

Reinforcement Learning – COMS4061A/7071A

Function Approximation

Prof. Benjamin Rosman

Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

Based on slides by Rich Sutton, Doina Precup, Steven James
Sutton and Barto [2018], Chapter 9, 10, 11, 16

Previously on RL...

$$v^*(s) \text{ or } q^*(s, a)$$

- Updates towards targets
 - MC: $G_t = R_{t+1} + \gamma R_{t+2} + \dots \gamma^{T+1} R_T$
 - TD: $\delta_t = R_{t+1} + \gamma V(S_{t+1})$
- We have methods that work/converge
- So what's the problem?

Real-world domains

- What if $|S|$ is large? Or infinite?
- What if $|A|$ is large? Or infinite?



Issues

- We can't fit every state value entry in **memory**!
 - We can't **visit** every state!
 - We may never see the same state **twice**!
 - Must **generalise**
-
- Backgammon $\approx 10^{20}$ states
 - Atoms in observable universe $\approx 10^{80}$
 - Go $\approx 10^{170}$ states
 - Robotics: continuous

Function approximation

- We will look to **approximate** the true value function

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

- Imagine we are given training data $s \mapsto v_{\pi}(s)$
- Then find \mathbf{w} to minimise MSE of training samples
 - Just like **supervised** learning!

Aim

- Learn a **parameterised function** to approximate the true value function

- Linear functions
- Neural networks
- Decision trees
- Nearest neighbours
- etc

→ *Input state, output real-valued number*

- Mean-squared value error

Distribution of states under policy

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

Strategy

- We consider **differentiable** function approximators
- Write the **objective function J** in terms of **w**
- Optimise to **learn w**
 - Typically by gradient descent

Stochastic gradient descent

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

- Gradient descent finds **local minimum** by following **direction** of greatest **decrease**
 - Adjust \mathbf{w} in direction of **negative gradient**

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

Targets

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[v_\pi(S) - \hat{v}(S, \mathbf{w})]\nabla \hat{v}(S, \mathbf{w})$$

- But wait! We don't know $v_\pi(S)$
- We have estimates U_t (TD or MC)

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

Targets

- Monte Carlo

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t - \hat{v}(S, \mathbf{w})]\nabla \hat{v}(S, \mathbf{w})$$

- Temporal difference

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla \hat{v}(S, \mathbf{w})$$

 *TD Error*

- And others: n-step returns, λ -returns, etc
- Cool?
 - Not cool.

Semi-gradient methods

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


- Holds because $v_\pi(S)$ is **not a function** of \mathbf{w}
- $R + \gamma \hat{v}(S', \mathbf{w})$ **is** a function of \mathbf{w}
- We will pretend the TD target is **fixed!**
 - **Semi-gradient** method
 - Convergence guarantees not as strong as full gradient



*Here be convergence
guarantees!*

Linear function approximation

Features

- We want $\hat{v}(s, \mathbf{w})$ where $\mathbf{w} \in \mathbb{R}^d$
- Let $\mathbf{x}(s) = (x_1(s), x_2(s), \dots, x_d(s))^T$
- Then $\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$  *Can make a function of (s, a) for approximating q*
- $\mathbf{x}(s)$ is the **feature vector** of state s
 - In linear case, these are called **basis functions**

Basis functions

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

- Let's say our state variables are xy -position
- We could use the variables as features **directly**

$$\mathbf{x}(s) = (1, x, y)^\top$$

- More powerful:
 - **Polynomials** in state variables

- 1st order: $(1, x, y, xy)$
- 2nd order: $(1, x, y, xy, x^2, y^2, x^2y, xy^2, x^2y^2)$



*Nonlinear in state variables, but
still linear in weights!*

Linear update rule

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

- Objective function **quadratic** in weights
- SGD converges to **global minimum**!
- $\nabla \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$
- $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[\mathbf{U}_t - \hat{v}(S, \mathbf{w})]\mathbf{x}(S)$
 - Update = step size \times prediction error \times features
- Semi-gradient converges too!

Approximation for control

- Simply approximate q instead of v
- Then apply e.g. **SARSA**

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

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

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., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 If S' is terminal:

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

 Go to next episode

 Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε -greedy)

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

$S \leftarrow S'$

$A \leftarrow A'$

Least-Squares TD [Bradtke and Barto, 1996]

- If we have a **batch** of data, can we just solve for \mathbf{w} directly?
 - In the linear case, only one global optimum. So yes!

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha [R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$$

- The update should be zero at the optimum

$$\sum_{t=1}^T [R_{t+1} + \gamma \mathbf{w}^\top \mathbf{x}(S_{t+1}) - \mathbf{w}^\top \mathbf{x}(S_t)] \mathbf{x}(S_t) = 0$$

Least-Squares TD

$$\sum_{t=1}^T [R_{t+1} + \gamma \mathbf{w}^\top \mathbf{x}(S_{t+1}) - \mathbf{w}^\top \mathbf{x}(S_t)] \mathbf{x}(S_t) = 0$$

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

$$\mathbf{w}^\top = \mathbf{A}^{-1} \mathbf{b}$$

where $\mathbf{A} = \sum_{t=1}^T [\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1})] \mathbf{x}(S_t)$ and
 $\mathbf{b} = \sum_{t=1}^T R_{t+1} \mathbf{x}(S_t)$

Feature construction for
linear representations

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

Table lookup Features

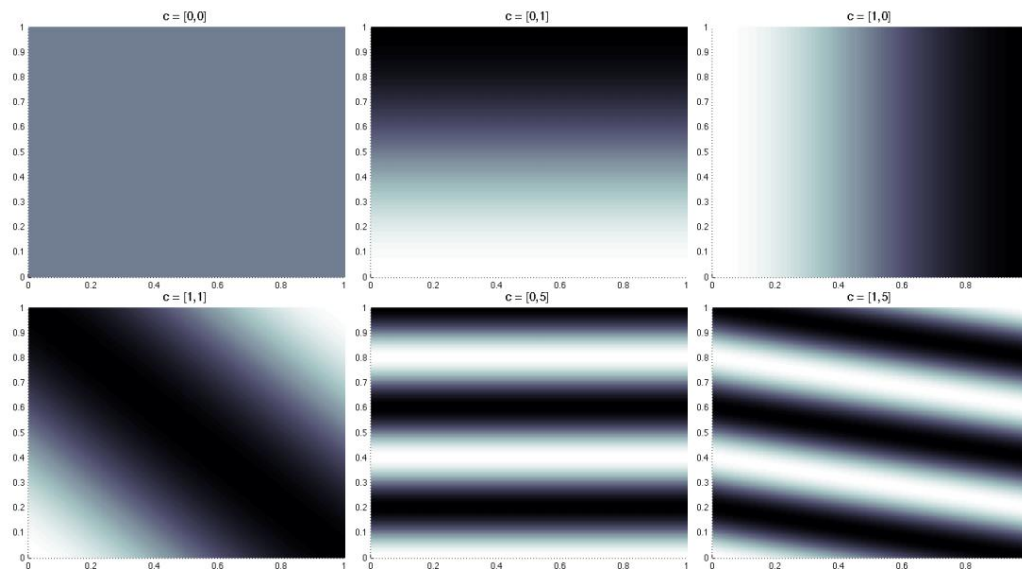
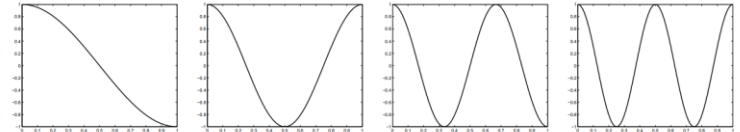
- **Tabular methods** are a **special case** of linear function approximation
- If there are n states, represent each state as a **one-hot encoded** vector
 - If in state 0: $\mathbf{x}(s_0) = (1, 0, 0, 0, \dots)$
 - If in state 1: $\mathbf{x}(s_1) = (0, 1, 0, 0, \dots)$
 - etc
- Then \mathbf{w} is a vector giving the value of each individual state

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

Fourier basis [Konidaris, et al 2011]

- Based on the Fourier series
- The n th order Fourier basis is:

$\mathbf{x}(s) = \cos(\pi \mathbf{c}^i \cdot s)$ where \mathbf{c}^i is a vector of elements in $\{0, \dots, n\}$ for all state variables



$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

Coarse coding

- Consider tasks whose state space is continuous, 2-dimensional
- Draw some **overlapping circles** (each circle is a **feature**)
 - If state in circle, it gets value 1, else 0
- Better **generalisation** between states who are in **more circles** together
- Circle **size** = range of generalisation

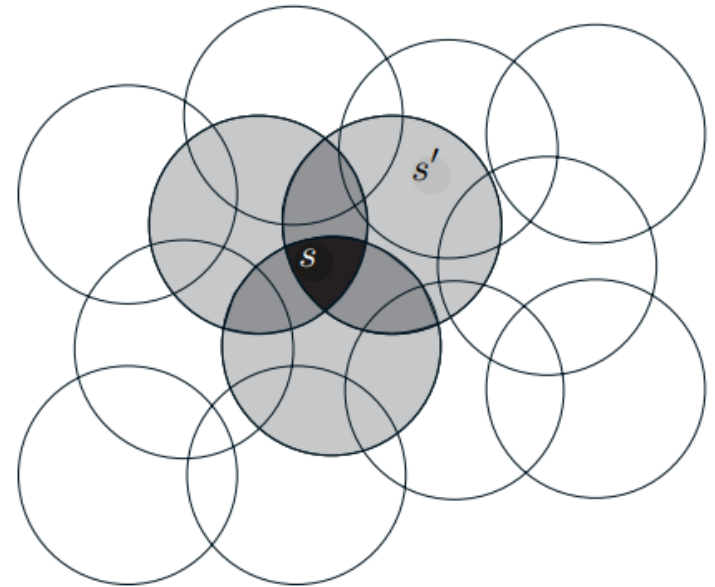
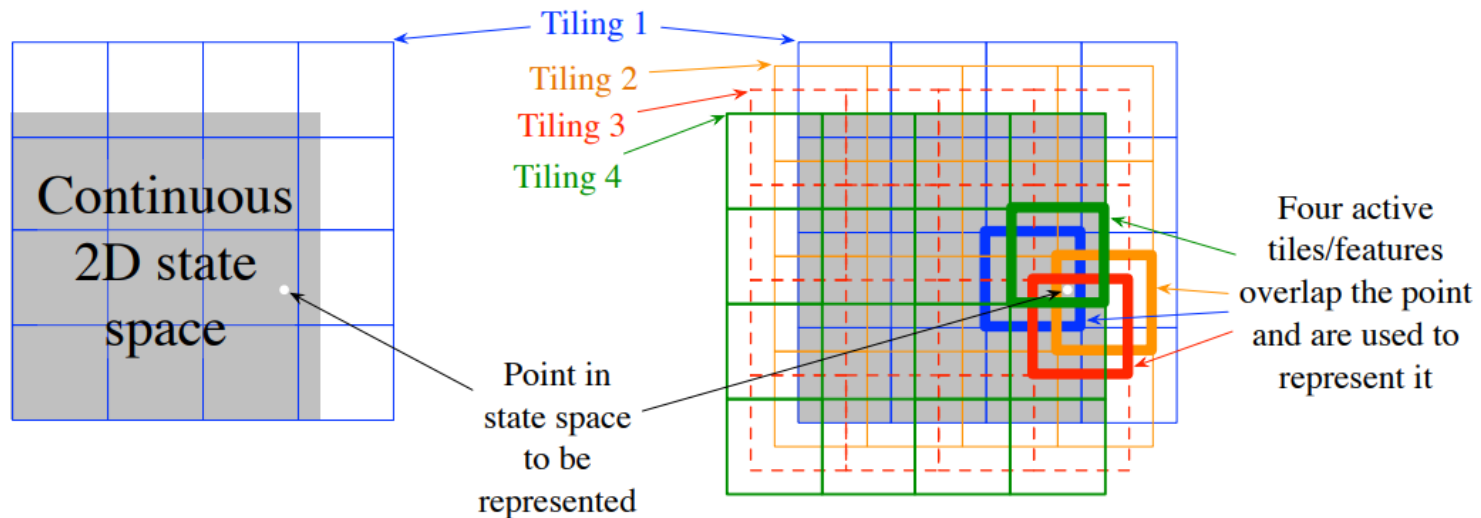


Figure 9.6: Coarse coding. Generalization from state s to state s' depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

Tile coding

- Coarse coding with **grids** offset from one another
- **Efficient** to compute
- Good for **multidimensional** continuous spaces



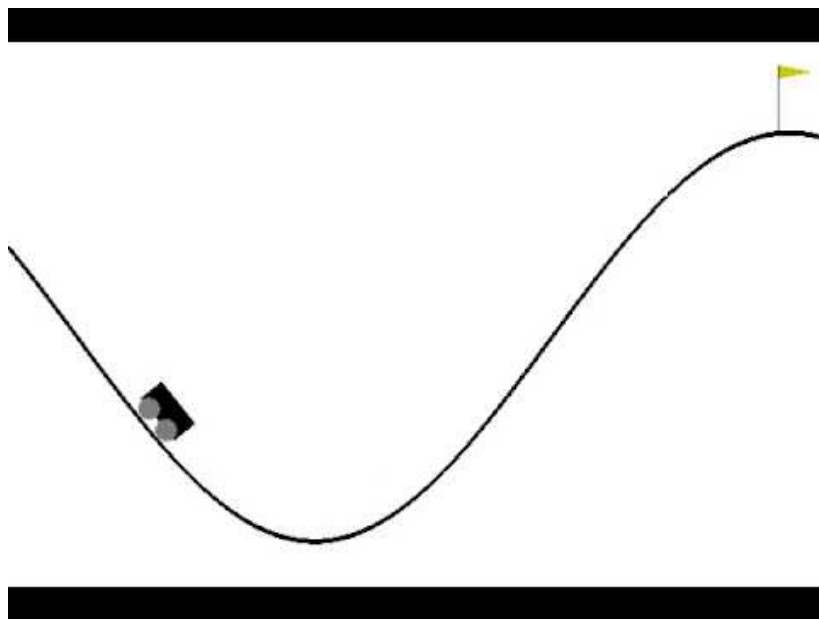
Linear methods don't scale

- Why not?
 - They have nice properties (**convex** error surface)
 - They are easy to use
- But:
 - How many basis functions in a complete n^{th} order Taylor series of d variables?

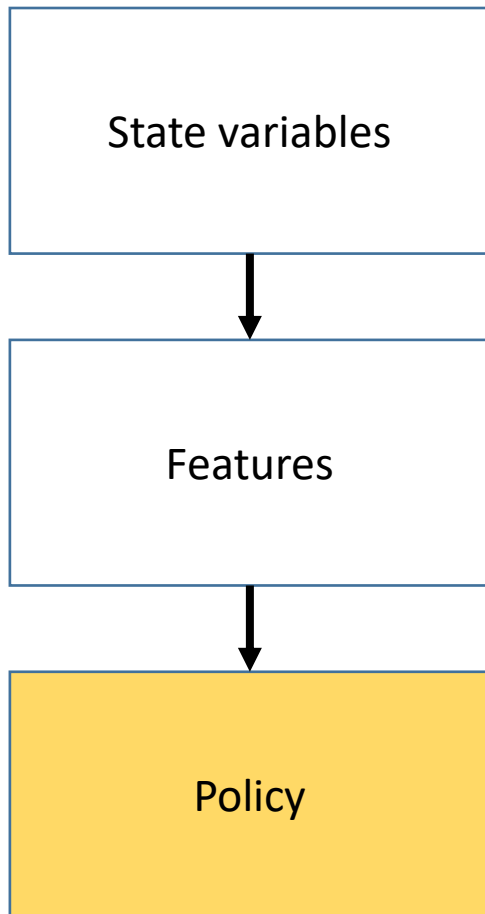
$$(n + 1)^d$$

Non-linear function approximation

What are the “right” features?

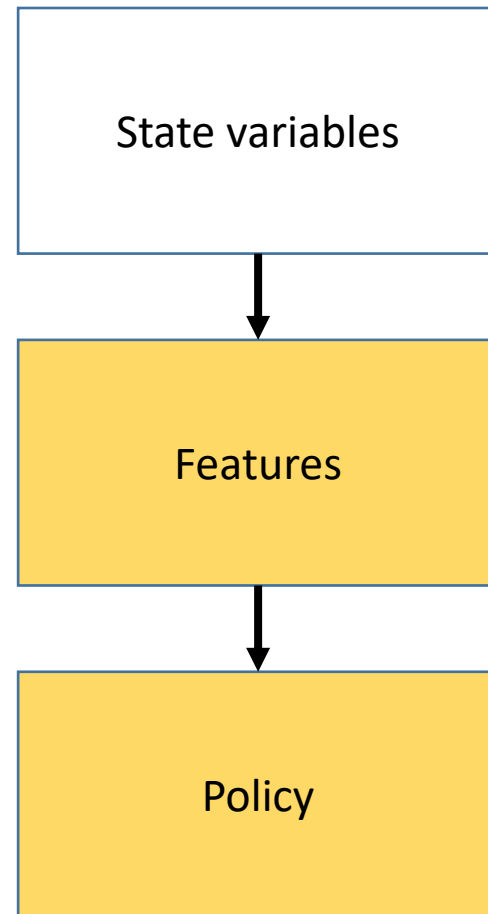


Classic RL



YELLOW = LEARNED

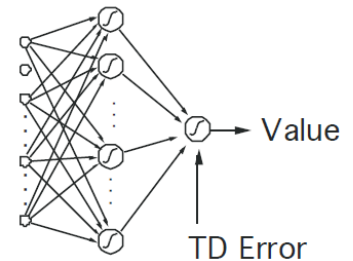
Deep RL



Function Approximation

TD-Gammon: Tesauro (circa 1992-1995)

- At or near best human level
- Learn to play Backgammon through self-play
- 1.5 million games
- Neural network function approximator
- TD(λ)



States = board configurations ($\approx 10^{20}$)

Actions = moves

$$\text{Rewards} = \begin{cases} 1 & \text{win} \\ -1 & \text{lose} \\ 0 & \text{else} \end{cases}$$

Changed the way the best human players played

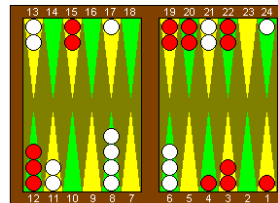
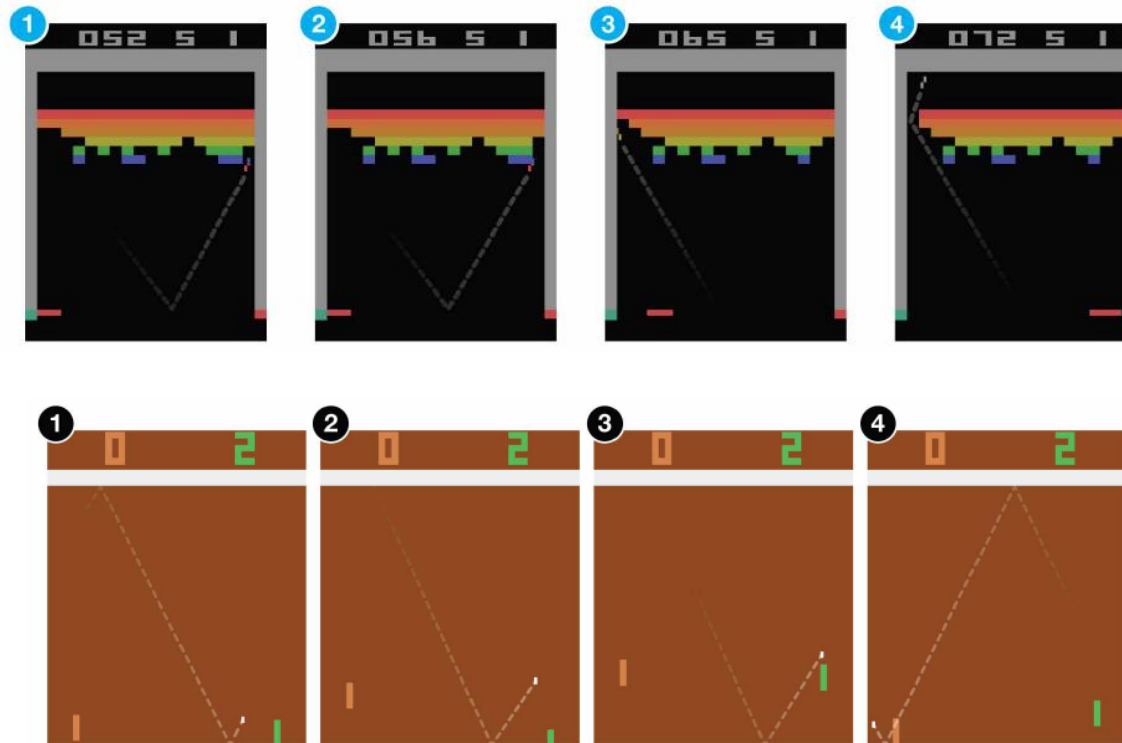


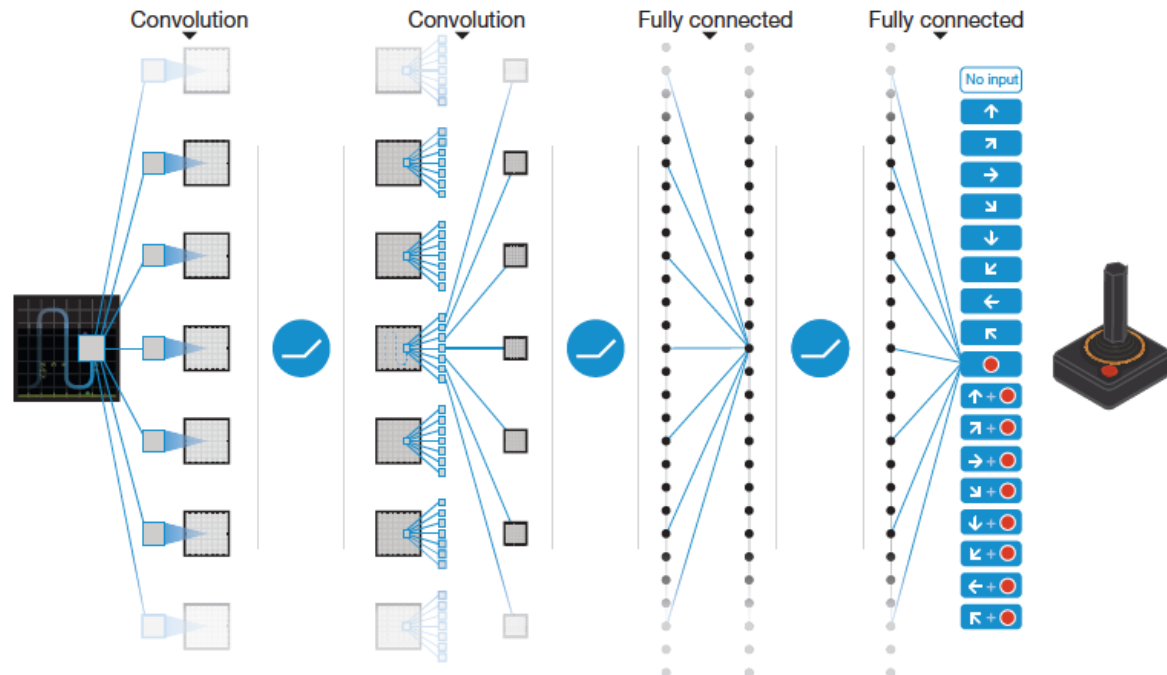
Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 8-4*, 8-4, 21-17, 21-17! TD-Gammon's analysis of the two plays is given in Table 3.

Arcade Learning Environment



[Bellemare 2013]

Deep Q-Networks



[Mnih et al., 2015]

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal

- $J(\mathbf{w}) = \mathbb{E}_{\pi}[\text{target} - \hat{q}(S, A, \mathbf{w}))^2]$
- $\text{target} = r + \gamma \max_{a'} Q(S', a', \mathbf{w})$

Problems

- Use **supervised learning** to move the Q-value function toward target
- But neural network has \sim **million parameters!**
Tough to optimise!
- Data is **not i.i.d**
 - Why?
- **Target changes** after each iteration
 - **Highly-nonstationary**
- Deep Q-learning (**DQN**) uses a number of **tricks**

Experience replay

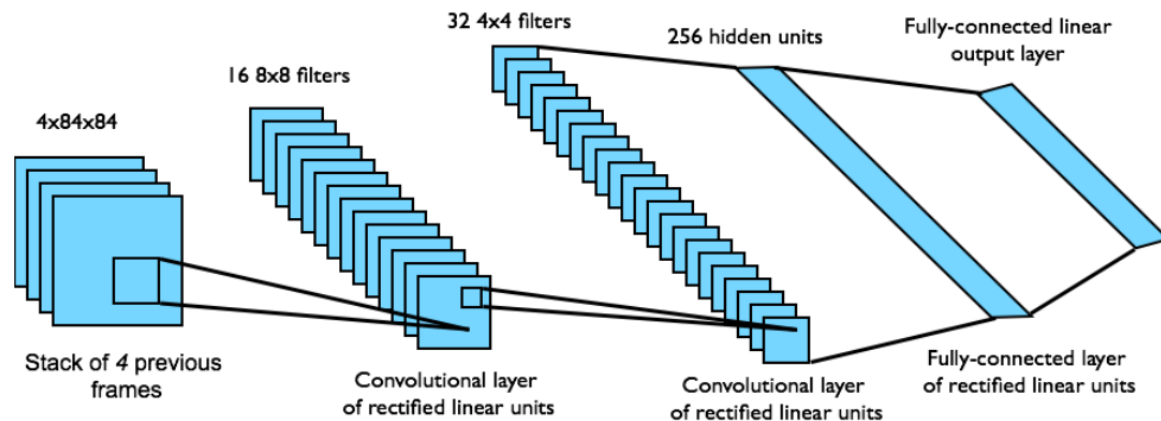
- Reuse batches of old data to update current network
- Take action a_t according to ϵ -greedy policy
 - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- When updating Q-value function:
 - Single sample not enough!
 - Sample mini-batch of transitions (s, a, r, s') from D
 - Decorrelates samples
- Good idea to overwrite old data after time
- Can do fancier things e.g. prioritised experience replay

Target network

- To overcome changing targets
- Have a **second copy** of the network
 - **Freeze** its weights
 - Fixed Q-targets: avoid oscillations
- **Compute Q-learning targets w.r.t old fixed parameters w^-**
- Optimise MSE between Q-network and Q-learning targets
 - $L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[\left(r + \gamma \max_{a'} Q(s', a', w_i^-) - Q(s, a, w_i) \right)^2 \right]$
 - Using stochastic gradient descent on $\nabla_{w_i} L_i(w_i)$
- Periodically **reset** the target network to the current one

DQN for 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 4-18 joystick/button positions
- Reward is change in score for that step



Atari

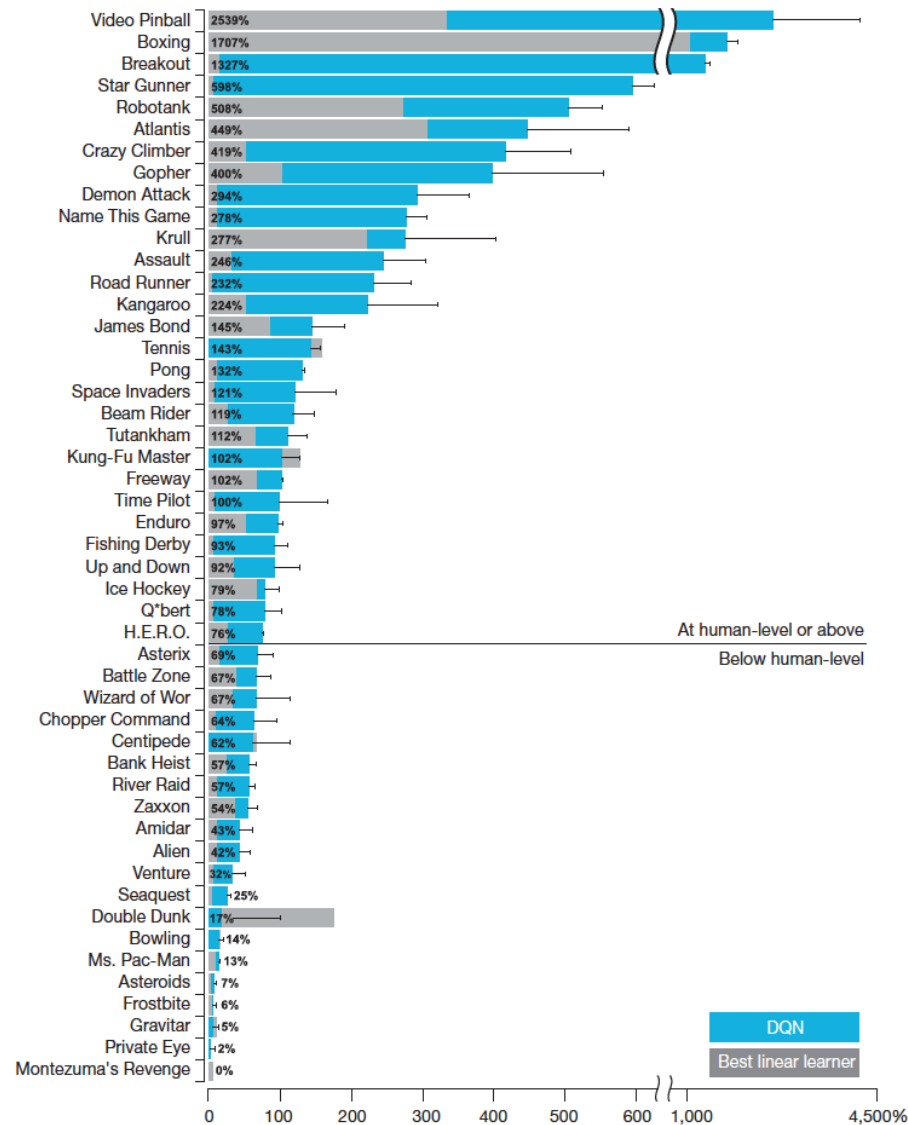
Starting out - 10 minutes of training

**The algorithm tries to hit the ball back, but
it is yet too clumsy to manage.**

[Mnih et al., 2015]

video: Two Minute Papers

Atari Results



[Mnih et al., 2015]

The deadly triad

- Beware!
- Instability/divergence occurs when we combine:
 - Function approximation
 - Bootstrapping (e.g. TD errors)
 - Off-policy training (e.g. Q-learning, experience replay)
- In practice we can use tricks to mitigate
- But note that no theoretical guarantees

Read the paper on Moodle!

Homework

- Use the MountainCar domain from OpenAI gym
 - See Example 10.1, pg 244 in Sutton and Barto [2018]
- Implement the SARSA semi-gradient algorithm with tile coding (8 tiles)
 - You may use the value function code provided on Moodle. The `__call__` function computes the value of a given state-action pair, the `update` function updates the Q-function, and the `act` function derives an epsilon-greedy policy
 - Use ϵ -greedy policies with $\epsilon = 0.1$ and a learning rate of $\alpha = 0.1$

By next week's lecture, submit on Moodle:

1. Run the algorithm for 500 episodes, and plot the number of steps per episode (on a log scale) as a function of number of episodes. Average your results over 100 runs and submit the graph
2. Use the rendering ability of OpenAI gym to render the learned Q-function acting in the environment (do not use exploration for this). Create a video of it solving the task and upload the video
3. Your code