\boldsymbol{w}$ value that minimizes a given error function J: W $\rightarrow \mathbf{R}$

- Mean-squared error (common supervised learning objective):

$$J(\boldsymbol{w}) = \frac{1}{2}\mathbf{E}_{s\sim\mu}\left[\left(\hat{V}(s;\boldsymbol{w}) - V^{\pi}(s)\right)^2\right]$$

  where μ is a distribution on the state space (specifies how the error is distributed over different states)

- Commonly assumed for μ: to be the on-policy distribution
  - Distribution of states encountered under policy π
  - Minimizes the error in states that actually occur while following the policy

Gradient-descent methods

- Assume $\hat{V}: S \times \mathbf{R}^n \rightarrow \mathbf{R}$ and $\hat{V}(s; \boldsymbol{w})$ differentiable wrt $\boldsymbol{w}$ for every state s
- Consider first a simple case where the training examples are $s_t \mapsto V^{\pi}(s_t)$
  - Input examples give the exact value of the value at a given state
  - Still a problem because not all states may be visited
- Gradient-descent algorithm: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t)$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \mathbf{E}_{s \sim \mu} \left[\left(V^{\pi}(s) - \hat{V}(s; \boldsymbol{w}_t) \right) \nabla_{\boldsymbol{w}} \hat{V}(s; \boldsymbol{w}_t) \right]$$

- Stochastic gradient descent algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \left(V^{\pi}(s_t) - \hat{V}(s_t; \boldsymbol{w}_t) \right) \nabla_{\boldsymbol{w}} \hat{V}(s_t; \boldsymbol{w}_t)$$

where s_t are independent and identically distributed $s_t \sim \mu$

Gradient-descent methods (cont'd)

- Assume training examples: $s_t \mapsto v_t$ where v_t is a target, some approx of $V^\pi(s_t)$

- Gradient-descent algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \mathbf{E}_{s\sim\mu}\left[\left(v_t - \hat{V}(s;\boldsymbol{w}_t)\right)\nabla_w \hat{V}(s;\boldsymbol{w}_t)\right]$$

  Stochastic gradient descent algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \left(v_t - \hat{V}(s_t;\boldsymbol{w}_t)\right)\nabla_w \hat{V}(s_t;\boldsymbol{w}_t)$$

- Sufficient condition for convergence to a local minimum: standard assumptions on step sizes, and v_t is an unbiased estimate of $V^\pi(s_t)$
  - Condition validated for MC targets
  - Not validated for n-step returns of TD(λ) method for all $\lambda \in [0,1)$

Gradient-descent form of TD(λ)

- Forward view update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \left[R_t^{\lambda} - \hat{V}(s_t; \boldsymbol{w}_t) \right] \nabla_{\boldsymbol{w}} \hat{V}(s_t; \boldsymbol{w}_t)$$

- Backward view implementation:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \delta_t \boldsymbol{e}_t$$

where

TD-error: $\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}; \boldsymbol{w}_t) - \hat{V}(s_t; \boldsymbol{w}_t)$

and

Eligibility traces: $\boldsymbol{e}_t = \gamma \lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{w}} \hat{V}(s_t; \boldsymbol{w}_t)$ and $\boldsymbol{e}_0 = \boldsymbol{0}$

Online gradient-descent TD(λ) prediction

- Initialization: $\boldsymbol{w}$ arbitrary and $\boldsymbol{e} = \boldsymbol{0}$

- **Repeat** for each episode:
  $s \leftarrow$ initial state of episode

  Repeat for each step of episode:
    $a \leftarrow$ action given by π for s
    Take action a, observe reward r and next state s'

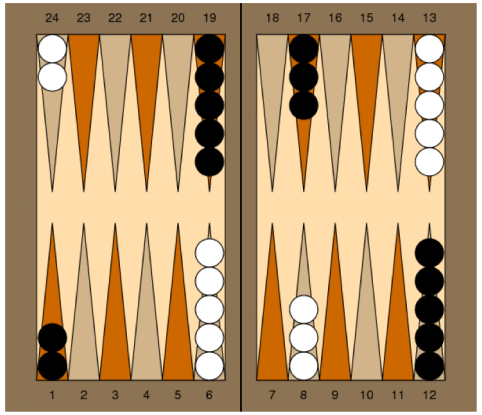
    $\delta \leftarrow r + \gamma \hat{V}(s'; \boldsymbol{w}) - \hat{V}(s; \boldsymbol{w})$
    $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \nabla_{\boldsymbol{w}} \hat{V}(s; \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\delta \boldsymbol{e}$
    $s \leftarrow s'$

    update η
  until s is terminal

What class of functions for f to use ?

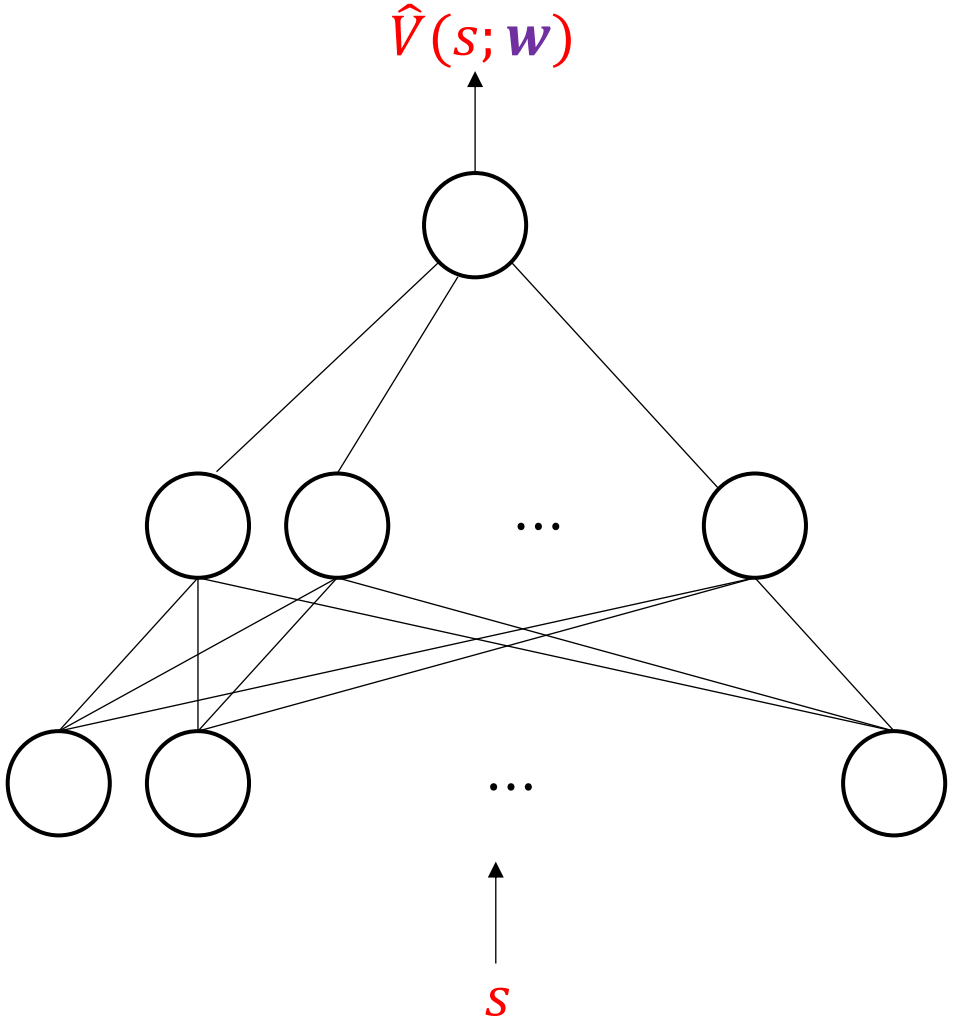
- Two common choices:
  - Linear functions
  - Neural networks (trained by using backpropagation)
- Pros and cons:
  - Linear functions require (hand-crafted) feature engineering
  - Theoretical guarantees differ for the two classes of functions

TD-Gammon example



- TD(λ) learning algorithm
- Neural network value function approximator
  - Multilayer perceptron with one hidden layer
- State represented by 198 input units

TD-Gammon neural network approximator



sigmoid activation functions

hidden units (60-80)

input units (198)

input units represent backgammon position

TD-Gammon 0.0 state representation

- For each point p, and piece color c, 4 units whose values are

Unit values	# of c-color pieces at p	Interpretation
0, 0, 0, 0	0	
1, 0, 0, 0	1	Blot (can be hit)
0, 1, 0, 0	>1	Made point
0, 1, 1, 0	3	Single spare
0, 1, 0, a	>3	

where $a = \frac{1}{2}([\text{number of pieces at point p}]-3)$

- For each piece color c,
  - 1 unit to count how many c-color pieces are on the bar
  - 1 unit = $\frac{1}{15}$[number of c-color pieces successfully removed from the board]
  - 1 unit = 1 if color-c player's turn to move, otherwise 0
- Total number of units: 24 x 4 x 2 + 2 x 3 = 192 + 6 = 198

Atari 2600 Bellemere et al (2015)

- Learning method:
  - SARSA(λ) with linear function approximation and ϵ-greedy
- Feature generation methods:

Features	Description
Basic	Binary encoding of the colors within the Atari 2600 screen
BASS	The basic method augmented with pairwise feature combinations (color encoding restricted to 3 bits)
DISCO	A heuristic object detection trained on offline data
LSH	A method encoding the Atari 2600 screen into a small set of binary features via Locally Sensitive Hashing
RAM	An encoding of the Atari 2600 1024 of memory together with pairwise combinations

Linear function approximation

Linear function approximation

- Each state $s \in S$ is represented by a feature vector $\phi_s = (\phi_{s,1}, \ldots, \phi_{s,k})^\top$
- Linear function approximator:

$$\hat{V}(s; \boldsymbol{w}) = \phi_s^\top \boldsymbol{w} = \sum_{i=1}^{k} \phi_{s,i} w_i$$

- Gradient vector: $\nabla_{\boldsymbol{w}} \hat{V}(s; \boldsymbol{w}) = \phi_s$

- Quadratic objective function:

$$J(\boldsymbol{w}) = \frac{1}{2} \mathbf{E}_{s \sim \mu}[(V^\pi(s) - \phi_s^\top \boldsymbol{w})^2]$$

- Stochastic gradient descent algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \left(V^\pi(s_t) - \hat{V}(s_t; \boldsymbol{w}_t) \right) \phi_{s_t}$$

Tabular representation as a linear approximator

- Each state $s \in S$ represented by one hot-encoding feature vectors

$$\phi_s = e_s$$

  where e_s is a vector of dimension $|S|$ with element s equal to 1 and all other elements equal to 0

- Here the parameter vector $\boldsymbol{w}$ specifies the value of each individual state

Backward-view implementation: linear case

- The parameter vector update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \delta_t \boldsymbol{e}_t$$

where

TD-error: $\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}; \boldsymbol{w}_t) - \hat{V}(s_t; \boldsymbol{w}_t)$

and

Eligibility traces: $\boldsymbol{e}_t = \gamma \lambda \boldsymbol{e}_{t-1} + \phi_{s_t}$ and $\boldsymbol{e}_0 = \boldsymbol{0}$

Convergence theorem [Tsitsiklis and Roy (1997)]

Assumptions:

1. States s_t evolve according to a Markov chain with transition matrix P that is irreducible and aperiodic and has unique stationary distribution μ
2. Instantaneous rewards r_t satisfy $\mathbf{E}_\mu[r_t^2] < \infty$
3. Feature vectors ϕ_s linearly independent and such that $\mathbf{E}_\mu\left[\phi_{s_t,i}^2\right] < \infty$
4. There exists a function $b: S \to \mathbf{R}_+$ such that for all s and $m \geq 0$:

   $$\sum_{t=0}^{\infty} \left\| \mathbf{E}\left[\phi_{s_t}\phi_{s_{t+m}}^\top \middle| s_0 = s\right] - \mathbf{E}_\mu\left[\phi_{s_t}\phi_{s_{t+m}}^\top\right] \right\| \leq b(s)$$

   $$\sum_{t=0}^{\infty} \left\| \mathbf{E}\left[r_{t+m+1}\phi_{s_t} \middle| s_0 = s\right] - \mathbf{E}_\mu\left[r_{t+m+1}\phi_{s_t}\right] \right\| \leq b(s)$$

   $\mathbf{E}[b^q(s_t)|s_0 = s] \leq \kappa_q b^q(s)$, for any $q \geq 1$ for some κ_q and all t
5. Step sizes η_t are positive nonincreasing, $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$

Convergence theorem (cont'd)

Thm:

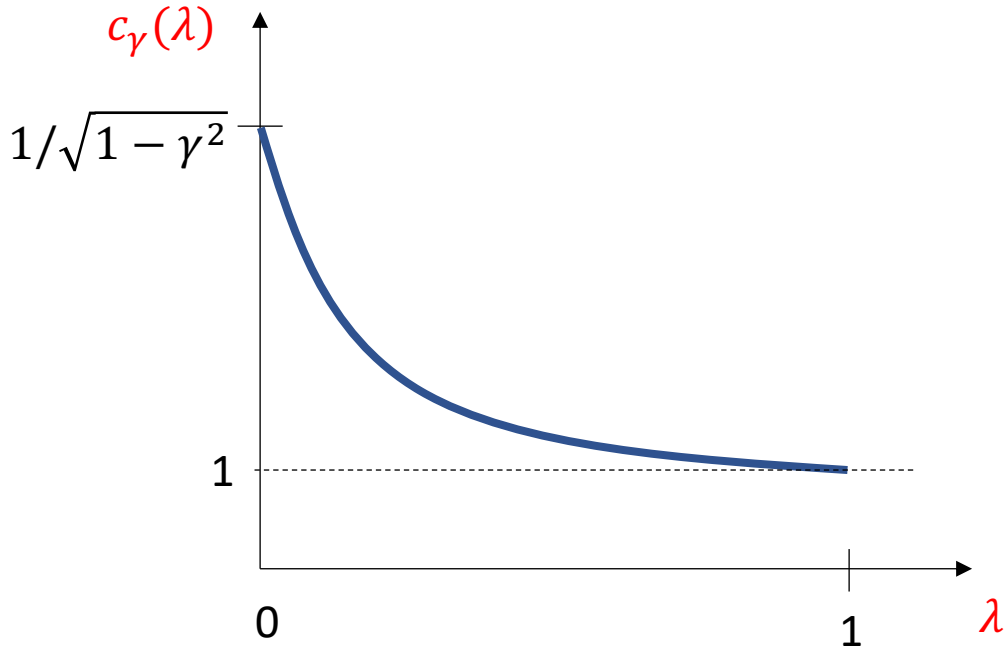
- For every $\lambda \in [0,1]$, the TD(λ) algorithm with linear function approximators converges with probability 1

- The limit parameter vector $\boldsymbol{w}_\infty$ satisfies

$$J(\boldsymbol{w}_\infty) \leq \frac{1-\gamma\lambda}{\sqrt{(1-\gamma)(1+\gamma-2\lambda\gamma)}} J(\boldsymbol{w}^*)$$

$c_\gamma(\lambda)$:= approximation factor

optimal value



Convergence of prediction algorithms

	Algorithm	Tabular	Linear	Non-linear
	MC	✓	✓	✓
On-policy	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
	MC	✓	✓	✓
Off-policy	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Source: Silver, UCL Course on RL, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Convergence of control algorithms

Algorithm	Tabular	Linear	Non-linear
MC	✓	✓	✗
Sarsa	✓	✓	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

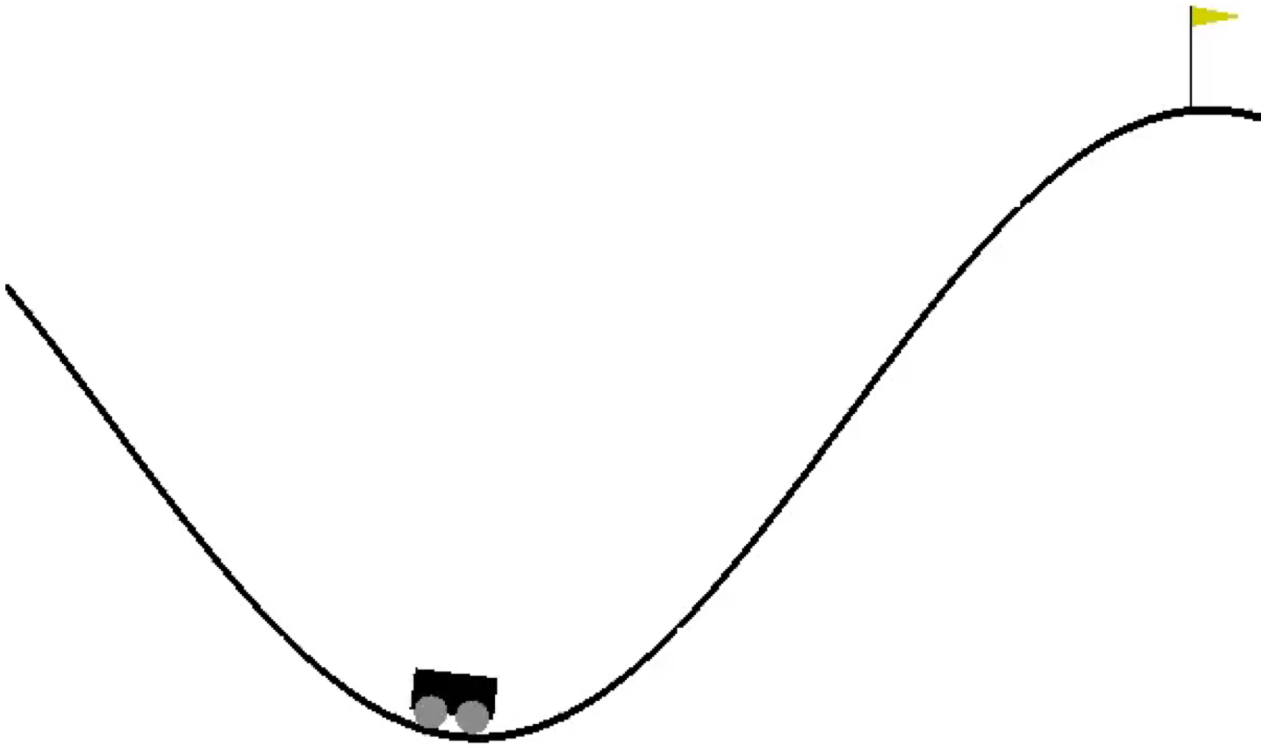
Source: Silver, UCL Course on RL, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Control with function approximation

- An approach based on the general policy iteration principle:
  - $\hat{Q}(s, a; \boldsymbol{w})$ is an approximator of $Q^{\pi}(s, a)$ with parameter vector $\boldsymbol{w}$
  - Input examples: $(s_t, a_t) \mapsto v_t$ where v_t is an estimate of $Q^{\pi}(s_t, a_t)$
  - Gradient descent: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \left(v_t - \hat{Q}(s_t, a_t; \boldsymbol{w}_t) \right) \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$
- Backward-view implementation: SARSA(λ)

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \boldsymbol{e}_t \delta_t$$
$$\boldsymbol{e}_t = \gamma \lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t), \boldsymbol{e}_0 = 0$$

Example: mountain-car task



Seminar exercise: solution by using a linear method

Nonlinear function approximation

Stability issue with nonlinear approximators

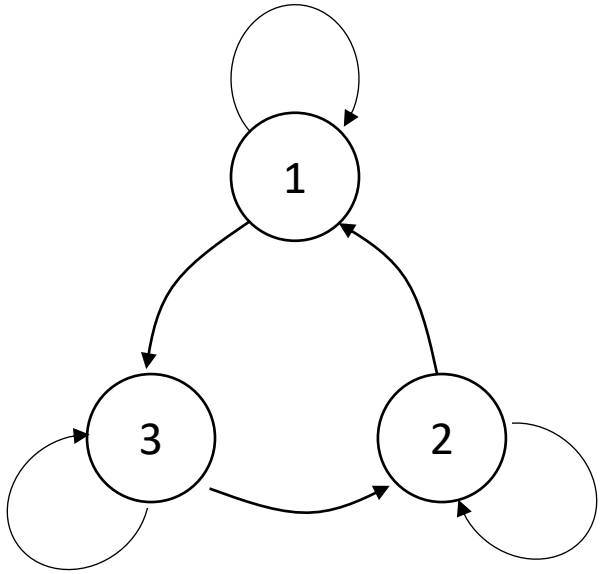
- Unlike to linear methods, temporal difference learning algorithm with nonlinear function approximation may diverge

- Proof by an example: next slides [Tsitsiklis and Roy (1997)]

Bad example definition

- States: Markov chain with the state space $S = \{1,2,3\}$ and the transition matrix

$$\mathbf{P} = \begin{pmatrix} 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \end{pmatrix}$$



- All instantaneous rewards equal to zero
- The optimum value function is $V^*(s) = 0$ for all $s \in S$

Bad example definition (cont'd)

- Nonlinear approximator $\widehat{V}(w) = \left(\hat{V}(s; w), s \in S\right)^{\mathsf{T}}$ for a scalar parameter w:
  - Arbitrary value $\hat{V}(0)$ such that $\hat{V}(0)^{\mathsf{T}}\mathbf{1} = 0$
  - $\frac{d}{dw}\hat{V}(w) = (\mathbf{Q} + \epsilon\mathbf{I})\hat{V}(w)$ for a small constant $\epsilon \geq 0$ and

$$\mathbf{Q} = \begin{pmatrix} 1 & 1/2 & 3/2 \\ 3/2 & 1 & 1/2 \\ 1/2 & 3/2 & 1 \end{pmatrix}$$

Bad example definition (cont'd)

- The TD(0) update equation:

$$w_{t+1} = w_t + \eta_t \left(\gamma \hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t) \right) \frac{\partial}{\partial w} \hat{V}(s_t; w_t)$$

- Mean squared-error objective function:

$$J(w) = \frac{1}{|S|} \left\| \hat{V}(w) \right\|^2 \quad \text{(recall the def of } J\text{)}$$

- We will show that for the given example $J(w_t)$ increases with t (divergence!)

Bad example: the limit mean ODE

- The limit mean ODE:

$$\frac{d}{dt}w = \mathbf{E}\left[\frac{w_{t+1} - w_t}{\eta_t} | w_t = w\right]$$

- In our case:

$$\mathbf{E}\left[\frac{w_{t+1}-w_t}{\eta_t} | w_t = w\right] = \frac{1}{\eta_t}\frac{1}{|S|}\sum_s\left(\gamma \sum_{s'} P_{s,s'}\hat{V}(s';w) - \hat{V}(s;w)\right)\frac{\partial}{\partial w}\hat{V}(s;w)$$

$$= \frac{1}{\eta_t|S|}\left[(\mathbf{Q}+\epsilon\mathbf{I})\hat{V}(w)\right]^\top(\gamma\mathbf{P}-\mathbf{I})\hat{V}(w)$$

Bad example: divergence proof

- The limit mean ODE:

$$\frac{d}{dt}w = \hat{V}(w)^{\top}(\mathbf{Q}^{\top} + \epsilon\mathbf{I})(\gamma\mathbf{P} - \mathbf{I})\hat{V}(w)$$

- For $\epsilon = 0$, we have

$$\begin{aligned}\frac{d}{dt}w &= \hat{V}(w)^{\top}\mathbf{Q}^{\top}(\gamma\mathbf{P} - \mathbf{I})\hat{V}(w) \\ &= \gamma\hat{V}(w)^{\top}\mathbf{Q}^{\top}\mathbf{P}\hat{V}(w) \qquad (\hat{V}(w)^{\top}\mathbf{Q}^{\top}\hat{V}(w) = 0) \\ &= \frac{\gamma}{2}\hat{V}(w)^{\top}\underbrace{(\mathbf{Q}^{\top}\mathbf{P} + \mathbf{Q}\mathbf{P}^{\top})}_{\text{positive definite}}\hat{V}(w)\end{aligned}$$

- $\Rightarrow \quad \frac{d}{dt}w \geq c\left\|\hat{V}(w)\right\|^2$ ($w(t)$ increases with t)
- By continuity the last inequality holds for small ϵ for a suitably defined constant c

Bad example: divergence proof

- Now, note

$$\frac{d}{dw}\left\|\hat{V}(w)\right\|^2 = \hat{V}(w)^{\mathsf{T}}(\mathbf{Q}+\mathbf{Q}^{\mathsf{T}})\hat{V}(w) + 2\epsilon\left\|\hat{V}(w)\right\|^2 \geq 2\epsilon\left\|\hat{V}(w)\right\|^2$$

  which shows that $\left\|\hat{V}(w)\right\|^2$ increases with w

- Since we have shown that $w(t)$ increases with t

$$\Rightarrow \quad \left\|\hat{V}(w(t))\right\|^2 \text{ increases with } t$$

Deep Q-Network (DQN)

- Q-learning like method that uses a neural network action value approximator and several tricks to mitigate instability

- Proposed by Mnih et al (2013, 2015) and showed to yield superior performance to previously known methods for playing Atari 2600 games

- Action value function approximated by a convolutional neural network
  - Used the same network architecture for all games !

- Additional tricks: experience replay, separation of timescales, clipping errors

Atari 2600 observed state

- Atari 2600 screen
- 210 x 160 pixel image frames
- 129 colors
- 60Hz frame rate

- Atari 2600 screen: partially observed state of the game

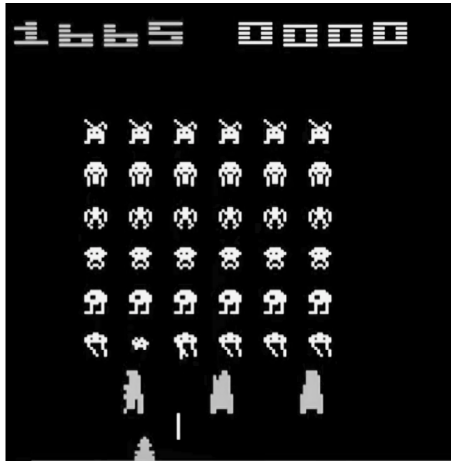
Input preprocessing

160

210

ϕ

84

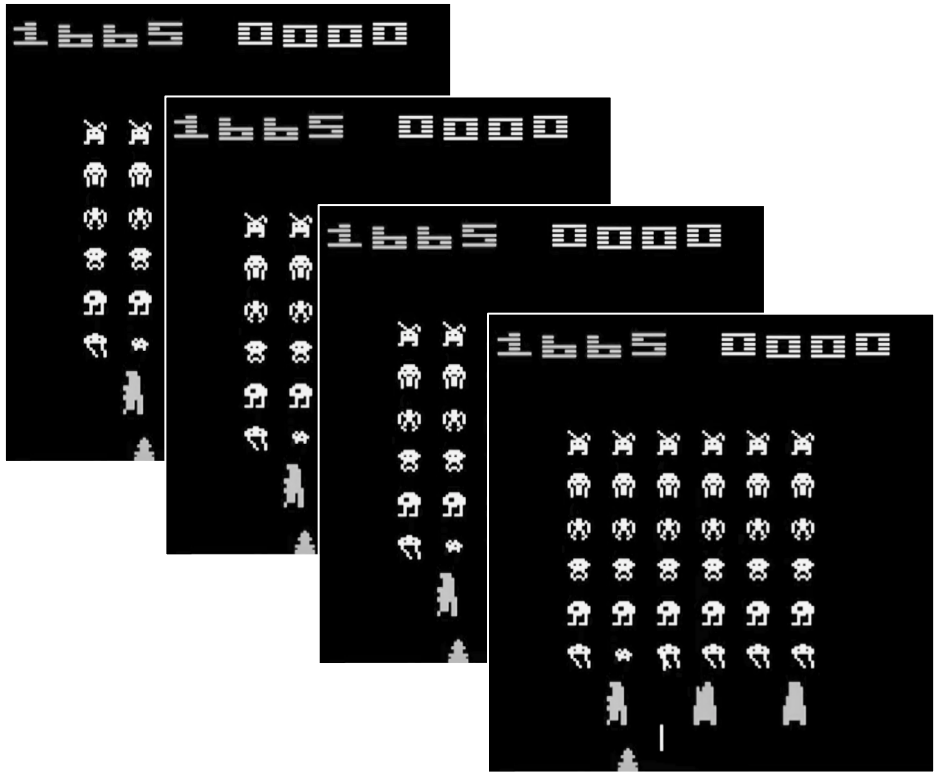


84

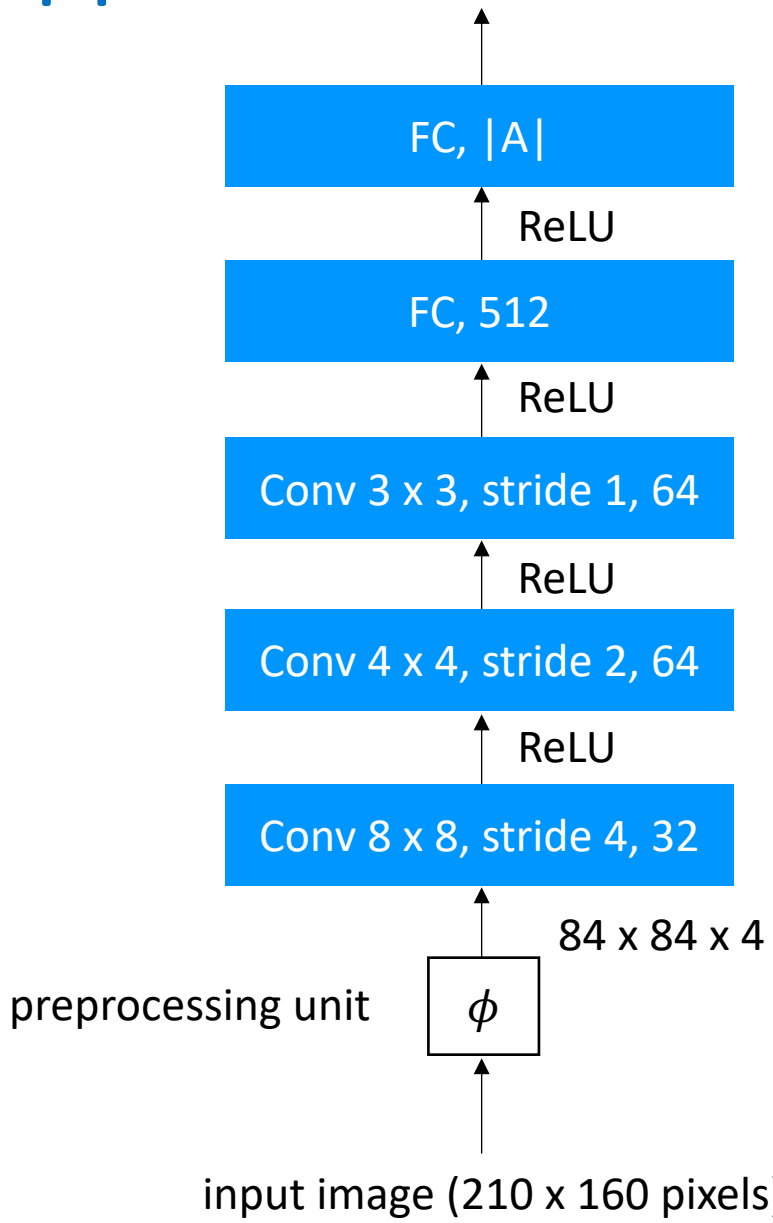
luminance values

Mitigating partial observability by stacking frames

- Input is a stack of 4 most recent frames to mitigate partial observability



Action value approximator



Function approximator learning

- The baseline is the semi-gradient form of Q-learning:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \boldsymbol{w}_t) - \hat{Q}(s_t, a_t; \boldsymbol{w}_t) \right] \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$$

- The gradient vector $\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$ is computed by backpropagation

- In addition, the following tricks are used:
  - Experience replay
  - Separation of timescale
  - Clipping error

Trick #1: experience replay

- The agent uses a replay memory that stores the experience at each time step
- At each time step t, the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is added to the replay memory
- The replay memory accumulates experience observed over multiple episodes
- The replay memory has a fixed size
- Note: experience replay was originally proposed by Lin (1992)

Trick #1: experience replay (cont'd)

$\boldsymbol{w} \leftarrow \boldsymbol{w}_t$

For each $(s_\tau, a_\tau, r_{\tau+1}, s_{\tau+1})$ sampled from the replay memory:

$$y_\tau = \begin{cases} r_{\tau+1} & \text{if } s_{\tau+1} \text{ is a terminal state} \\ r_{\tau+1} + \gamma \max_{a'} \hat{Q}(s_\tau, a'; \boldsymbol{w}) & \text{otherwise} \end{cases}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\left[y_\tau - \hat{Q}(s_\tau, a_\tau; \boldsymbol{w})\right]\nabla_{\boldsymbol{w}} \hat{Q}(s_\tau, a_\tau; \boldsymbol{w})$$

$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}$

Trick # 1: benefits of experience replay

- Stored experience can be used for many updates which allows DQN to learn more efficiently from its experiences

- Reduces variance of the updates because successive updates are not correlated with one another as they would be with standard Q-learning

- Removing the dependence of successive experiences of the current weights, it mitigates one source of instability

Trick #2: separation of timescales

- In the semi-gradient form of Q-learning:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \left[\underbrace{r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \boldsymbol{w}_t)}_{\text{Q-learning target depends on the parameter } \boldsymbol{w}_t} - \hat{Q}(s_t, a_t; \boldsymbol{w}_t) \right] \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$$

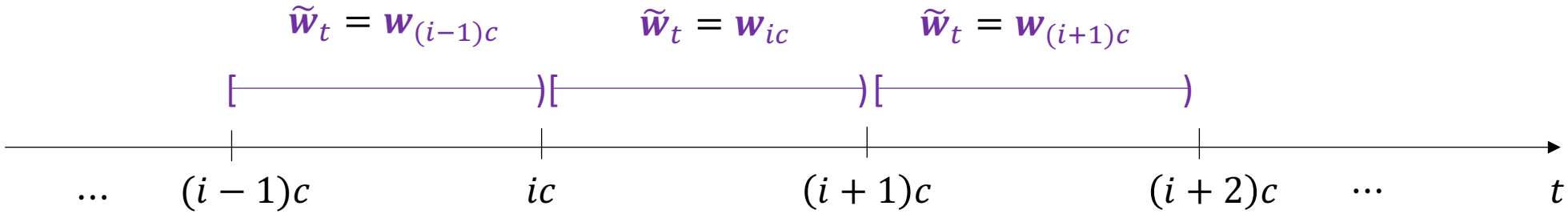
- May cause instability
  - Different from supervised learning where targets are independent of parameters

- Mitigation method: use a fixed parameter value for generation of Q-learning targets over periods of time steps

Trick #2: separation of timescales - DQN approach

- Use $\hat{Q}(s_{t+1}, a; \widetilde{\boldsymbol{w}}_t)$ for generating Q-learning targets where $\widetilde{\boldsymbol{w}}_t$ is periodically updated as follows, for fixed integer value $c \geq 1$,

$$\widetilde{\boldsymbol{w}}_t = \boldsymbol{w}_{\lfloor t/c \rfloor c}$$



- Q-learning targets are generated for fixed parameter vector over periods of c time steps: making Q-learning targets less varying with the parameter

Trick #3: clipping error

- The error term $r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \widetilde{\boldsymbol{w}}_t) - \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$ is clipped so that it remains in [-1,1]

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \Pi \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \widetilde{\boldsymbol{w}}_t) - \hat{Q}(s_t, a_t; \boldsymbol{w}_t) \right] \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w}_t)$$

where $\Pi(x) = \min\{\max\{x, -1\}, 1\}$ performs the clipping to [-1,1]

- Clipping was observed to improve stability

Performance evaluation

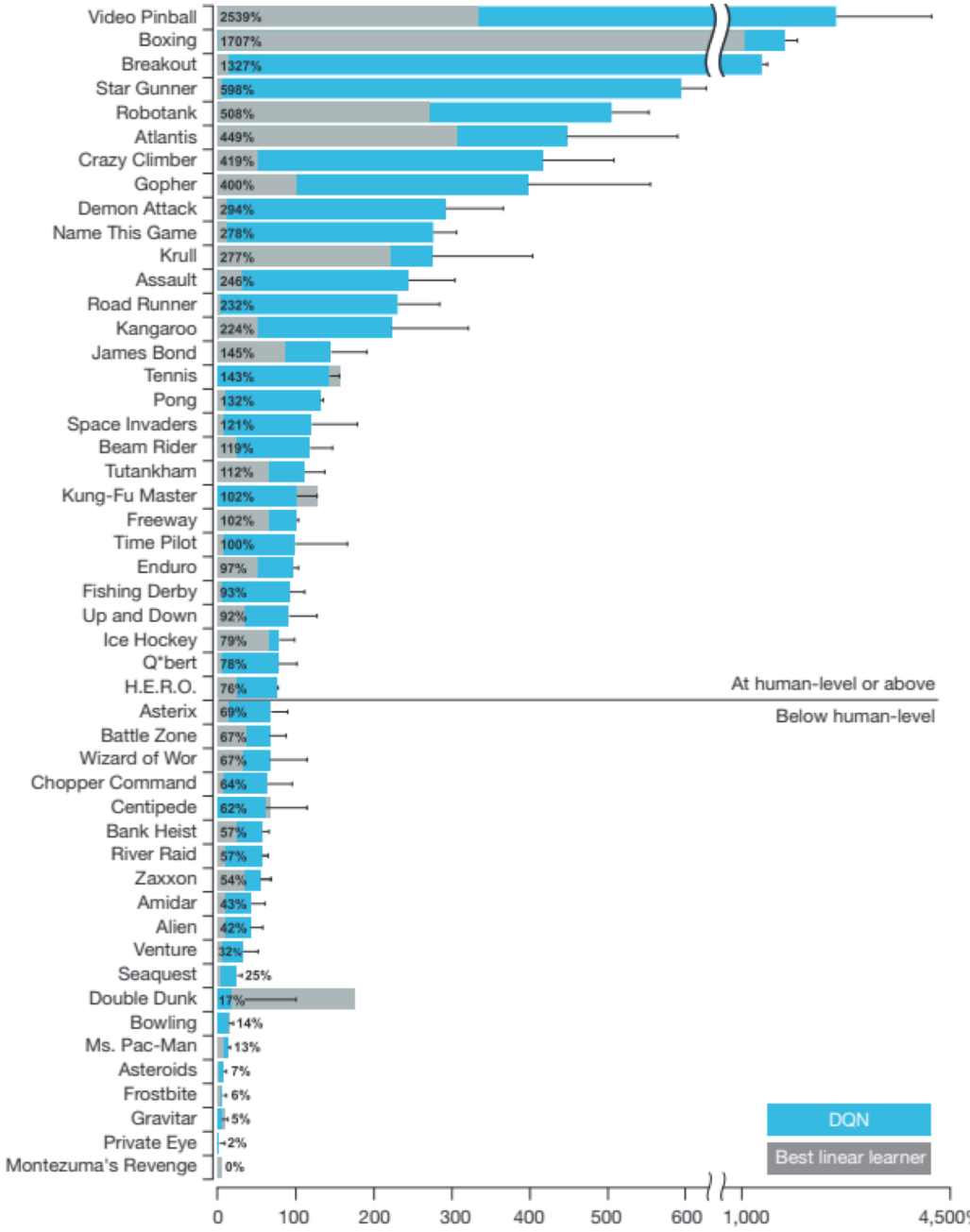
The performance of DQN is normalized with respect to a professional human games tester and random play

The normalized performance of DQN:

$$100 \times \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$

DQN outperforms competing methods in almost all the games, and performs at a level that is broadly comparable with or superior to a professional human games tester (i.e. operationalized as a level of 75% or above) in the majority of games

Error bars indicate standard deviation across 30 evaluation episodes, with different initial conditions



References

- Sutton and Barto, Reinforcement Learning: An Introduction, Chapter 16: Applications and Case Studies, 2nd Edition, The MIT Press, 2018
- Sutton and Barto, Reinforcement Learning: An Introduction, Chapter 8: Generalization and Function Approximation, 1st Edition, The MIT Press, 1998
- Bellemare et al, The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract), IJCAI 2015
- Lin (1992), Self-improving reactive agents based on reinforcement learning, planning and teaching, Machine Learning, 8(3-4):293:321
- Mnih et al, Player Atari with Deep Reinforcement Learning, Arxiv, 2013
- Mnih et al, Human-level Control through Deep Reinforcement Learning, Nature, 518, 529-533 2015

Seminar exercises

- The Mountain Car example - linear methods
- Playing Atari games - DQN