Learning

Paolo Turrini

Department of Computing, Imperial College London

Introduction to Artificial Intelligence

The lectures

- The agent and the world (**Knowledge Representation**)
  - Actions and knowledge
  - Inference
- Good decisions (**Risk and Decisions**)
  - Chance
  - Gains
- Good decisions in time (**Markov Decision Processes**)
  - Chance and gains in time
  - Patience
  - Finding the best strategy
- Learning from experience (**Reinforcement Learning**)
  - Finding a reasonable strategy

Learning

'Rationality means the best we can do, given what we know. But how much should we know?'

Outline

- Reinforcement Learning, the very basics
- Conclusion

The book

Stuart Russell and Peter Norvig
Artificial Intelligence: a modern approach
Chapter 21

From the book

Imagine playing a new game whose rules you don't know; after a hundred moves, your opponent says, 'You lose.' This is reinforcement learning in a nutshell.


ALPHAGO
02:00:00
LEE SEDOL
01:59:41
AlphaGo
Lee Sedol
Google
"I hope this advanced technology will be used for useful things," Mr Lee told the BBC

Complicated positions

Game size	Board size N	3^N	Percent legal	Maximum legal game positions (A094777)[10]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}	1.196%	$2.08168199382\times10^{170}$
21×21	441	2.57×10^{210}		

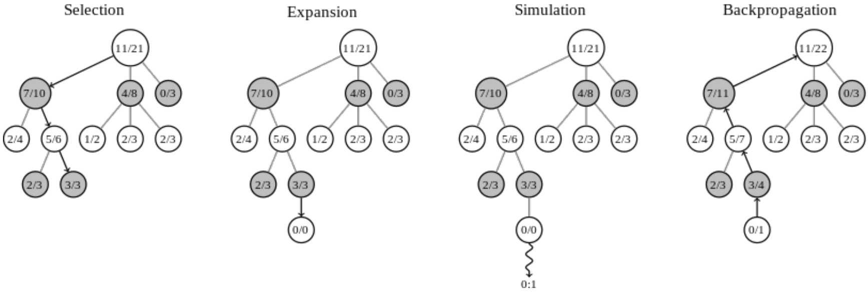
The complexity of Go

Reinforcement Learning in games

- We cannot possibly calculate everything
- We need an assessment of the value of:
  - intermediate positions
  - and moves in general

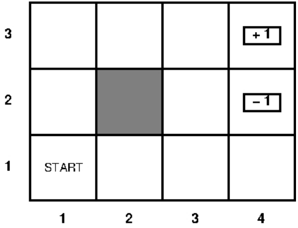
We cannot always hope to find the best continuation

Learning, by trial and error



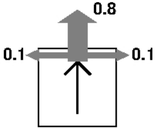
Monte Carlo Tree Search

The world



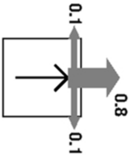
- Begin at the start state
- The game ends when we reach either goal state $+1$ or -1
- Rewards: $+1$ and -1 for terminal states respectively, -0.04 for all others

Full observability



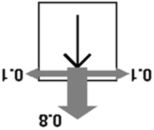
Stochastic actions (possibly different at each state!), four directions.

Full observability



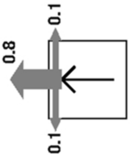
Stochastic actions (possibly different at each state!), four directions.

Full observability



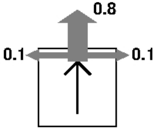
Stochastic actions (possibly different at each state!), four directions.

Full observability



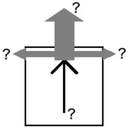
Stochastic actions (possibly different at each state!), four directions.

Full observability



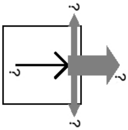
Stochastic actions (possibly different at each state!), four directions.

Partial observability



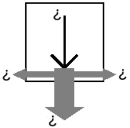
Stochastic actions (possibly different at each state!), four directions.

Partial observability



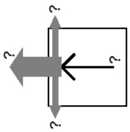
Stochastic actions (possibly different at each state!), four directions.

Partial observability



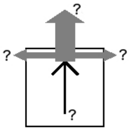
Stochastic actions (possibly different at each state!), four directions.

Partial observability



Stochastic actions (possibly different at each state!), four directions.

Partial observability



Stochastic actions (possibly different at each state!), four directions.

Assumptions

This is what is known (by the agent) about the environment

- Partially observable (we know where we are, not where we will end up being)
- Markovian (past doesn't matter)
- Stochastic actions (we are not in full control of our choices)
- Discounted rewards (we might be more or less patient)

Passive and Active RL

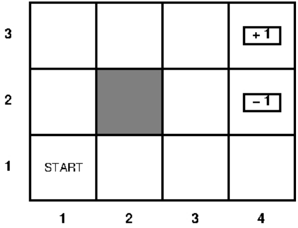
Passive reinforcement learning:

- I have a policy
- I don't know the probabilities
- I don't know the values of states
- I don't know the value of actions

Active reinforcement learning:

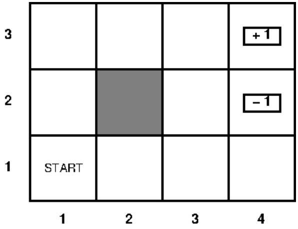
- I don't even have a policy

Passive Reinforcement Learning



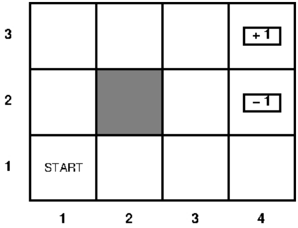
- I don't know the values nor the rewards

Passive Reinforcement Learning



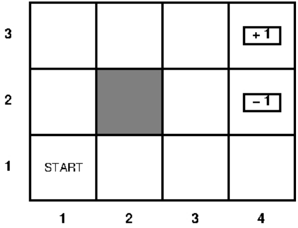
- I don't know the values nor the rewards
- I don't know the probabilities

Passive Reinforcement Learning



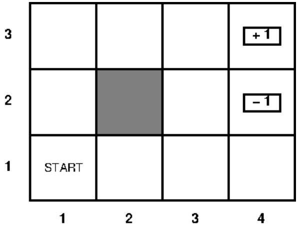
- I don't know the values nor the rewards
- I don't know the probabilities
- I'm gonna play anyway

Passive Reinforcement Learning



The plan: I execute a series of trials until the end states, just like Monte-Carlo Tree Search!

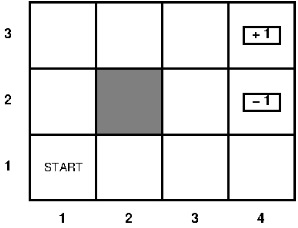
Passive Reinforcement Learning



Remember: the expected utility is the expected sum of discounted rewards under the policy

Assume: $\gamma = 1$, just to make things simple

Passive Reinforcement Learning



Suppose I get these trials:

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

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

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

Direct value estimation

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

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

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

The value of a state is the expected total reward from that state.

Direct value estimation

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

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

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

The value of a state is the expected total reward from that state.

Idea: Frequency is the key! Each trial provides a sample of the expected rewards for each state visited.

Direct value estimation

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

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

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

Direct value estimation

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

The first trial provides:

- a sample total reward of 0.72 for state $[1,1]$
- two samples of 0.80 and 0.88 for $[3,1]$
- etc.

We keep putting together these rewards-to-go.
In the end, **with infinitely many trials**, it gets the right values.

Direct value estimation

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

But we can do so much better, as **values are not independent!**

$$v^{\pi}(s) = r(s) + \gamma \sum_{s'} P(s' \mid (s, \pi(a)))v^{\pi}(s')$$

is the Bellmann equation for a fixed policy.

Direct value estimation

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

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

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

- The second of the three trials reaches state $[2,3]$, which has not previously been reached.
- The transition next reaches $[3,3]$, which is known from the first trial to have high utility.

Direct value estimation

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

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

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

- The Bellmann equation suggests $[2,3]$ is likely to have a high utility.
- Direct value estimation learns nothing, so the algorithm converges very slowly.

Temporal difference learning

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

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

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

Now let's try and 'adjust' the values we obtain using Bellmann equation:

Temporal difference learning

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

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

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

Now let's try and 'adjust' the values we obtain using Bellmann equation:
Suppose $v^{\pi}([3,1]) = 0.84$ and $v^{\pi}([3,2]) = 0.92$

Temporal difference learning

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

Now let's try and 'adjust' the values we obtain using Bellmann equation:
Suppose $v^\pi([3,1]) = 0.84$ and $v^\pi([3,2]) = 0.92$
If this transition occurred all the time we would expect:
$v^\pi([3,1]) = -0.04 + v^\pi([3,2])$
which is 0.88...

Temporal difference learning

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

Now let's try and 'adjust' the values we obtain using Bellmann equation:
Suppose $v^{\pi}([3,1]) = 0.84$ and $v^{\pi}([3,2]) = 0.92$
If this transition occurred all the time we would expect:
$v^{\pi}([3,1]) = -0.04 + v^{\pi}([3,2])$
which is 0.88...so we adjust it!

Temporal difference learning

When a transition occurs from state s to state s' we apply the following update:

$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s))$$

where $\alpha \in [0, 1]$ is a learning parameter: how much we value the incoming information.

Temporal difference learning

When a transition occurs from state s to state s' we apply the following update:

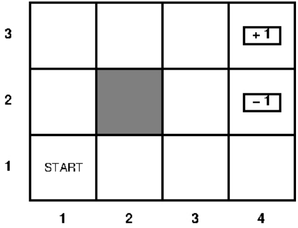
$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s))$$

where $\alpha \in [0, 1]$ is a learning parameter: how much we value the incoming information.

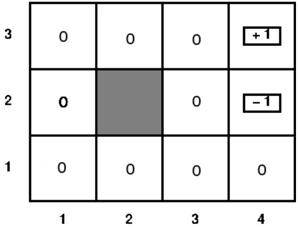
α can be the inverse of the number of times we visited a state: the more we visited, the less we want to learn.

Notice: rare transitions? well they are rare.

TDL in action



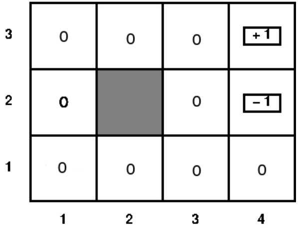
TDL in action

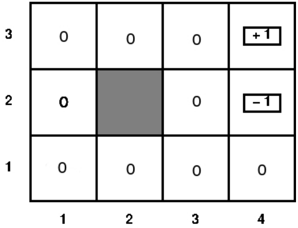


Initialise the values, for:

- $\gamma = 1$
- deterministic agent
- $\alpha = \frac{1}{n+1}$ where n is the number of times we visited a state
- $r = 0$ everywhere but the terminal states

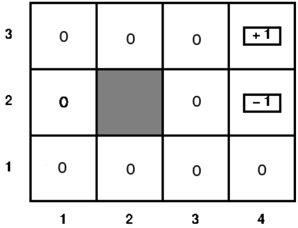
TDL in action

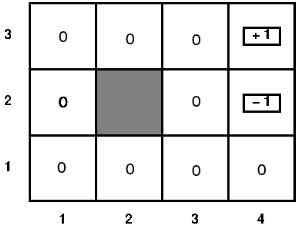




Suppose we can walk $(Up, Up, Right, Right, Right)$.

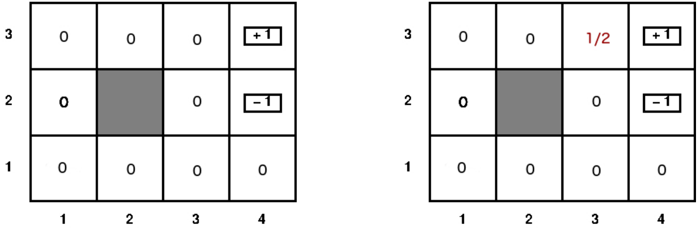
TDL in action





Suppose we can walk $(Up, Up, Right, Right, Right)$.
So let's walk repeatedly until the end state.

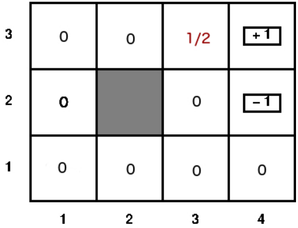
TDL in action

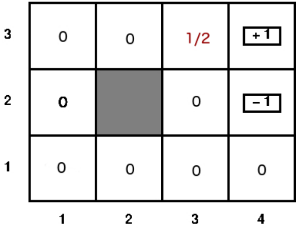


Apply the update to states, as you walk along:

$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s))$$

TDL in action

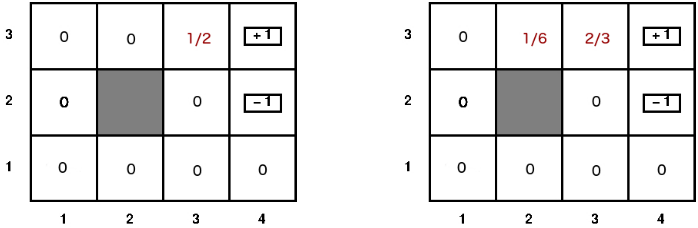




I keep walking the same way...

$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s))$$

TDL in action



Again $(Up, Up, Right, Right, Right)$....

$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s))$$

Passive Reinforcement Learning

- We have a policy which we follow;
- We backpropagate the value with a Bellmann-like adjustment;
- We can use a learning rate, depending on our confidence.

Active Reinforcement Learning

Now we start without a fixed policy...

Active Reinforcement Learning

Now we start without a fixed policy...

What the agent needs to learn is the values of the optimal policy

$$v(s) = r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a))v(s')$$

Active Reinforcement Learning

Now we start without a fixed policy...

What the agent needs to learn is the values of the optimal policy

$$v(s) = r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a)) v(s')$$

Important: we can't stick to our (locally optimal) habits, we need to try new stuff!

Exploration vs Exploitation

Q-learning

$$Q(s, a)$$

the value of performing action a in state s

Q-learning

$$Q(s, a)$$

the value of performing action a in state s

$$Q(s, a) = r(s) + \gamma \sum_{s'} P(s'|(s, a)) \max_{a'} Q(s', a')$$

Q-learning

$$Q(s, a)$$

the value of performing action a in state s

$$Q(s, a) = r(s) + \gamma \sum_{s'} P(s'|(s, a)) \max_{a'} Q(s', a')$$

$$v(s) = \max_{a} Q(s, a)$$

is the value of performing action a in state s

Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

It's a temporal difference learning, without fixed policy!

The Maze

- A 2×3 grid world

The Maze

- A 2×3 grid world
- A pit, an exit and some walls are known in this grid world, but their locations are unknown

The Maze

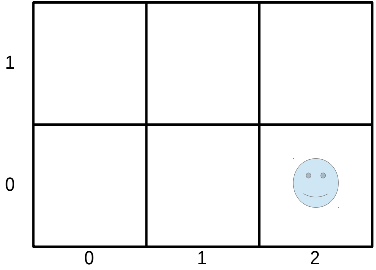
- A 2×3 grid world
- A pit, an exit and some walls are known in this grid world, but their locations are unknown
- Arrive at the exit: win; fall in the pit: die; hit a wall, suffer

The Maze

- A 2×3 grid world
- A pit, an exit and some walls are known in this grid world, but their locations are unknown
- Arrive at the exit: win; fall in the pit: die; hit a wall, suffer
- Goal: Get out of this maze (i.e. safely arrive at the exit) as quickly as possible

The Maze

- A 2×3 grid world
- A pit, an exit and some walls are known in this grid world, but their locations are unknown
- Arrive at the exit: win; fall in the pit: die; hit a wall, suffer
- Goal: Get out of this maze (i.e. safely arrive at the exit) as quickly as possible

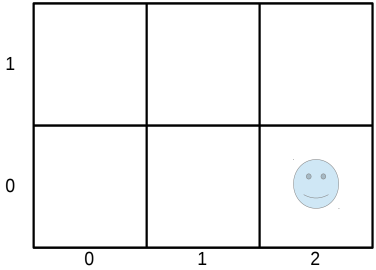


RL components in this problem

- **State**: The agent's current location
- **Action**: *LEFT, RIGHT, UP, DOWN*
- **Environment Dynamics**:
  - Collusion results in no movement
  - otherwise, move one square in the intended direction
- **Rewards**:
  - normal move: -1
  - hit a wall: -10
  - die: -100
  - exit: +100
- **Our Goal**: find the best route to exit

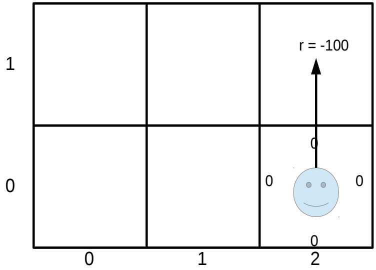
Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0



Applying Q-Learning to The Maze Problem

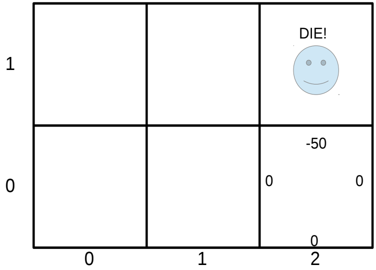
- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0
- Choose *UP*, and receive -100



Applying Q-Learning to The Maze Problem

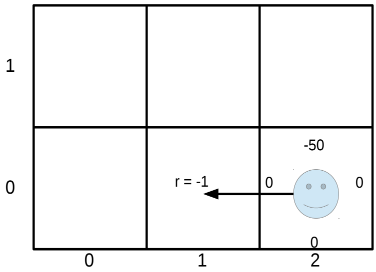
- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0
- Choose *UP*, and receive -100
- update Q-value:

$$Q([0,2], UP)$$
$$= (1 - 0.5) \times 0 +$$
$$0.5 \times (-100 + 0.9 \times 0)$$
$$= -50$$



Applying Q-Learning to The Maze Problem

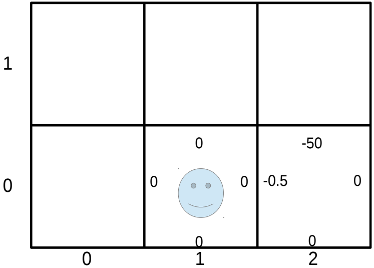
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



Applying Q-Learning to The Maze Problem

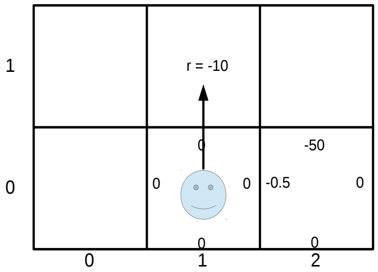
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

$$Q([0,2], LEFT)$$
$$= (1 - 0.5) \times 0 +$$
$$0.5 \times (-1 + 0.9 \times 0)$$
$$= -0.5$$



Applying Q-Learning to The Maze Problem

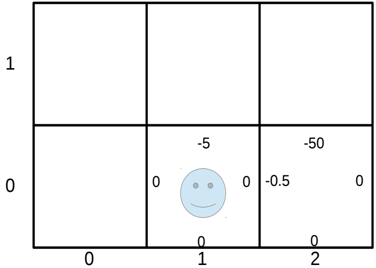
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10



Applying Q-Learning to The Maze Problem

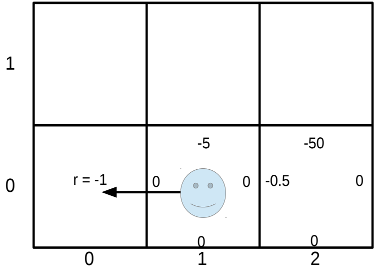
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10
- update Q-value:

$$Q([0,1], UP)$$
$$= (1-0.5) \times 0 +$$
$$0.5 \times (-10 + 0.9 \times 0)$$
$$= -5$$



Applying Q-Learning to The Maze Problem

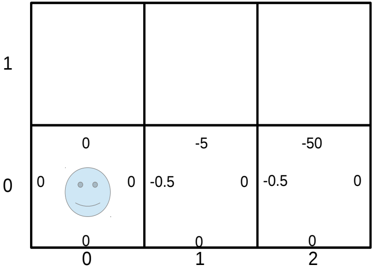
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



Applying Q-Learning to The Maze Problem

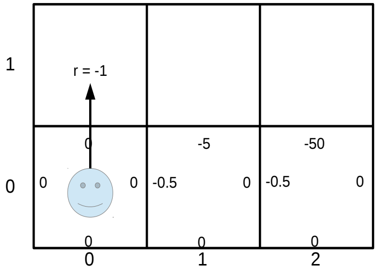
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

$$
\begin{aligned}
& Q([0,1], \mathit{LEFT}) \\
& = (1 - 0.5) \times 0 + \\
& \quad 0.5 \times (-1 + 0.9 \times 0) \\
& = -0.5
\end{aligned}
$$



Applying Q-Learning to The Maze Problem

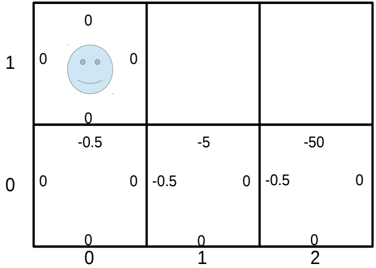
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -1



Applying Q-Learning to The Maze Problem

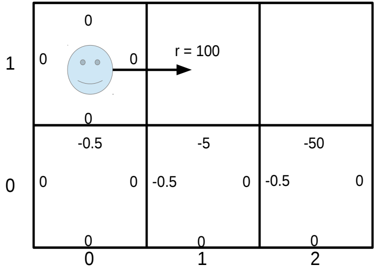
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -1
- update Q-value:

$$Q([0,0], UP)$$
$$= (1 - 0.5) \times 0 +$$
$$0.5 \times (-1 + 0.9 \times 0)$$
$$= -0.5$$



Applying Q-Learning to The Maze Problem

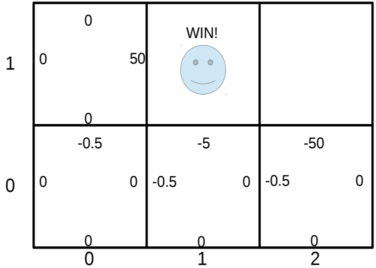
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *RIGHT*, and receive 100



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *RIGHT*, and receive 100
- update Q-value:

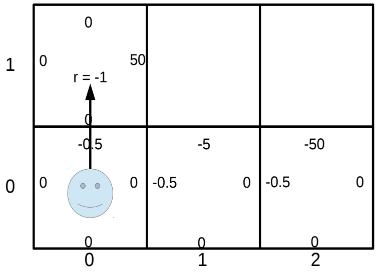
$$Q([0,1], RIGHT)$$
$$= (1 - 0.5) \times 0 +$$
$$0.5 \times (100 + 0.9 \times 0)$$
$$= 50$$



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- The next time agent visits [0,0] and performs *UP*:

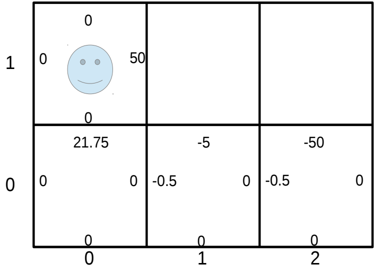
$$Q([0,0], UP)$$
$$= (1 - 0.5) \times (-0.5) +$$
$$0.5 \times (-1 + 0.9 \times 50)$$
$$= 21.75$$



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- The next time agent visits [0,0] and performs *UP*:

$$Q([0,0], UP)$$
$$= (1 - 0.5) \times (-0.5) +$$
$$0.5 \times (-1 + 0.9 \times 50)$$
$$= 21.75$$



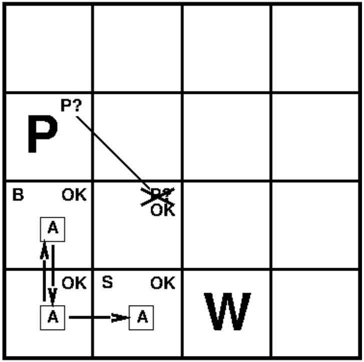
Property of Q-Learning

- Quick learning speed.
- Model-free: no need to explicitly compute probabilities, or record trajectory.
- Guarantee to converge.

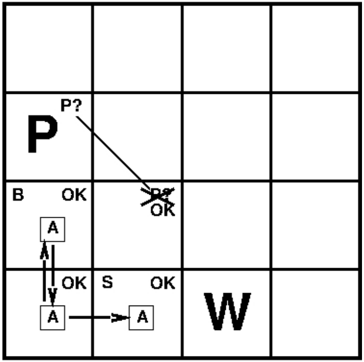
The learning parameters in Q-Learning

- α:
  - Learning step
  - balance between existing experiences (weight: $1 - \alpha$) and new observations (weight: α)
- γ:
  - Future discount
  - balance between current reward (weight: 1) and next N step's reward (weight: γ^N)
- ϵ:
  - indicating how 'bold' the agent is
  - balance between **exploitation** (take greedy action, $1 - \epsilon$ chance) and **exploration** (take random action, ϵ chance)

Domain Knowledge and RL



Domain Knowledge and RL



Logic can save us a lot of time

Conclusion

The agent's mind is a Knowledge Base

- What we TELL the knowledge base
- What we ASK the knowledge base

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time
   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

We can add new facts...

"Joffrey Baratheon is a king"

We can apply logical tools and infer new facts

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

For example,

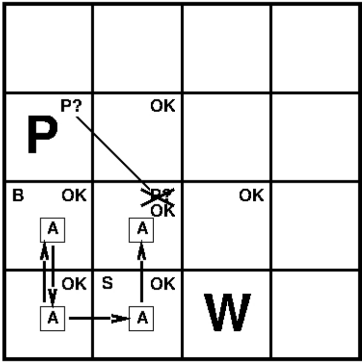
$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x)\\ Rich(Berlusconi)\end{array}}{Unhappy(Berlusconi)}$$

with $\theta = \{x/Berlusconi\}$

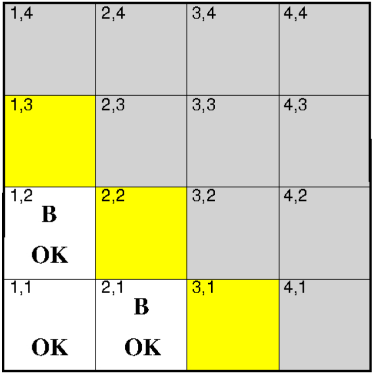
Apply resolution steps to $CNF(KB \wedge \neg\alpha)$

Knowledge helps us get to our goal

- The further we go the more we know



However knowledge is not enough



$P_{ij} = true$ iff $[i, j]$ contains a pit
$B_{ij} = true$ iff $[i, j]$ is breezy
Include only $B_{1,1}, B_{1,2}, B_{2,1}$ in the probability model

But we can use probabilistic assertions

A and B are independent iff

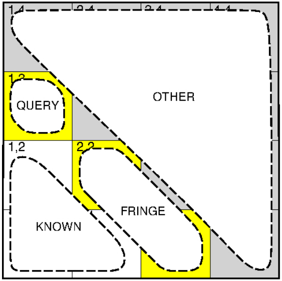
$\mathbf{P}(A|B) = \mathbf{P}(A)$ or $\mathbf{P}(B|A) = \mathbf{P}(B)$ or $\mathbf{P}(A, B) = \mathbf{P}(A)\mathbf{P}(B)$

$P(cavity|\text{Cristiano Ronaldo scores}) = P(cavity)$

$P(\text{Cristiano Ronaldo scores}|cavity) =$
$P(\text{Cristiano Ronaldo scores}|\neg cavity) =$
$P(\text{Cristiano Ronaldo scores})$

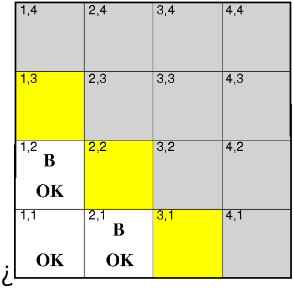
And use probabilistic inference

Basic insight: observations are conditionally independent of other hidden squares given neighbouring hidden squares



Define $\mathit{Unexplored} = \mathit{Fringe} \cup \mathit{Other}$
$\mathbf{P}(b|P_{1,3}, \mathit{Explored}, \mathit{Unexplored}) = \mathbf{P}(b|P_{1,3}, \mathit{Explored}, \mathit{Fringe})$
Manipulate query into a form where we can use this!

And get to the goal, most likely

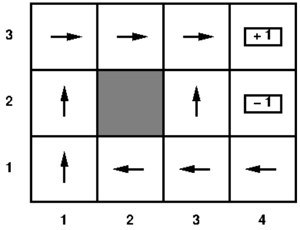


Rewards:

- -1000 for dying
- 0 any other square

What's the expected utility of going to $[3,1], [2,2], [1,3]$?

Stochastic sequential environments

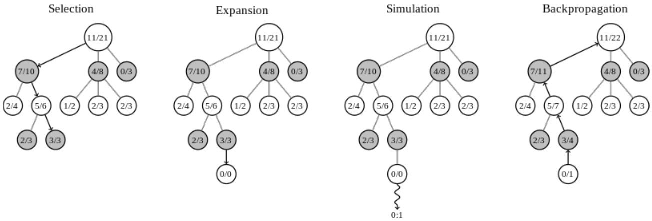


The optimal policy

$$\pi^*(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s'} P(s' \mid s, a) v(s')$$

Maximise the expected utility of the subsequent state

Learning, by trial and error



Basics behind the best of AI

Rational Agents

An **agent**'s behaviour is rational (or intelligent if you like) if it is in their best interest given their information

Rational Agents

An **agent**'s behaviour is rational (or intelligent if you like)
if it is in their best interest given their information
... and learning tells us how much information to look for and how.

Special thanks to...

Special thanks to...

Special thanks to...

Special thanks to...