Reinforcement Learning

CITS3001 Algorithms, Agents and Artificial Intelligence

Tim French
Department of Computer Science and Software Engineering
The University of Western Australia

2219, Semester 2

Introduction

- We will define and motivate
  - Reinforcement learning vs. supervised learning
  - Passive learning vs. active learning
  - Utility learning vs. Q-learning
- We will discuss passive learning in known and unknown environments
  - With emphasis on various updating schemes, esp.
    - Adaptive dynamic programming
    - Temporal-difference learning
- We will discuss active learning
  - With emphasis on the issue of exploration vs. exploitation
- We will discuss generalisation of learning

Reinforcement Learning

THE UNIVERSITY OF WESTERN AUSTRALIA

- *Supervised learning* is where a learning agent is provided with input/output pairs on which to base its learning
- However learning is sometimes needed in less generous environments
  - No examples provided
  - No model of the environment
  - No utility function at all!
- In general, the less generous the environment, the more we need learning…

- The agent relies on *feedback* about its performance on order to assess its functionality
  - *e.g.* in chess you may be told only what a legal move is, and the result of each game
  - Try random moves and see what happens?
  - But even if you win, which moves were good?
- This is the basis of *reinforcement learning*
  - Use rewards to learn a successful agent function
  - In many complex environments, it's the only feasible learning option…

Aspects of reinforcement learning

THE UNIVERSITY OF WESTERN AUSTRALIA

- Is the environment *known*?
  - *e.g.* we may not know the transition model
  - An unknown environment must be learned, alongside the other required functionality
- Is the environment *accessible*?
  - An accessible environment is where the state that an agent is in can be identified from its percepts
  - In an inaccessible environment, the agent must remember information about its state, and recognise it by other means
- Are rewards given only in terminal states, or in every state?
  - *e.g.* only at the end of a game, or at other stages too?
- Are rewards given only "in bulk", or they are given for components of the utility?
  - *e.g.* dollar returns for a gambling agent, or hints ("nice move!")
- *All* feedback should be utilised!
  - Usually learning is hard!

Passive learning vs active learning

THE UNIVERSITY OF WESTERN AUSTRALIA

- One fundamental distinction is between passive and active learning
- *Passive learning*: given a fixed agent function, learn the utilities of that function in the environment
  - Essentially watch the world go by, and assess how well things are going
- *Active learning*: no fixed function, agent must select actions using what has been learned so far
  - *i.e.* learn the agent function too
  - Use a problem generator to (systematically?) explore the environment, and learn what options exist
- Passive learning agents may be associated with a "higher-level" intelligence (a designer?) to suggest different functions to try
  - Active learning agents try to do the entire job as one

Utility learning vs Q-learning

THE UNIVERSITY OF WESTERN AUSTRALIA

- A second fundamental distinction is between learning utilities, and simply(?) learning actions

- *Utility learning*: agent learns state utilities, then (subsequently) selects actions that maximise expected utility
  - Needs to know where actions can lead, so must have (or learn) a model of the environment
  - But this “deep” knowledge can mean faster learning
  - *cf*. value iteration

- *Q-learning*: agent learns an *action-value* function, *i.e.* the expected utility of taking an action in a state
  - Doesn’t need to know where actions lead, just learns how good they are
  - “Shallow” knowledge can restrict the ability to learn
  - *cf*. policy iteration

Passive learning in a known environment

- Assume:
  - Accessible environment
  - Actions are pre-selected for the agent
  - Effects of actions are known
- The aim is to learn the utility function of the environment
- The agent executes a set of *trials* in the environment
  - In each trial, the agent moves from the start state to a terminal state according to its given function
  - Its percepts identify both the current state and the immediate reward

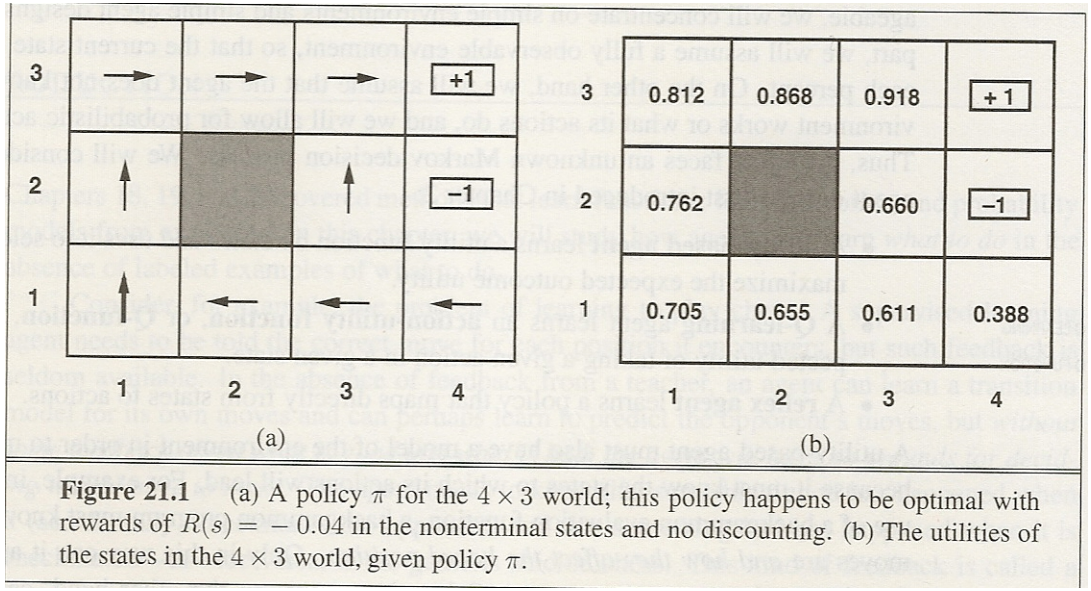


Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

Passive learning continued

- An example trial would be

$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$

- This trial generates a sample utility *for each state the agent passes through*
  - Assuming an additive utility function, and working backwards
- A set of trials generates a set of samples for each state in the environment
- In the simplest model, we just maintain an average of the samples observed for each state
- With enough trials, these estimates will converge on the true utilities

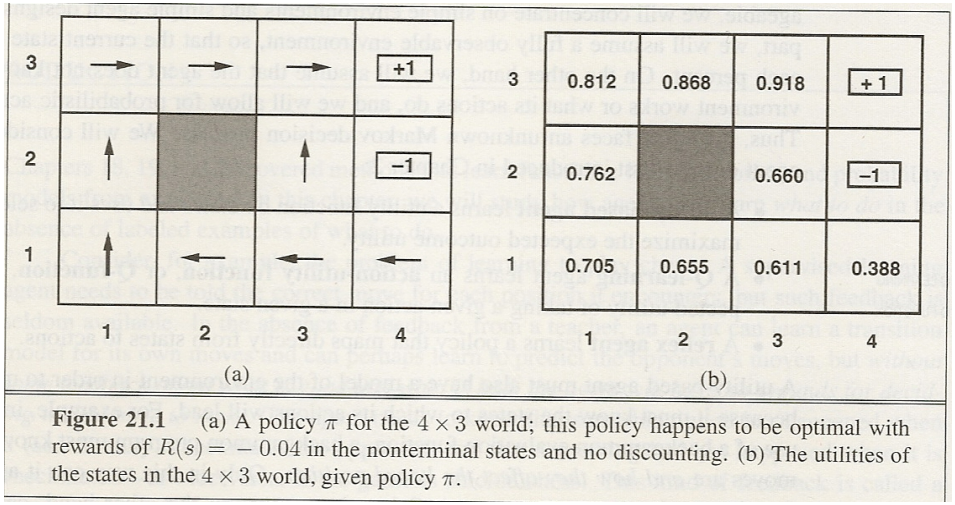


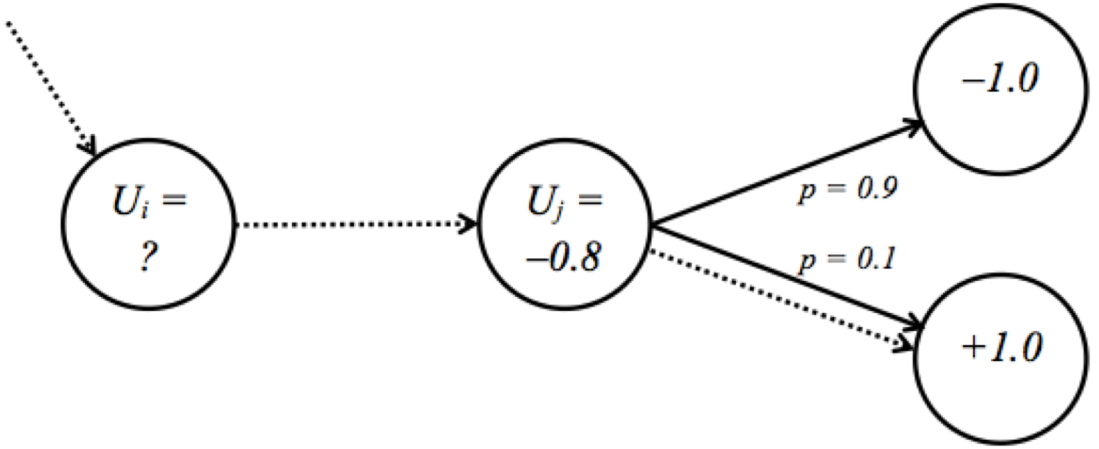
Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	(4,3)
0.72	0.76 0.84	0.80 0.88	0.92	0.96	1.00

Updating

- A key to reinforcement learning is the *update function*
- The Bellman equation (and our intuition) tells us that states' utilities are not independent.
- (The estimate of) U_j has been set by previous trials
  - Represented by the solid lines
- U_i is set by the new trial
  - Represented by the dotted line

- The initial estimate for U_i will be highly positive
  - But the link to U_j tells us it should be negative…
  - And this is U_i's only known link at the time
- This estimate will be corrected with sufficient trials
  - But with naïve updating, convergence will be slow



Adaptive dynamic programming

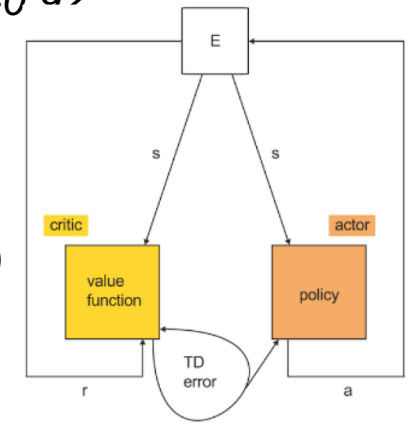
- One updating scheme that tries to learn faster by exploiting these connections is *ADP*
- As discussed in Lecture 9, the (true) utility of a state is a probability-weighted average of its successors, plus its own reward
  - In a passive situation:

$$U_i = R_i + \sum_j M_{ij}^a U_j$$

- ADP needs enough trials to learn the transition model of the environment
  - *i.e.* it needs to learn M_{ij}^a
  - It can estimate this from experience, *e.g.* if (3,1) → (3,2) occurs 20% of the time
- Then learning reduces to the *value determination* process
  - Page 7 of Lecture 9
- ADP is a good benchmark for learning
  - But as discussed previously, for *n* states it generates *n* simultaneous equations
  - Thus the process is often intractable

Temporal Difference Learning

- *TDL* tries to get the best of both worlds
  - Exploit the constraints between states
  - But without solving for all states simultaneously
- The idea is to use the *observed* transitions to adjust utilities *locally* to be consistent with Bellman
  - *e.g.* say in a particular trial, we transition from (1,3) to (2,3), and that $U_{2,3}=0.92$
  - If this is correct, then $U_{1,3}=0.92-0.04=0.88$
  - So if $U_{1,3}\neq 0.88$, move it towards that value
- But don't over-commit!
  - $U_{2,3}$ may not be correct yet , There will probably be other paths out of (1,3)



Hence TDL uses the update

$$U_i \leftarrow U_i + \alpha(R_i + U_j - U_i)$$

- α is called the learning rate
  - Higher values of α mean we change *Ui* more
  - $\alpha=0$ does no update; $\alpha=1$ uses the new value
- Sometimes α is set to decrease over time
  - Basically as the number of observations goes up, we trust the current estimate more
- The average value of *Ui* converges eventually
  - Different transitions will contribute in proportion to how often they happen

ADP vs TDL

- TDL can be seen as a crude (but efficient) approximation to ADP
- Conversely, ADP can be seen as a version of TDL using “pseudo-experience”, derived from the transition model

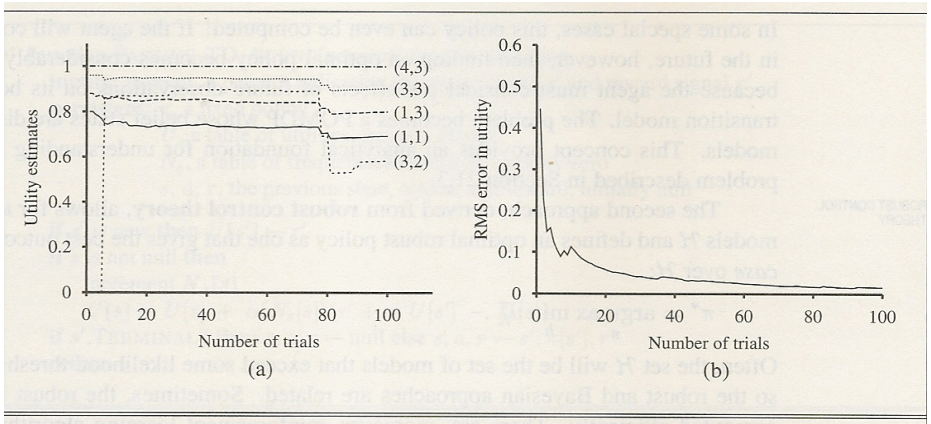


Figure 21.3 The passive ADP learning curves for the 4×3 world, given the optimal policy shown in Figure 21.1. (a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice the large changes occurring around the 78th trial—this is the first time that the agent falls into the -1 terminal state at (4,2). (b) The root-mean-square error (see Appendix A) in the estimate for $U(1,1)$, averaged over 20 runs of 100 trials each.

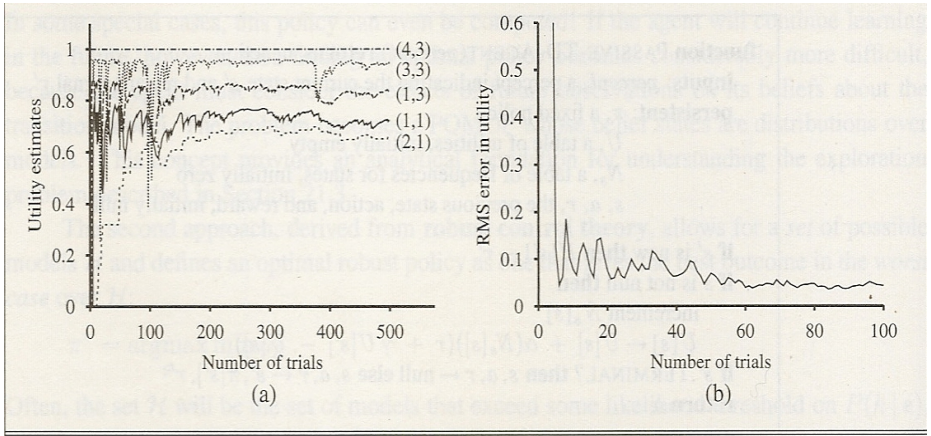


Figure 21.5 The TD learning curves for the 4×3 world. (a) The utility estimates for a selected subset of states, as a function of the number of trials. (b) The root-mean-square error in the estimate for $U(1,1)$, averaged over 20 runs of 500 trials each. Only the first 100 trials are shown to enable comparison with Figure 21.3.

Active learning

- In active learning, the agent not only needs to learn utilities, it also must select actions
  - Thus the agent needs to evolve its performance element by exploring its options
  - To do this it needs a problem generator
- The former requires that
  - For each state, the agent maintains an estimated utility for each action separately
    - 3D data instead of 2D data
  - If using ADP, the agent uses the active version of the Bellman equation to select actions
    - Rather than simply following a fixed policy
  - But TDL requires no change to the update scheme
- The latter requires balancing present vs. future rewards

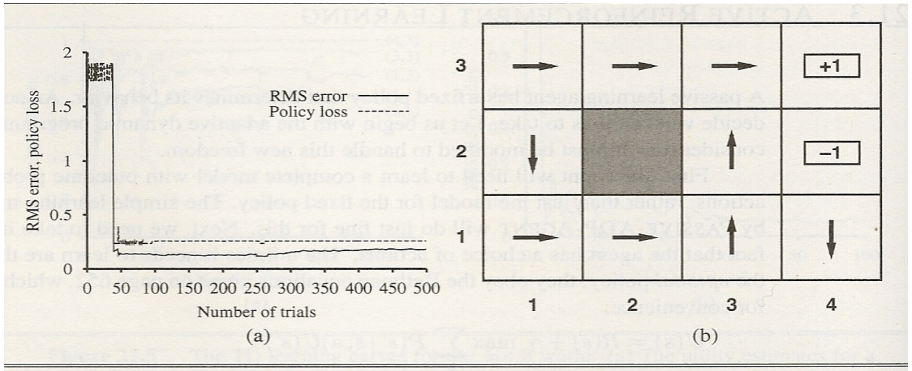
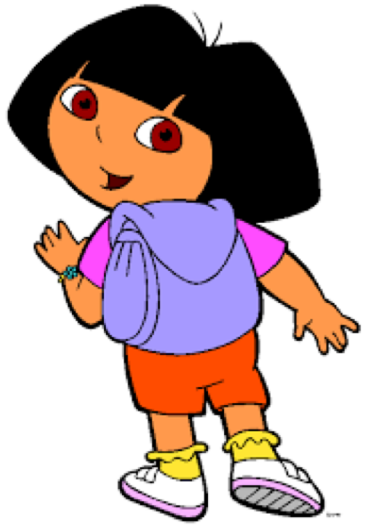


Figure 21.6 Performance of a greedy ADP agent that executes the action recommended by the optimal policy for the learned model. (a) RMS error in the utility estimates averaged over the nine nonterminal squares. (b) The suboptimal policy to which the greedy agent converges in this particular sequence of trials.

Exploration vs exploitation



- In active learning, the agent must select actions that both
  - Enable it to perform well in its environment
  - Enable it to learn about its environment
- So it needs to balance
  - Getting good rewards on the current sequence
    - *Exploitation* for the immediate good
  - Observing new percepts, and thus improving rewards on future sequences
    - *Exploration* for the long-term good
- This is a general, non-trivial problem
  - Insufficient exploration will mean that the agent gets "stuck in a rut". *Greedy behaviour settles for the first good solution that it finds.*
  - Insufficient exploitation will mean that the agent never gets anything done. *Whacky behaviour (probably) finds all solutions, but never knows it!*
  - Not just a problem for artificial agents!
- The fundamental problem is that at any moment, very likely the agent's learned model differs from the true model

Greedy in the limit of infinite exploration

THE UNIVERSITY OF WESTERN AUSTRALIA

- The “optimal” exploration policy is known as *GLIE*
  - Start whacky, get greedier
- The fundamental idea is to give weight to actions that have not been tried often, whilst also avoiding actions with low utilities
  - *Unknown* preferred to *good* preferred to *bad*
  - Obviously it’s not applicable in all environments!
- One scheme uses an *optimistic prior*
  - Assume initially that everything is good
- Let U_i^+ be the initial estimate, and N_i^a be the number of times the agent has performed Action *a* in State *I*

$$U_i^+ \leftarrow R_i + max_a f(\sum_j M_{ij}^a U_j^+, N_i^a)$$

  - Where *f(u,n)* is the *exploration function*
- Using *U*+ on the RHS of the equation propagates the tendency to explore
  - Regions near the start are likely to be explored first
  - More-distant regions are likely to be sparsely-explored, so we need to make them look good

GLIE cont.

- *f(u,n)* determines the trade-off between "greed" and "curiosity"
  - Should increase with *u* and decrease with *n*, where R^+ is the optimistic prior, and N_e is the minimum number of tries for each action

$$f(u,n) = \begin{cases} R^+, & if \quad n < N_e \\ u, & otherwise \end{cases}$$

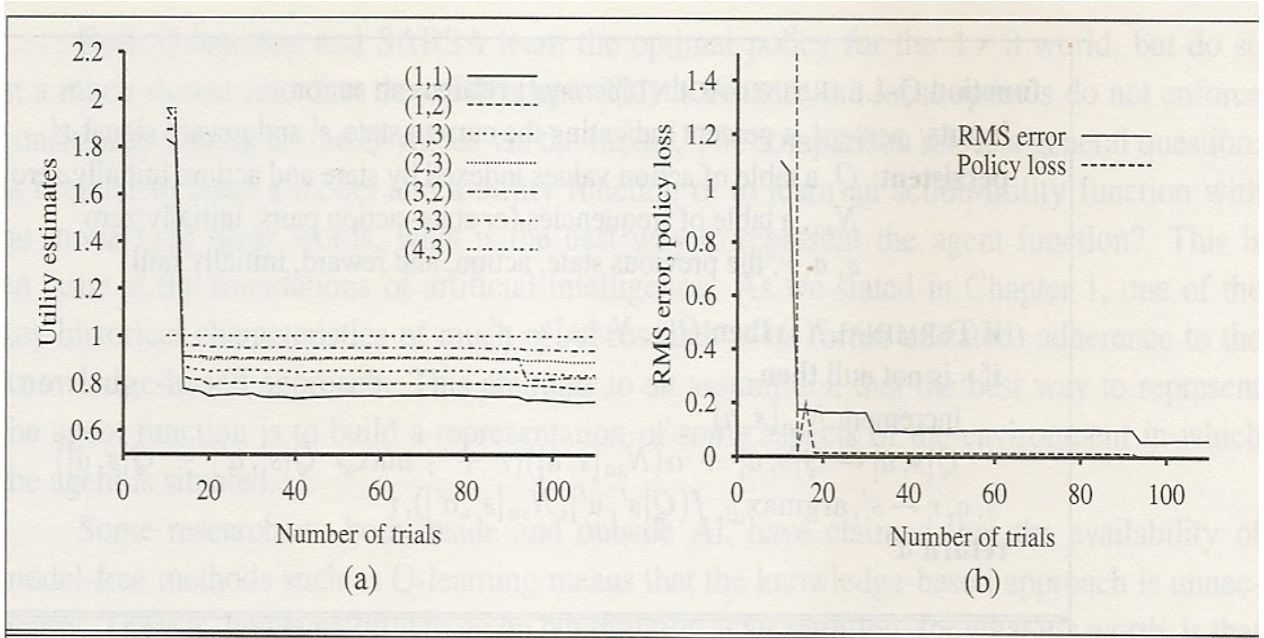


Figure 21.7 Performance of the exploratory ADP agent. using $R^+ = 2$ and $N_e = 5$. (a) Utility estimates for selected states over time. (b) The RMS error in utility values and the associated policy loss.

- For the above problem
  - Best policy loss for pure greedy behaviour ≈ 0.25
  - For pure whacky behaviour ≈ 2.3

Q-learning

- *Q-learning* basically means instead of learning the overall utility of State *i*, we learn separately the utility of taking each action *a* that is available in *i*
- The principal advantage is that we no longer need to know the transition model
  - We don't need to know explicitly what effects an action can have, just how good it is
- If Q_i^a is the utility of doing Action *a* in State *i*:

$$U_i = max_a Q_i^a$$

- If we want to apply ADP to Q-learning, we still need to learn the transition model
  - ADP updates explicitly require the model

$$Q_i^a \leftarrow R_i + \sum_j M_{ij}^a max_b Q_j^b$$

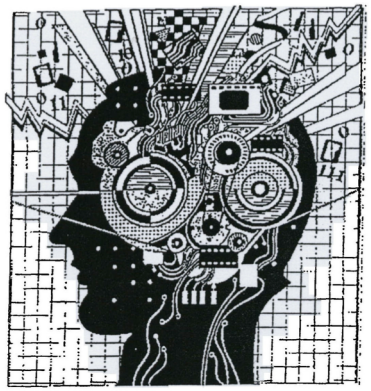
- But applying TDL is much more natural

$$Q_i^a \leftarrow Q_i^a + \alpha(R_i + max_b Q_j^b - Q_i^a)$$

Q-Learning

THE UNIVERSITY OF WESTERN AUSTRALIA

- But learning via Q-values is still usually slow
  - Because they do not enforce consistency between states' (or actions') utilities
  - So why is it interesting?
- Mostly for philosophical reasons
- Does an intelligent agent really need to incorporate a model of its environment to learn anything?
  - If so, how can we ever develop a universal agent?
  - Some biologists say that our DNA can be interpreted as a description of the environment(s) in which we evolved…
- Does the availability of model-free techniques like Q-learning offer hope?
- When we discussed the nature of AI, we said we would take essentially an engineering viewpoint
  - "Can we develop systems that do useful stuff?"
  - And of course this is the best way to get a job ☺
- But bear in mind that there may be bigger goals too



Generalization in learning

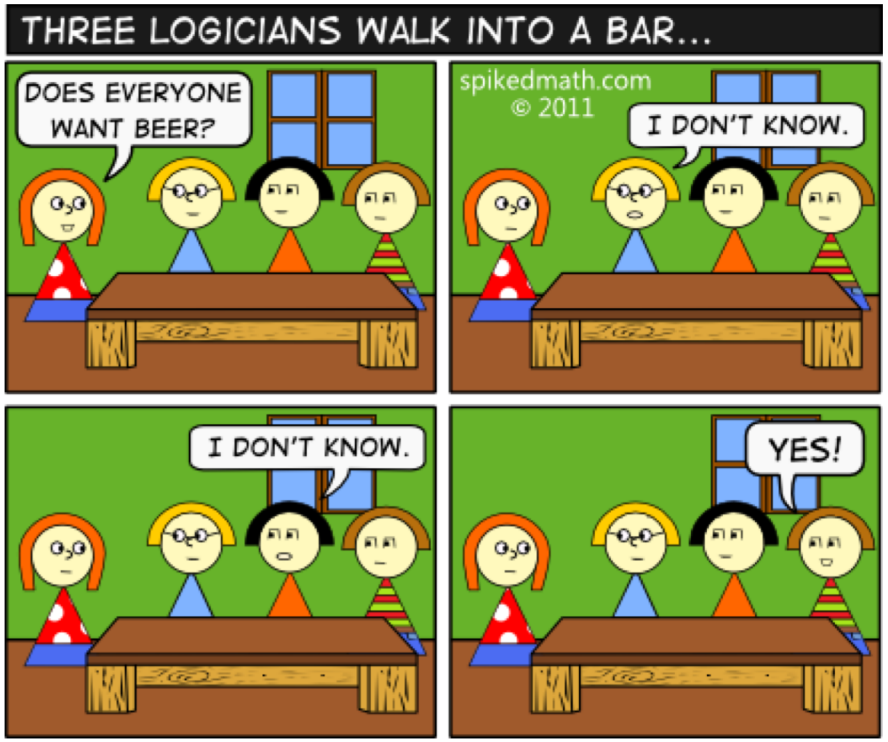
THE UNIVERSITY OF WESTERN AUSTRALIA

- Ultimately, neither supervised learning nor reinforcement learning can expose an agent to all of the states it will ever need to deal with
  - Chess has over 10^{40} states: what proportion of those has Magnus Carlsen ever seen?
  - We need to *generalise* from what we learn about seen states to cope with unseen states
- Agents require an implicit, compact representation
  - *e.g.* weighted linear sum of features
  - Colossal compression ratio
  - Enables generalisation
- States are related to each other via their shared features/properties/attributes
- The hypothesis space for the representation must be rich enough to allow for the "correct answer"
  - *e.g.* can the "true" utility function for chess really be represented in 10–20 numbers!?
- The current world champion, aged 23, peak rating 2,882 – the highest ever for a human.

Trade offs in representation

THE UNIVERSITY OF WESTERN AUSTRALIA

- Typically, a larger/richer hypothesis space means
  - There is more chance that it includes a suitable function
  - The space is more sparse
  - The function requires more memory
  - More examples are needed for learning
  - Convergence will be slower
  - It is harder to learn online vs. offline
- As often happens, the best answer is highly problem-dependent
- That's one reason these skills are valuable!



- Next up, Logical Agents!