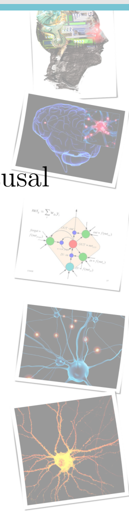
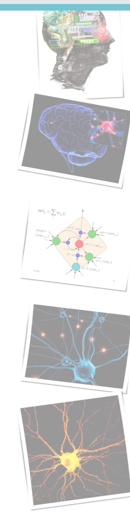
An introduction to Reinforcement Learning (with an intro to neural networks and causal reasoning)

Spyros Samothrakis
Research Fellow, IADS
Univerisity of Essex

November 14, 2016





What is Reinforcement Learning?

- *Reinforcement learning is the study of how animals and artificial systems can learn to optimize their behavior in the face of rewards and punishments* – Peter Dyan, Encyclopedia of Cognitive Science
- **Not** supervised learning - the animal/agent is not provided with examples of optimal behaviour, it has to be discovered!
- **Not** unsupervised learning either - we have more guidance than just observations

Links to other fields

- It subsumes most artificial intelligence problems
- Forms the basis of most modern intelligent agent frameworks
- Ideas drawn from a wide range of contexts, including psychology (e.g., Skinner's "Operant Conditioning"), philosophy, neuroscience, operations research, **Cybernetics**
- Modern Reinforcement Learning research has fused with Neural Networks Research

Examples of Reinforcement Learning closer to CS

- Play backgammon/chess/go/poker/any game (at human or superhuman level)
- Helicopter control
- Learn how to walk/crawl/swim/cycle
- Elevator scheduling
- Optimising a petroleum refinery
- Optimal drug dosage
- Create NPCs

The Markov Decision Process

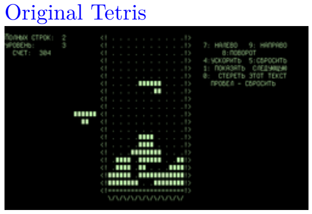
- The primary abstraction we are going to work with is the Markov Decision Process (MDP).
- MDPs capture the dynamics of a mini-world/universe/environment
- An MDP is defined as a tuple $< S, A, T, R, \gamma >$ where:
  - S, $s \in S$ is a set of states
  - A, $a \in A$ is a set of actions
  - $R : S \times A$, $R(s, a)$ is a function that maps state-actions to rewards
  - $T : S \times S \times A$, with $T(s'|s, a)$ being the probability of an agent landing from state s to state s' after taking a
  - γ is a discount factor - the impact of time on rewards

The Markov Property and States

- States represent sufficient statistics.
- Markov Property ensures that we only care about the present in order to act - we can safely ignore past states
- Think Tetris - all information can be captured by a single screen-shot

First DOS Version

Original Tetris



Agents, Actions and Transitions

- An agent is an entity capable of actions
- An MDP can capture any environment that is inhabited either by
  - Exactly one agent
  - Multiple agents, but only one is adaptive
- Notice how actions are part of the MDP - notice also how the MDP is a "world model"
- The agent is just a "brain in a vat"
- The agent perceives states/rewards and outputs actions
- Transitions specify the effects of actions in the world (e.g., in Tetris, you push a button, the block spins)

More on states, agents and actions

- Pick a game
- What would be state in the game?
  - Do agents/NPCs have access to it?
- Do agents/NPCs have access to actions
- Do agents/NPCs have access to transitions?

Rewards and the Discount Factor

- Rewards describe state preferences
- Agent is happier in some states of the MDP (e.g., in Tetris when the block level is low, a fish in water, pacman with a high score)
- Punishment is just low/negative reward (e.g., being eaten in pacman)
- γ, the discount factor,
  - Describes the impact of time on rewards
  - "I want it now", the lower γ is the less important future rewards are
- There are no "springs/wells of rewards" in the real world
  - What is "human nature"?

Examples of Reward Schemes

- Scoring in most video games
- The distance a robot walked for a bipedal robot
- The amount of food an animal eats
- Money in modern societies
- Army medals ("Gamification")
- Vehicle routing
  - (-Fuel spent on a flight)
  - (+ Distance Covered)
- Cold/Hot
- Do you think there is an almost universal reward in modern societies?

Long Term Thinking

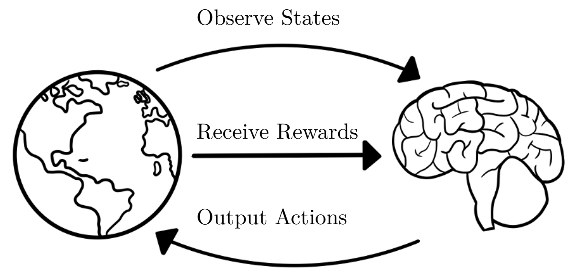
- It might be better to delay satisfaction
- Immediate reward is not always the maximum reward
- In some settings there are no immediate rewards at all (e.g., most solitaire games)
- MDPs and RL capture this
- "Not going out tonight, study"
- Long term investment

Policy

- The MDP (the world) is populated by an agent (an actor)
- You can take actions (e.g., move around, move blocks)
- The type of actions you take under a state is called the *policy*
- $\pi : S \times A$, $\pi(s, a) = P(a|s)$, a probabilistic mapping between states and actions
- Finding an optimal policy is *mostly* what the RL problem is all about

The Full Loop

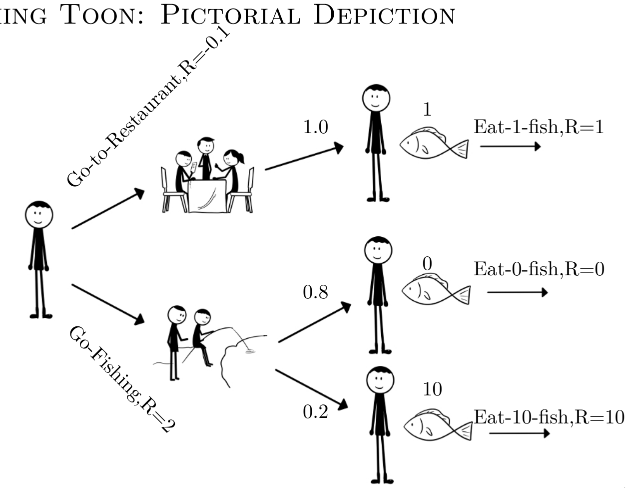
- See how the universe described by the MDP defines actions, not just states and transitions
- An agent needs to act upon what it perceives
- Notice the lack of body - “brain in a vat”. Body is assumed to be part of the world.



Fishing Toon

- Assume a non-player character (let’s call her *toon*)
- Toon is Hungry!
- Eating food is rewarding
- Has to choose between going fishing or going to the restaurant (to eat fish)
  - Fishing can get you better quality of fish (more reward), but you might also get no fish at all (no reward)!
  - Going to the restaurant is a low-risk, low-reward alternative

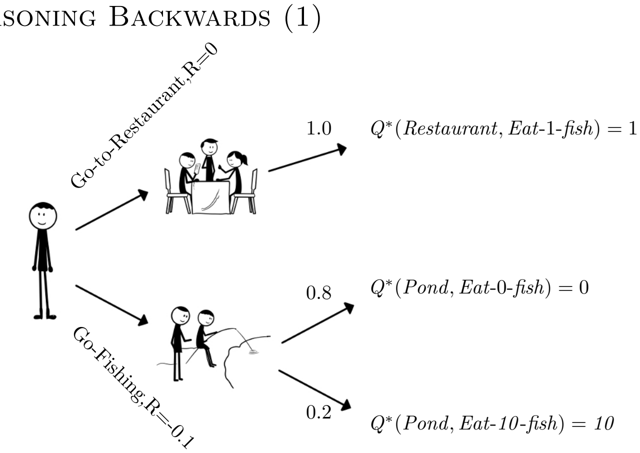
Fishing Toon: Pictorial Depiction



Sum of Expected Rewards

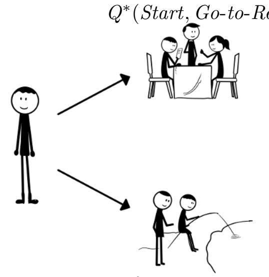
- Our toon has to choose between two different actions
- `Go-To-Restaurant` or `Go-Fishing`
- We assume that toon is interested in maximising *the expected sum* of happiness/reward
- We can help the toon reason using the tree backwards

Reasoning Backwards (1)



Reasoning Backwards (2)

$Q^*(Start, Go\text{-}to\text{-}Restaurant) = 1$



$Q^*(Start, Go\text{-}Fishing) = 0.2 * 10 + 0. * 8 * 0.0 - 0.1 = 1.9$

Correct Action

- Toon should go `Go-Fishing`
- **Would you do the same?**
- **Would a pessimist toon do the same?**
- We just went through the following equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a' \in A} Q^*(s',a')$$

- Looks intimidating - but it's really simple
- Let's have a look at another example
  - How about toon goes to the restaurant after failing to fish?
  - How would that change the reward structure?

Agent Goals

- The agent's goal is to maximise its long term reward $\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R\left(s^t, a^t\right) \right]$
- Risk Neutral Agent - think of the example above
- Rewards can be anything, but most agents receive rewards only in a very limited amount of states (e.g., fish in water)
- What if your reward signal is only money?
  - Sociopathic, egotistic, greed-is-good Gordon Gekko (*Wall Street*, 1987)
  - No concept of "externalities" - agents might wreak havoc for marginal reward gains
  - Same applies to all "compulsive agents" - think Chess

Searching for a good Policy

- One can possibly search through all combinations of policies until she finds the best
- Slow, does not work in larger MDPs
- Exploration/Exploitation dilemma
  - How much time/effort should be spend exploring for solutions?
  - How much time should be spend exploiting good solutions?

Planning

- An agent has access to model, i.e. has a copy of the MDP (the outside world) in its mind
- Using that copy, it tries to "think" what is the best route of action
- It then executes this policy on the real world MDP
- You can't really copy the world inside your head, but you can copy the dynamics
- "This and that will happen if I push the chair"
- Thinking, introspection...
- If the model is learned, sometimes it's called "Model Based RL"

Bellman Expectation Equations / Bellman Backups

- The two most important equations related to MDP
- Recursive definitions
- $V^\pi(s) = \sum_{a \in A} \pi(s,a) \left(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^\pi(s') \right)$
- $Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(s',a') Q^\pi(s',a')$
- **Called V-Value(s) (*state-value function*) and Q-Value(s) (*state-action value function*) respectively**
- Both calculate the expected rewards under a certain policy

Link between V^π and Q^π

- V and Q are interrelated
- $V^\pi(s) = \sum_{a \in A} \pi(s,a) Q^\pi(s,a)$
- $Q^\pi(s,a) = R(s,a) + \sum_{s' \in S} T(s'|s,a) V^\pi(s')$
- **V-values are defined on states, Q-values on policies!**

Optimal Policy and the Bellman Optimality Equation

- An optimal policy can be defined in terms of Q-values
- It is the policy that maximises Q values
- $V^*(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s')$
- $Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a' \in A} Q^*(s',a')$
- $\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$

Link between V^* and Q^*

- Again, they are interrelated
- $V(s)^* = \max_{a \in A} Q^*(s,a)$
- $Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s')$
- Let's assume that toon has another option
- She can go and buy and eat some meat with a reward of 1.5
- Or go down the fish route
- Write down the MDP
  - Find out the new Q and V values with:
  - Toon acting randomly on choosing a decision point
  - Toon choosing action *Go-Fishing*
  - Toon choosing action *Go-to-Restaurant*

Agents Revisited

- An Agent can be composed of a number of things
- A policy
- A Q-Value/and or V-Value Function
- A Model of the environment (the MDP)
- Inference/Learning Mechanisms
- ...
- An agent has to be able to *discover a policy* either on the fly or using Q-Values
- The Model/Q/V-Values serve as intermediate points towards constructing a policy
- Not all RL algorithms used that (but most do)...

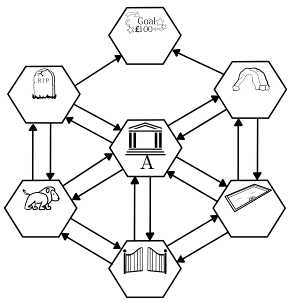
Simplifying assumptions

- Assume deterministic transitions
- Thus, taking an action on a state will lead only to ONE other possible state for some action a_c
  - $T(s'|s,a_i) = \begin{cases} 1 & \text{if } a_i = a_c \\ 0 & \text{otherwise} \end{cases}$
  - $V^*(s) = \max_{a \in A} [R(s,a) + \gamma V^*(s')]$
  - $Q^*(s,a) = R(s,a) + \gamma \max_{a' \in A} Q(s',a')$
- It is easier now to solve for problems that have loops in them
- We can also attempt to learn Q-Values without a model!
- All we need in order to find the optimal policy is $Q(s,a)$

Deterministic Q-Learning (1)

- The policy is deterministic from start to finish
- We will use $\pi(s) = \arg\max_{a \in A} Q(s,a)$ to denote the optimal policy
- The algorithm now is:
  - Initialise all $Q(s,a)$ to low values
  - Repeat:
    - Select an action a using an exploration policy
    - $Q(s,a) \leftarrow R(s,a) + \gamma \max_{a' \in A} Q(s',a')$
    - $s \leftarrow s'$
  - Also known as "Dynamic Programming", "Value Iteration"

An Example (1)



$$R(HALL, \textit{To-CAVE}) = 0$$

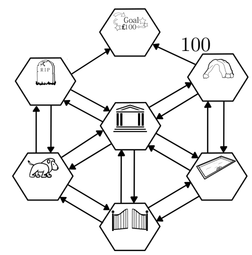
$$Q(CAVE, a) = 0 \text{ for all actions } a$$

An Example (2)

Next suppose the agent, now in state CAVE , selects action $To - GOAL$

$R(CAVE, \textit{To-GOAL}) = 100,\ Q(GOAL, a) = 0$ for all actions (there are no actions)

Hence $Q(CAVE, \textit{To-GOAL}) = 100 + \gamma * 0 = 100$



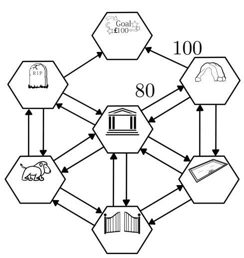
An Example (3)

Let's start at hall again and select the same action To-CAVE

$R(HALL, To - CAVE) = 0,\ Q(CAVE, GOAL) = 100$

$Q(CAVE, a) = 0$ for all other actions a

Hence $\max_{a \in A} Q(CAVE, a) = 100$, if $\gamma = 0.8$,
$Q(HALL, To - CAVE) = 0 + \gamma * 100 = 80$



Exploration / Exploitation

- How do we best explore?
- Choose actions at random - but this can be very slow
- $\epsilon - \textit{greedy}$ is the most common method
- Act ϵ-greedily
  - $\pi^{\epsilon}(s, a) = \begin{cases} a = \arg\max_{a \in A} Q(s, a) & \text{if } 1 - \epsilon + \epsilon/|A| \\ U_a & \text{otherwise} \end{cases}$
  - ϵ-greedy means acting greedily with probability $1 - \epsilon$, random otherwise
- When you are done, act greedily $\pi(s) = \arg\max_{a \in A} Q(s, a)$

Algorithms for non-deterministic settings

- What can we do if the MDP is not deterministic?
- Q-learning
  - $Qs, a) \leftarrow Q(s, a) + \eta \left[R(s, a) + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right]$
- SARSA(0)
  - $Q(s, a) \leftarrow Q(s, a) + \eta \left[R(s, a) + \gamma Q(s', a') - Q(s, a) \right]$
- SARSA(1)/MC,
  - $Q(s, a) \leftarrow Q(s, a) + \eta \left[\mathrm{v}_\tau - Q(s, a) \right]$
  - $\mathrm{v}_\tau \leftarrow R(s, a) + \gamma R(s', a') + ... \gamma^2 R(s'', a'') + \gamma^{\tau - 1} R(s^\tau, a^\tau)$
- η is a small learning rate, e.g., $\eta = 0.001$

SARSA vs Q-Learning vs MC

- MC: updated using the whole chain
  - Possibly works better when the markov property is violated
- SARSA: update based on the next action you actually took
  - On Policy learning
- Q-Learning: update based on the best possible next action
  - Will learn optimal policy even if acting off-policy

Monte Carlo Control (1)

- Remember Q is just a mean/average
- MC (Naive Version)
  - Start at any state, initialise $Q_0(s,a)$ as you visit states/actions
  - Act ϵ-greedily
- Add all reward you have seen so far to $\mathrm{v}_\tau^\mathrm{i} = R(s',a') + \gamma R(s'',a'') + \gamma^2 R(s''',a''') + \gamma^{\tau-1} R(s^\tau, a^\tau)$ for episode i
- $Q_n(s,a) = E_{\pi^\epsilon}[\mathrm{v}_\tau^\mathrm{i}] = \frac{1}{n}\sum\limits_{i=1}^{n} \mathrm{v}_\tau^\mathrm{i}$, where n is the times a state is visited

Monte Carlo Control (2)

- ϵ-greedy means acting greedily $1-\epsilon$, random otherwise
- Better to calculate mean incrementaly

$$
\begin{aligned}
Q_n(s,a) &= E_{\pi_n}[\mathrm{v}_\tau^\mathrm{i}] \\
Q_n(s,a) &= \frac{1}{n}\sum_{i=1}^{n} \mathrm{v}_\tau^\mathrm{i} \\
Q_n(s,a) &= \frac{1}{n}\left(\mathrm{v}_\mathrm{t}^1 + \mathrm{v}_\tau^2 .... \mathrm{v}_\tau^{\mathrm{n}-1} + \mathrm{v}_\tau^\mathrm{n}\right) \\
Q_n(s,a) &= \frac{1}{n}\left(\sum_{i=1}^{n-1} \mathrm{v}_\tau^\mathrm{i} + \mathrm{v}_\tau^\mathrm{n}\right)
\end{aligned}
$$

Monte Carlo Control (3)

by definition

$Q_{n-1}(s,a) = \frac{1}{n-1}\sum\limits_{i=1}^{n-1} \mathrm{v}_\tau^\mathrm{i} \implies (n-1)Q_{n-1}(s,a) = \sum\limits_{i=1}^{n-1} \mathrm{v}_\tau^\mathrm{i}$

$$
\begin{aligned}
Q_n(s,a) &= \frac{1}{n}\left((n-1)Q_{n-1}(s,a) + \mathrm{v}_\tau^\mathrm{n}\right) \\
Q_n(s,a) &= \frac{1}{n}\left(Q_{n-1}(s,a)n - Q_{n-1}(s,a) + \mathrm{v}_\tau^\mathrm{n}\right) \\
Q_n(s,a) &= \frac{Q_{n-1}(s,a)n}{n} + \frac{-Q_{n-1}(s,a) + \mathrm{v}_\tau^\mathrm{n}}{n} \\
Q_n(s,a) &= Q_{n-1}(s,a) + \frac{\overbrace{\mathrm{v}_\tau^\mathrm{n} - Q_{n-1}(s,a)}^{\textbf{MC-Error}}}{n}
\end{aligned}
$$

Monte Carlo Control (4)

- But π^n changes continuously, so the distribution of rewards is non-stationary

$$
\begin{aligned}
Q_n(s,a) &= Q_{n-1}(s,a) + \frac{1}{n}\left[\mathrm{v}_\tau^\mathrm{n} - Q_{n-1}(s,a)\right] \rightarrow \textbf{Bandit case} \\
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[\mathrm{v}_\tau^\mathrm{n} - Q_{n-1}(s,a)\right] \rightarrow \textbf{Full MDP case}
\end{aligned}
$$

- A Bandit can be seen as MDP with a chain of length one (i.e. s) - like the initial EagleWorld, η is a learning rate (e.g., 0.001)

Monte Carlo Control (5)

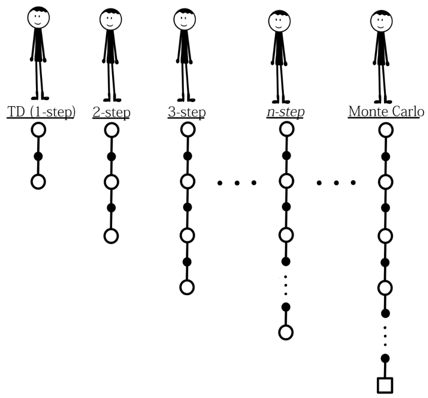
- Start at any state, initialise $Q_0(s,a)$ as you visit states/actions
- Act ϵ-greedily
- Wait until episode ends, i.e. a terminal state is hit - ϵ set to some low value, e.g., 0.1
- Add all reward you have seen so far to $\mathrm{v}_\tau^\mathrm{i} = R(s,a) + \gamma R(s',a') + ...\gamma^2 R(s'',a'') + \gamma^{\tau-1} R(s^\tau, a^\tau)$ for episode i
- $Q_n(s,a) = Q_{n-1}(s,a) + \eta\left[\mathrm{v}_\tau^\mathrm{n} - Q_{n-1}(s,a)\right]$

From monte carlo control to SARSA and Q-Learning

- With MC we update using the rewards from the whole chain
- Can we update incrementally?

$$
\begin{aligned}
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[\mathrm{v}_\tau^\mathrm{n} - Q_{n-1}(s,a)\right] \\
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[R(s,a) + \gamma R(s',a') + ...\gamma^2 R(s'',a'') + \gamma^{\tau-1} R(s^\tau,a^\tau) - Q_{n-1}(s,a)\right] \\
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[R(s,a) + \gamma(R(s',a') + ...\gamma R(s'',a'') + \gamma^{\tau-2} R(s^\tau,a^\tau)) - Q_{n-1}(s,a)\right] \\
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[R(s,a) + \gamma(\mathrm{v}_\tau^{\mathrm{n},(\mathrm{s}',\mathrm{a}')}) - Q_{n-1}(s,a)\right] \\
Q_n(s,a) &= Q_{n-1}(s,a) + \eta\left[R(s,a) + \gamma Q_{n-1}(s',a') - Q_{n-1}(s,a)\right]
\end{aligned}
$$

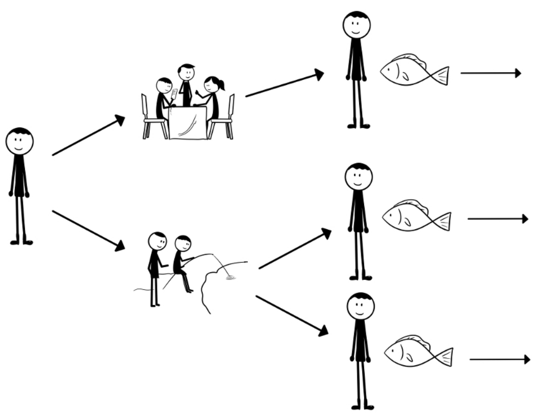
n-step returns



Let's go over the toon example, without a model

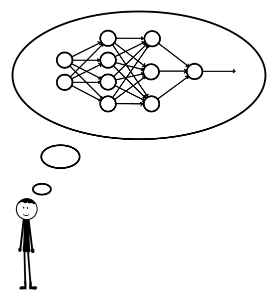
- $\epsilon - \textit{greedy}$, with $\epsilon = 0.1$

Model free toon



Function approximation (1)

- There is usually some link between states
- We can train function approximators incrementally to model $Q(s, a)$
- We now have $Q(s, a; \theta)$, where θ are the parameters



Function approximation (2)

- What are the links in states in Toon?
- Can we write down the Q-values in a more compact way?
- Let's devise a tree to do this
- Examples include linear function approximators, neural networks, n-tuple networks
- Not easy to do, few convergence guarantees
  - But with some effort, this works pretty well

Neural Networks and Function Approximation

- Most common modern function approximation scheme is neural networks
- Can approximate almost any function
- We had a series of recent advances
- Go
- Atari

Platforms

- Let's look at open AI gym
- A lot of modern work is a combination of RL with neural networks
- We have good libraries now

More on neural networks

- A function approximator loosely based on the brain
- Main idea - a graph of neurons
- Multiple layers
- Of a certain type of neurons
- Multiple types of training methods

Relationship to the rest of Machine Learning

- How can one learn a model of the world?
  - Possibly by breaking it down into smaller, abstract chunks
    - Unsupervised Learning
  - ... and learning what effects ones actions have the environment
    - Supervised Learning
- RL weaves all fields of Machine Learning (and possibly Artificial Intelligence) into one coherent whole
- The purpose of all learning is action!
  - You need to be able to recognise faces so you can create state
  - ... and act on it

Causality (bonus)

- We often colliqually say "A is caused by B"
- Can you discuss the meaning of this?

Counterfactuals

- If I take action a I land on state s
- What if I don't take action a?'
- "Experimenter forced you to pick up smoking" vs
- "Experimenter observed that you smoked"
- Will you get lung disease?
- The experimenter takes the actions vs observes

What is the link?

- Off-policy evaluation learning
- Let's see an example
  - Features are colour of hair, height, smoking
  - Reward is -1000 (lung disease), 1 (healthy)
- This would have been supervised learning if we knew the policy!
- Let's see a possible example of data
- Can you write down an example policy?

Conclusion

- RL is a massive topic
- We have shown the tip of iceberg
- Rabbit hole goes *deep* - both on the application level and the theory level

Further study (1)

- **Tom Mitchell, Chapter 13**
- David Silver's UCL Course: http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
  - Some ideas in these lecture notes taken from there
  - Probably the best set of notes there is on the subject
  - Online at http://www.machinelearningtalks.com/tag/rl-course/
- Reinforcement Learning, by Richard S. Sutton and Andrew G. Barto
  - Classic book
  - Excellent treatment of most subjects

Further Study (2)

- Artificial Intelligence: A Modern Approach by Stuart J. Russell and Peter Norvig
  - The Introductory A.I. Textbook
  - Chapters 16 and 21
- Algorithms for Reinfocement Learning by Csaba Szepesvari
  - Very "Mathematical", but a good resource that provides a very unified view of the field
- Reinforcement Learning: State-Of-The-Art by Marco Wiering (Editor), Martijn Van Otterlo (Editor)
  - Edited Volume