Artificial Intelligence

Steve James

Reinforcement Learning

Machine learning

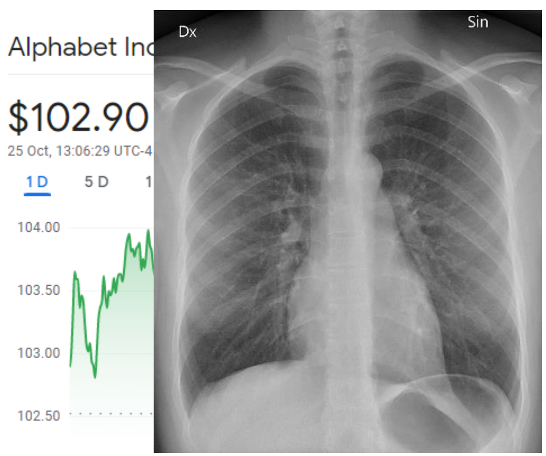
- Subfield of AI concerned with learning from data

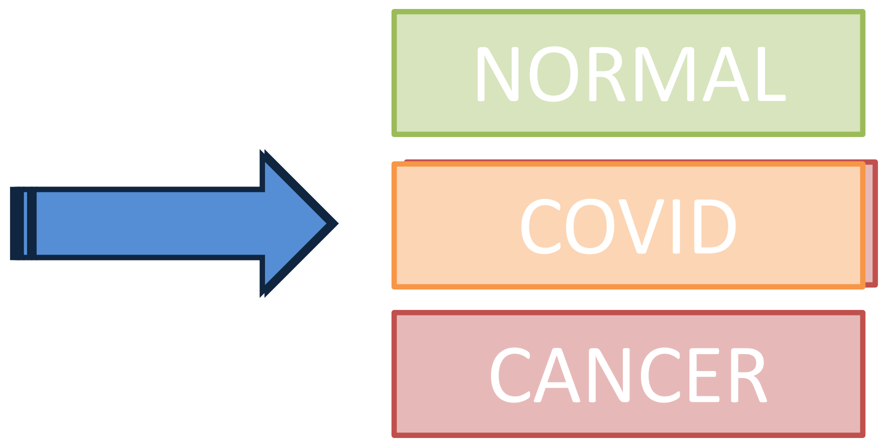
- Broadly, using:
  - Experience
  - To improve performance
  - On some task

(Tom Mitchell, 1997)

What is machine learning?

- Supervised learning is about making predictions, e.g.
  - Classification
  - Regression





- Predictions should inform our behaviours

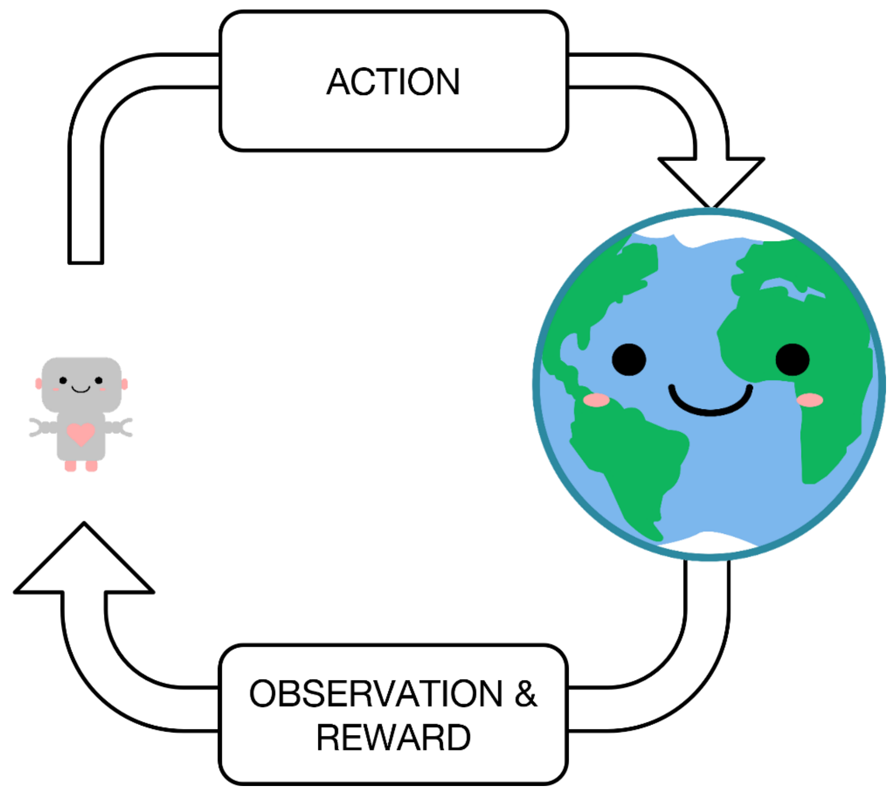
Reinforcement vs supervised learning

- In SL, model is given data with labels
  - Must learn input-output function

- In RL, environment gives model positive/negative "rewards" (numerical feedback)
  - Learn behaviours to maximise total rewards over time
- Rewards encode what to do, rather than how

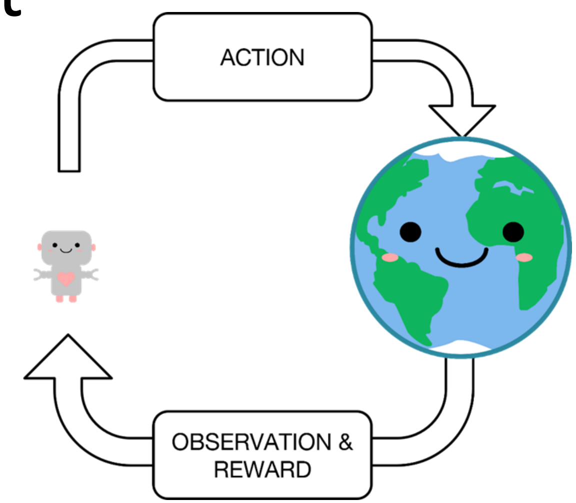
Reinforcement learning loop

- Decision maker (agent) exists within an environment
- Agent takes actions based on the environment state
- Environment state updates
- Agent receives feedback as rewards



MDPs

- Agent interacts with environment
- At each time t
  - Receives sensor signal s_t
  - Executes action a_t
  - Transition
    - New sensor signal s_{t+1}
    - Reward r_t



- Find policy π that maximises expected return

$$\max_{\pi}\left[R^{\pi}=\sum_{t=0}^{\infty}\gamma^{t}r^{t}\right]$$

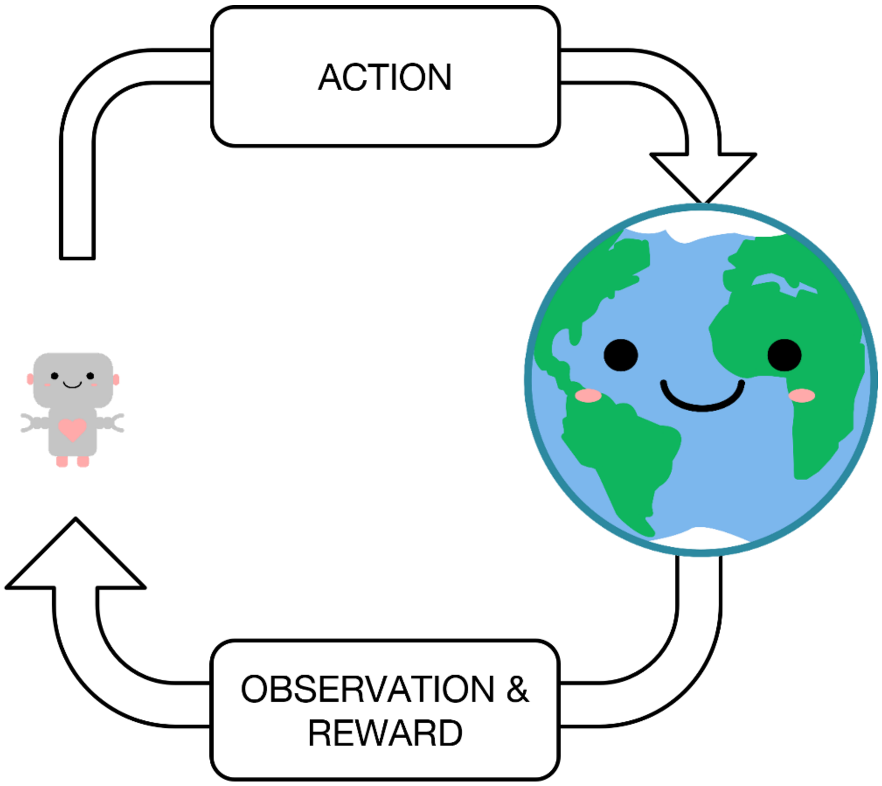
Markov decision processes

$\langle S, A, R, T, \gamma \rangle$

- S: set of states
- A: set of actions
- γ: discount factor $\in [0,1]$
- R: reward function
  - $R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'
- T: transition function
  - $T(s'|s, a)$ is the probability of transitioning to state s' after taking action a in state s

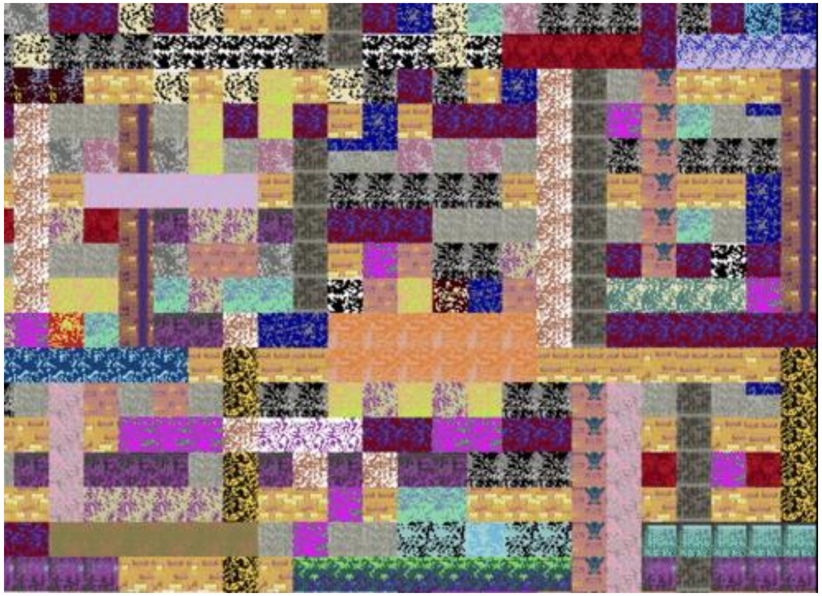
But there's a problem

- We don't know T and/or R!
- Must generate samples of data (s, a, r, s') from environment
  - Use that to learn value functions
- We need:
  - Method to pick actions
  - To keep track of current estimate of value function



RL is harder than you think!

- Agent knows nothing



MDPs

- Key quantity is the return given by a policy from a state:

$$R^{\pi}(s)$$

- Define the value function to estimate this quantity:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t}\right]$$

Planning via policy iteration

- In planning, we used policy iteration to find optimal policy
  - Start with policy π
  - Estimate V^{π}
  - Improve π
    - $\pi(s) = \max_a \mathbb{E}[r + \gamma V^{\pi}(s')]$, $\forall s$

Repeat

- More precisely:
  - $\pi(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[r(s,a,s') + \gamma V(s')]$

Value functions

- For learning, we use a state-action value function:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_t \mid s_0 = s, a_0 = a\right]$$

- This is the value of executing a in s, **then** following π

- Note: $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$

Policy iteration

- Start with policy π
- Learn Q^{π}
- Improve π
  - $\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)\ \forall s$

Repeat

- Need to do a lot more learning
  - $|A| \times$ value functions
- … but now one-step greedy lookahead is trivial

Value function learning

- Learning proceeds by gathering samples of $Q(s, a)$
- Methods differ by
  - How you get the samples?
  - How you use them to update Q?

Action selection

- Assume agent is in state s with value function estimate Q
  – What action should agent take?

- Exploration vs exploitation
  – Exploitation: take best action according to Q
  – Exploration: take a different action

- Tradeoff: how do we know if we're optimal? Maybe something better out there?

Action selection strategy

- Common selection strategy is ϵ-greedy
- With probability $1 - \epsilon$, exploit:
  - Pick action $a = \underset{a'}{\operatorname{argmax}}\, Q(s, a')$
- With probability ϵ, explore:
  - Pick action uniformly at random

Learning a value function

- We use action selection strategy and generate (s, a, r, s')
  - Must use to update estimate of value function

- Standard update towards a target:

$$f_{t+1} \leftarrow f_t + \alpha(target\ - f_t)$$

- Value function represents expected optimal return. So we want:

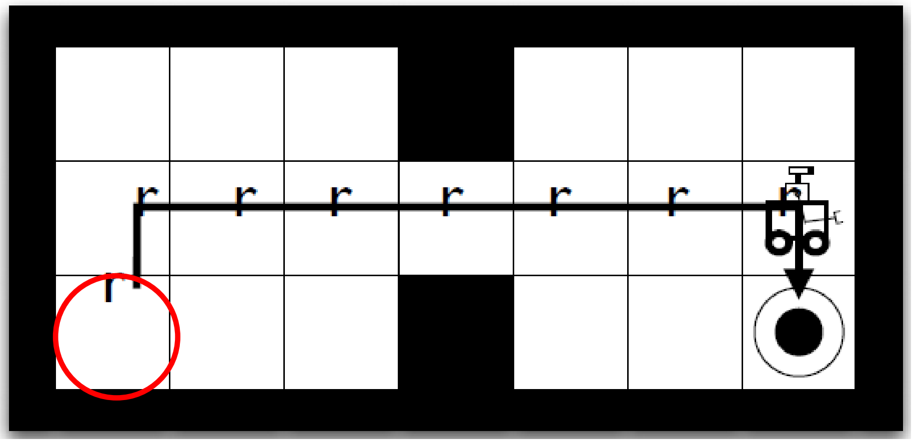
$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha(\mathbb{E}[R^\pi]\ - Q_t(s,a))$$

- But don't know expected return!

Monte Carlo

- Simplest thing you can do: sample $R(s)$

$R = \sum \gamma^t r_t$



- Do this repeatedly and average:

$$Q^{\pi}(s, a) = \frac{R_1(s) + R_2(s) + \ \ldots + R_n(s)}{n}$$

Temporal difference learning

- Estimate return based on current value function

Update toward estimated return

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha[r(s,a,s') + \gamma Q_t(s',a') - Q_t(s,a)]$$

- $r(s,a,s') + \gamma Q_t(s',a') - Q_t(s,a) = 0$
  - Q is correct if this holds in expectation for all states
  - When not, temporal difference error

SARSA

- Initialise $Q(s, a) = 0$ for all s, a

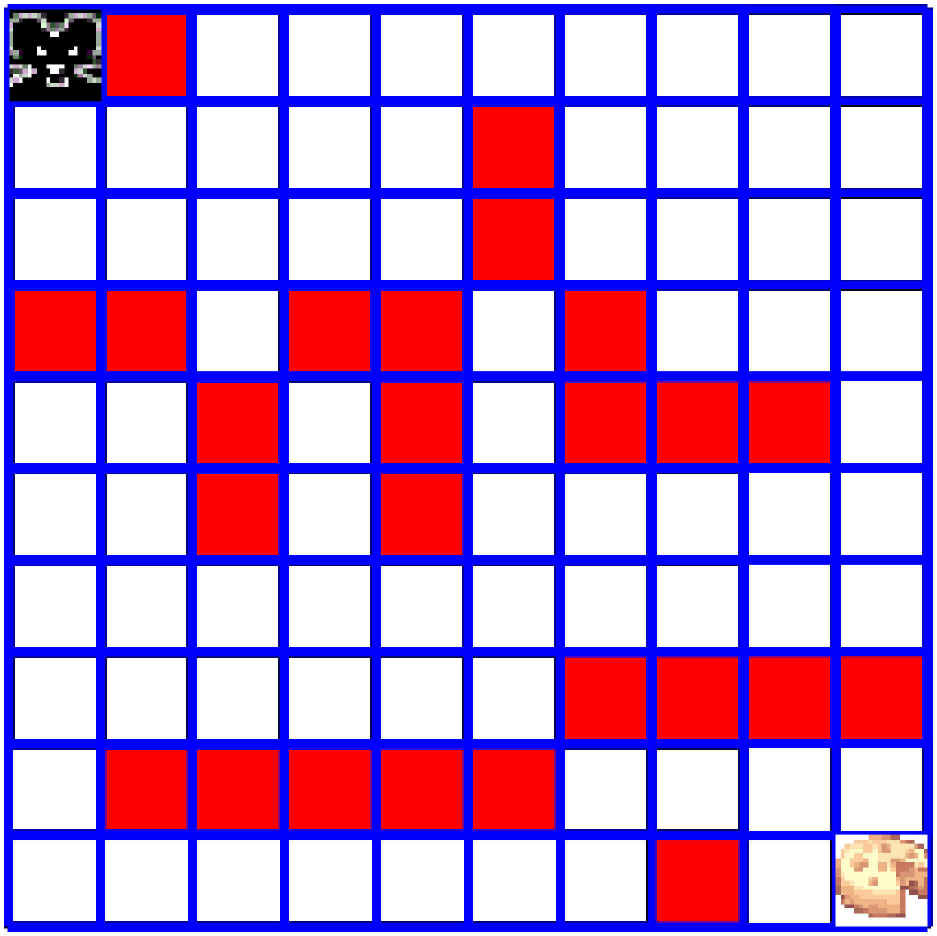
- Repeat (for n episode):
  - Observe s
  - Select a using Q (e.g. ϵ-greedy)
  - Observe transition $\langle s, a, r, s' \rangle$
  - Select a' from s' using Q (e.g. ϵ-greedy)
  - Compute TD error: $\delta = r + \gamma Q(s', a') - Q(s, a)$
  - $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$
  - If not end of episode, repeat

Q learning

- Initialise $Q(s, a) = 0$ for all s, a

- Repeat (for n episode):
  - Observe s
  - Select a using Q (e.g. ϵ-greedy)
  - Observe transition $\langle s, a, r, s' \rangle$
  - Select $a' = \operatorname*{argmax}_b Q(s', b)$
  - Compute TD error: $\delta = r + \gamma Q(s', a') - Q(s, a)$
  - $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$
  - If not end of episode, repeat

Tabular gridworlds



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

$$V^{\pi}(s) \approx \hat{V}(s, \boldsymbol{w}) \text{ or } Q^{\pi}(s, a) \approx \hat{Q}(s, a, \boldsymbol{w})$$

- We want to learn weights according to some objective

- Objective to minimise:

target

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(r + \gamma \hat{Q}(s', a', \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w}))^2]$$

- Hopefully $\hat{Q}$ is differentiable so we can use gradient descent!

Linear function approximation

- We want $\hat{Q}(s, a, \boldsymbol{w})$ where $\boldsymbol{w} \in \mathbb{R}^d$

- Let $\boldsymbol{x}(s, a) = \left(x_1(s, a), x_2(s, a), \dots, x_d(s, a)\right)^\top$

- Then $\hat{Q}(s, a, \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}(s, a)$

- $\boldsymbol{x}(s, a)$ is the feature vector
  - In linear case, these are called basis functions

Basis functions

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

- Let's say our state variables are xy-position

- We could use the variables as features directly
$$\boldsymbol{x}(s) = (1, x, y)^\top$$

- More powerful:
  - Polynomials in state variables
    - 1st order: $(1, x, y, xy)$
    - 2nd order: $(1, x, y, xy, x^2, y^2, x^2y, xy^2, x^2y^2)$

Linear update rule

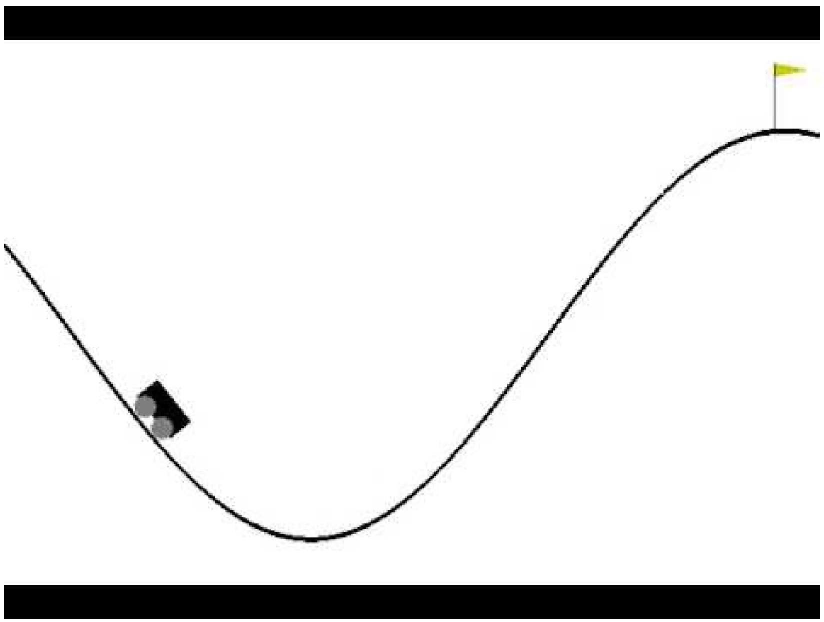
$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(r + \gamma \boldsymbol{w}^{\top}\boldsymbol{x}(s', a') - \boldsymbol{w}^{\top}\boldsymbol{x}(s, a))^2]$$

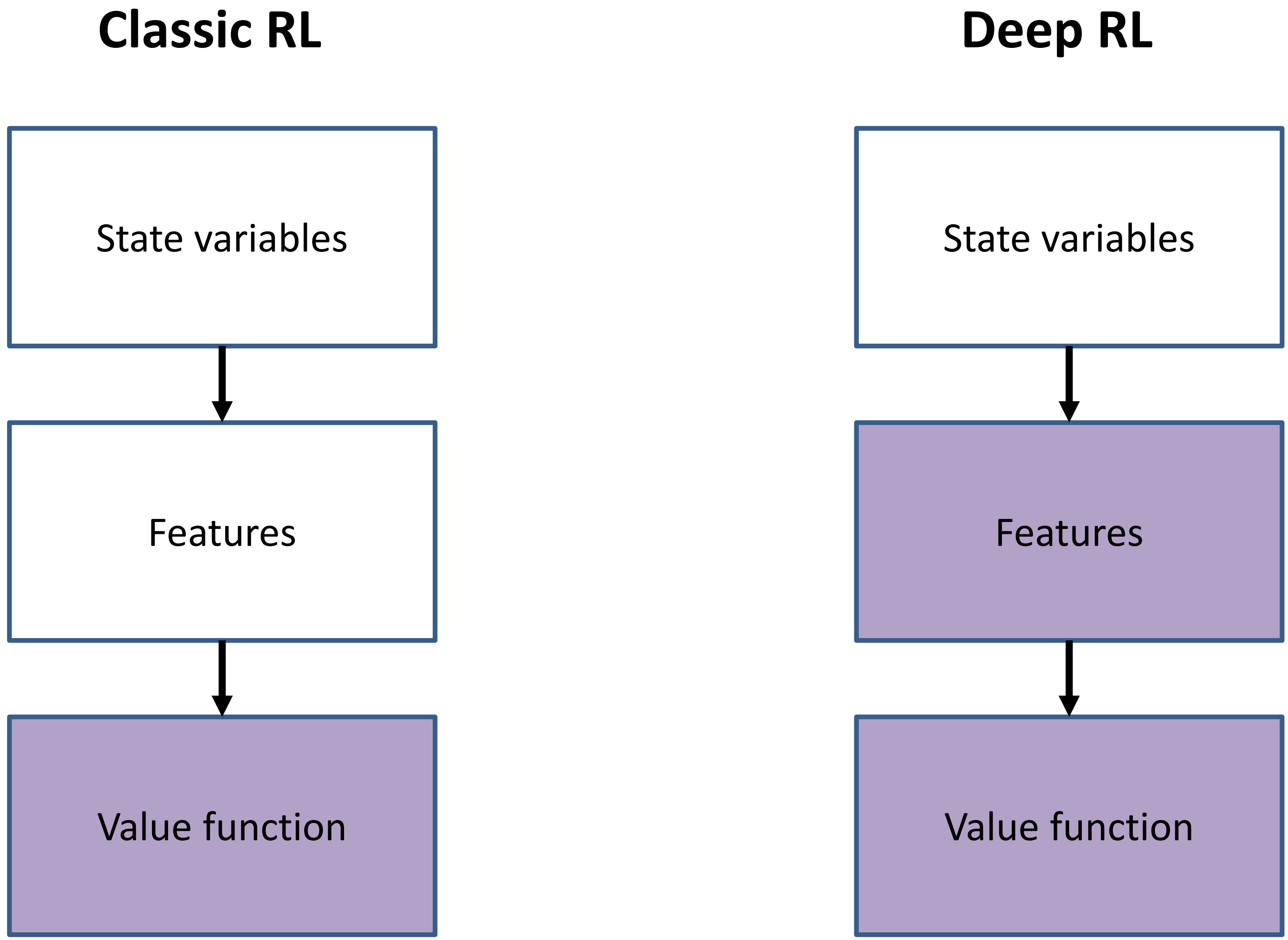
- Objective function quadratic in weights
- SGD converges to global minimum!

- $\nabla \hat{Q}(s, \boldsymbol{w}) = \ \boldsymbol{x}(s, a)$

- $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\left[target - \hat{Q}(s, a, \boldsymbol{w})\right]\boldsymbol{x}(s)$
  - Update = step size × prediction error × features

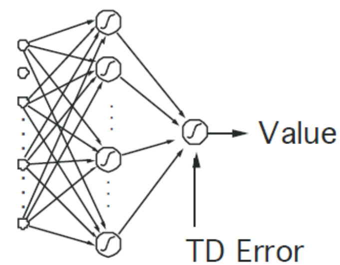
What are the “right” features?





Classic RL
Deep RL
State variables
Features
Value function
State variables
Features
Value function

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

Changed the way the best human players played

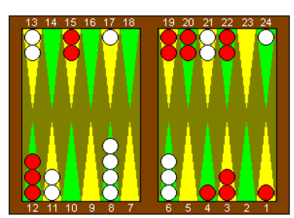
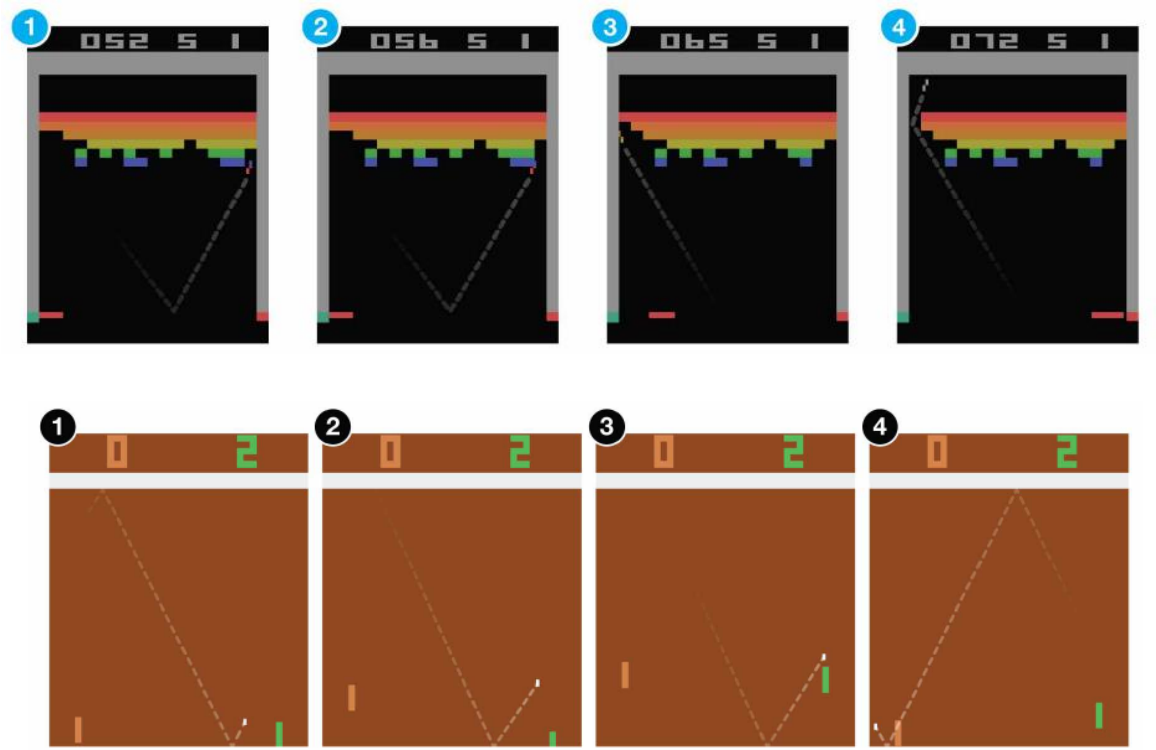


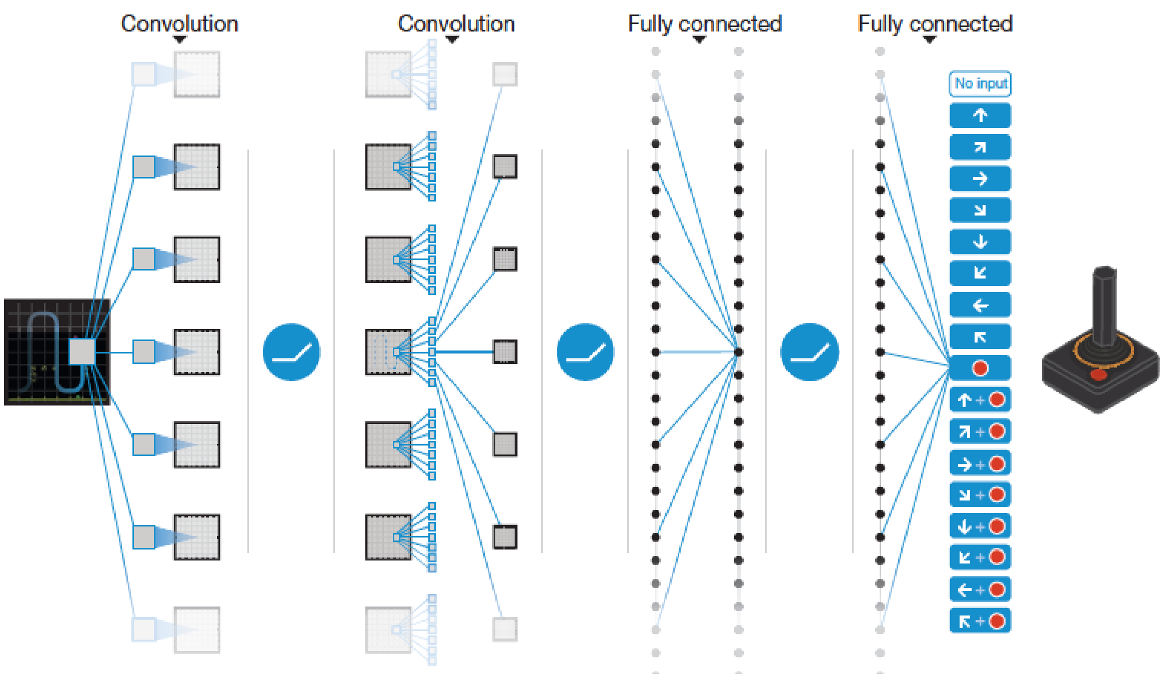
Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 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 $Q(terminal, \cdot) = 0$

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

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[target - \hat{Q}(s, a, \boldsymbol{w}))^2]$$

$$target = r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w})$$

Problems

- Use supervised learning to move the Q-value function toward target
- But neural network has ~million parameters! Tough to optimise!

- 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

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

Atari

What if actions are continuous?

- How would we represent $Q_\pi(s, a)$?

- Idea: ignore value functions completely!

- Directly learn parameterised policy
$$\pi_\theta(s, a) = \pi(s, a; \theta)$$

- Use gradient ascent to find θ that maximises return
  - Note: if π is not differentiable, use gradient free methods like genetic algorithms, etc

REINFORCE

- Execute policy in environment until episode ends
- Compute R, discouted sum of rewards from episode

- Do gradient ascent on $\frac{dR}{d\theta}$
  - Gradient of return with respect to policy parameters

$$\Delta\theta = \alpha R \nabla_\theta \log \pi_\theta(a_t|s_t)$$

REINFORCE

- Initialise θ

- For each episode:
  - Choose actions according to π_θ: $a \sim \pi_\theta(a|s)$
  - Gather samples $\{s_1, a_1, r_1, \dots, s_T, a_T, r_T\}$
  - For $t = 1\ to\ T$
    - $\theta \leftarrow \theta + \alpha R_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

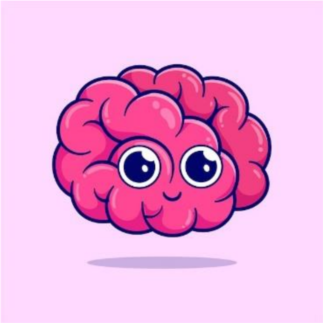
Learning Dynamic Arm Motions for Postural Recovery

Scott Kuindersma, Rod Grupen, Andy Barto
University of Massachusetts Amherst

Humanoids 2011
Bled, Slovenia

Reinforcement learning

- Very active area of current research (including here! ☺)
  - Robotics
  - Operations research
  - Computer games
  - Theoretical neuroscience



- AI
  - The primary function of the brain is control

Summary

- The problem of learning behaviours from experience

- Components of solutions
  - Policies, value functions
  - Exploration/exploitation

- Large, continuous spaces
  - Function approximation
  - Incorporate ideas from supervised learning, deep learning, etc to scale up