Agent-Based Systems

Paolo Turrini

www.dcs.warwick.ac.uk/~pturrini p.turrini@warwick.ac.uk

Today

We have seen MDPs and how to calculate the optimal policy (VIA).

However:

- Maybe the state space is too big to do it
- Even if we do know the states we might not know how they are related.

Today we are going to see how to handle these cases, using Reinforcement Learning.

Incomplete knowledge

What if we don't know what game we are playing?

Play anyway and see what happens! and play as much as possible!

We can't possibly calculate everything

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^{21}[illegible]	[illegible]	[illegible]

Figure: The complexity of Go

Neural networks + tree search

Understanding the value of game positions using:

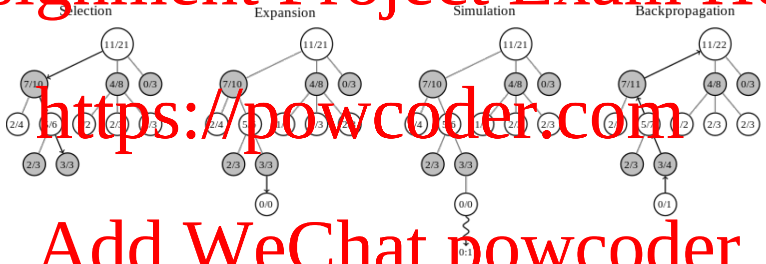
Neural Networks using pattern recognition from a database of previously played games.

Tree Search self-playing (a lot!) and estimating the value of moves;

David Silver et al.
Mastering the game of Go
with deep neural networks and tree search
Nature, 2016.

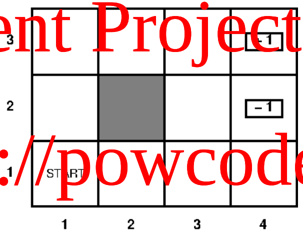
David Silver et al.
Mastering the game of Go without human knowledge
Nature, 2017.

I'm going to only focus on how to infer value
without using pre-processed information



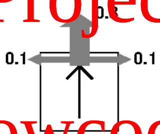
To evaluate intermediate game positions we play a huge number of games from then on.

The world



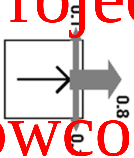
- Begin at the start state
- The game ends when we reach either goal state $+1$ or -1
- Rewards: $+1$ and -1 for terminal states respectively, -0.04 for all others

Full observability



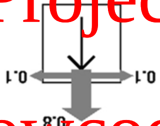
Stochastic actions (possibly different at each state!) four directions

Full observability

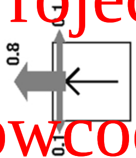


Stochastic actions (possibly different at each state!) four directions

Full observability

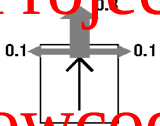


Stochastic actions (possibly different at each state), four directions



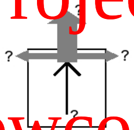
Stochastic actions (possibly different at each state), four directions

Full observability

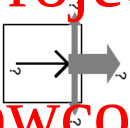


Stochastic actions (possibly different at each state), four directions

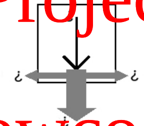
Partial observability



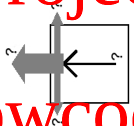
Stochastic actions (possibly different at each state): four directions



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

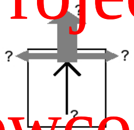


Stochastic actions (possibly different at each state), four directions



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

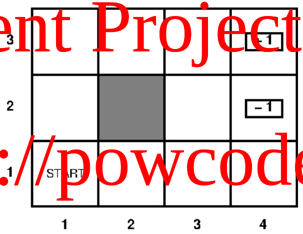
Passive reinforcement learning:

- 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:

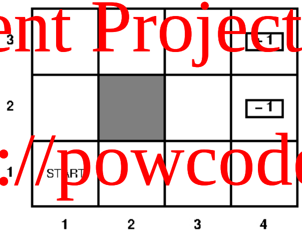
- I don't even have a policy

Passive Reinforcement Learning



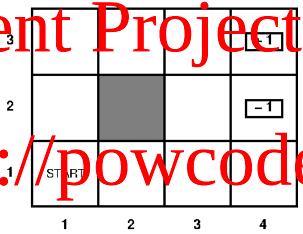
- I don't know the values nor the rewards
- I don't know the probabilities
- I'm gonna play anyway

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



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

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



Suppose I get these trials:

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

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

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

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

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

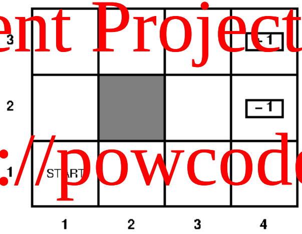
$$v^\pi(s) = v^\pi(s) + \alpha(r(s) + \gamma v^\pi(s') - v^\pi(s))$$

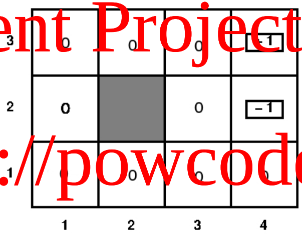
where $\alpha \in [0, 1]$ is a **confidence** parameter: how much we value the new information.

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

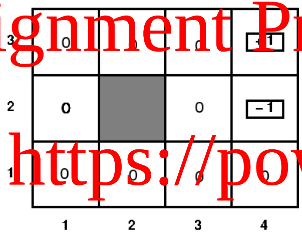


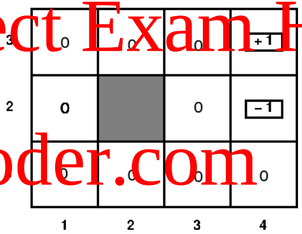


Initialise the values, for

- $\gamma = 1$
- 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

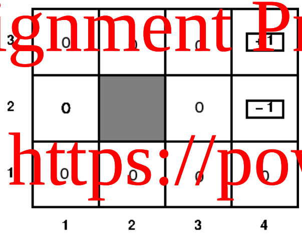


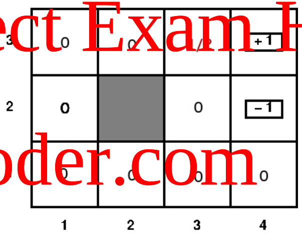


Suppose we can walk *(Up, Up, Right, Right, Right)*

Apply the update to states, as you walk along:

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

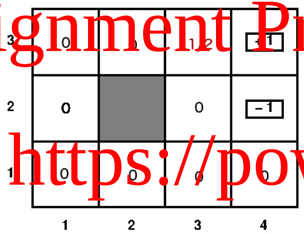


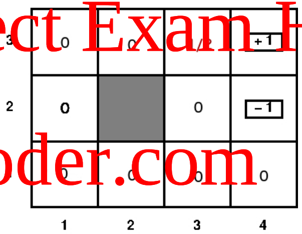


Suppose we can walk *(Up, Up, Right, Right, Right)*

Apply the update to states, as you walk along:

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

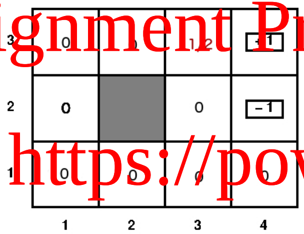


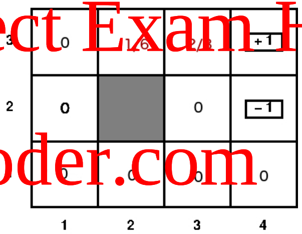


I keep walking the same way...

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

TDL in action





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

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

- 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.

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

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

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

is the value of performing action a in state s

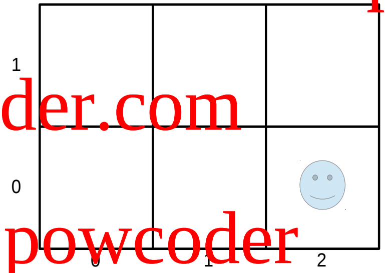
Q-learning

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

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

- 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

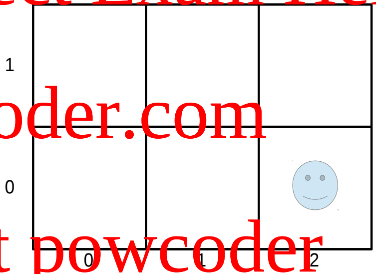


RL components in this problem

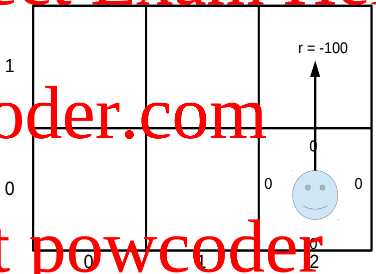
- **State**: The agent's current location
- **Action**: *LEFT*, *RIGHT*, *UP*, *DOWN*
- **Environment Dynamics**:
  - Collision results in no movement
  - otherwise, move one square in the intended direction
- **Rewards**:
  - normal move: -1
  - hit a wall: -10
  - die: -100
  - exit: +100
- **Our Goal**: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



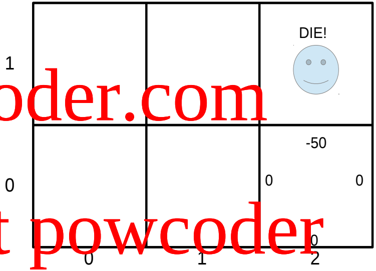
- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0
- Choose *UP*, and receive -100



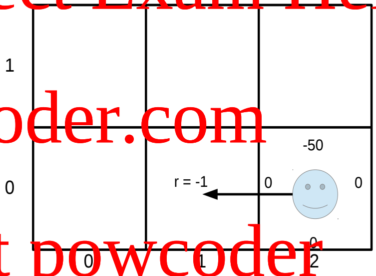
Applying Q-Learning to The Maze Problem

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

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



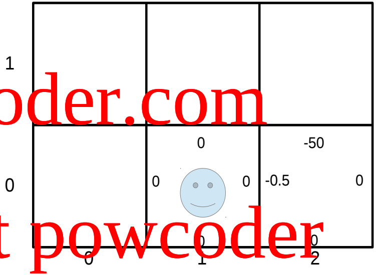
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



Applying Q-Learning to The Maze Problem

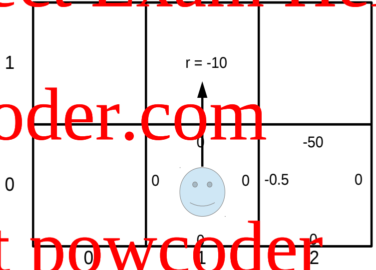
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

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

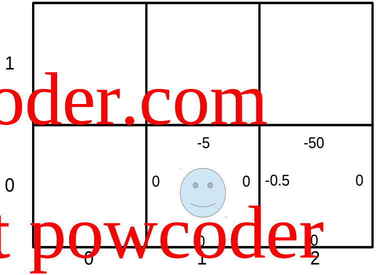


Applying Q-Learning to The Maze Problem

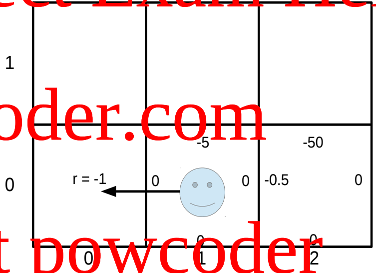
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10



- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10
- update Q-value:
  $Q([0,1], UP)$
  $= (1 - 0.5) \times 0 +$
  $0.5 \times (-10 + 0.9 \times 0)$
  $= -5$

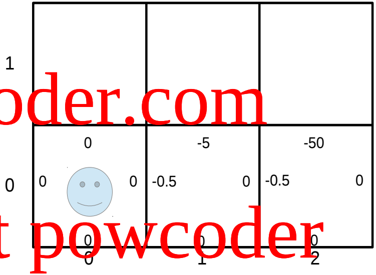


- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



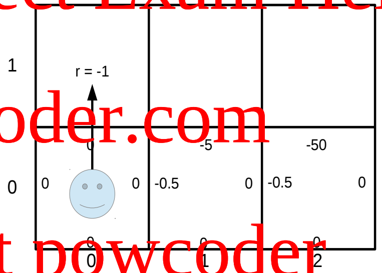
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

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



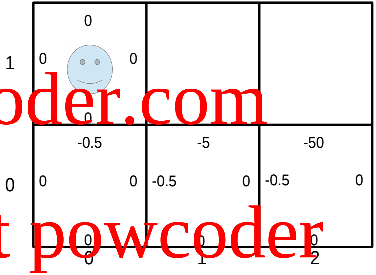
Applying Q-Learning to The Maze Problem

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

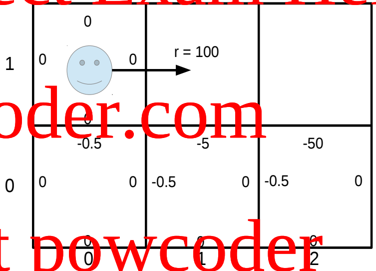


- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -1
- update Q-value:

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

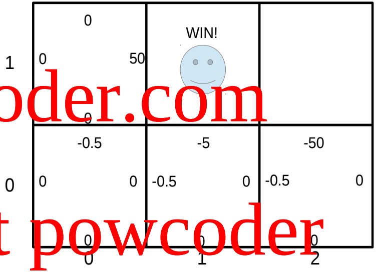


- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *RIGHT*, and receive 100



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

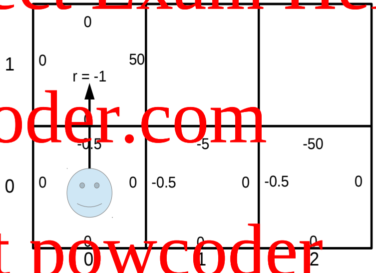
$$Q([0,1], RIGHT) = (1-0.5)\times 0 + 0.5\times(100 + 0.9\times 0) = 50$$



Applying Q-Learning to The Maze Problem

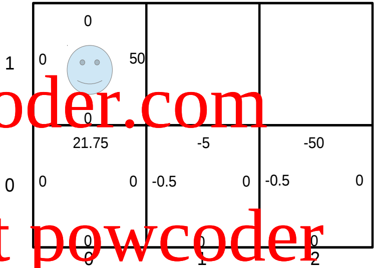
- $\alpha = 0.5$, $\gamma = 0.9$
- The next time agent visits [0,0] and performs *UP*:

$$
\begin{aligned}
&Q([0,0], UP) \\
&= (1 - 0.5) \times (-0.5) + \\
&\quad 0.5 \times (-1 + 0.9 \times 50) \\
&= 21.75
\end{aligned}
$$



- $\alpha = 0.5$, $\gamma = 0.9$
- 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)

What we have seen so far

- Decision making in sequential environments typical of AI practice
- Optimisation techniques (VIA)
- ... under incomplete information (Active/Passive Learning by Belmann updates)

What next? Population dynamics and multi-agent reinforcement learning.