Gerard and Tony's PhD Project:
Characterisation of nuisance calls in-network and improved detection using machine learning techniques
https://www.findaphd.com/search/projectdetails.aspx?PJID=96714

Reinforcement Learning

Dr. Jason Lines
j.lines@uea.ac.uk

University of East Anglia

Overview

Introduction

- Agent/Environment Model
- Difference between ML and RL
- N-Armed Bandit and Mazes

Formulating the Problem

- Strategies and Policies
- Discounted rewards
- Value function and Q function

Solving Reinforcement Learning Problems

- Planning vs Learning
- Q-Learning

Further Issues

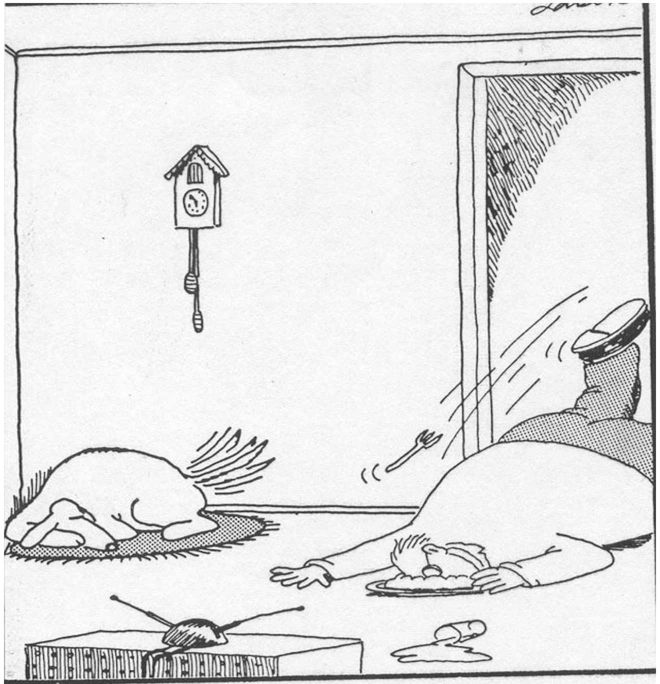
- Generalisation
- Limited Perception

Reinforcement Learning

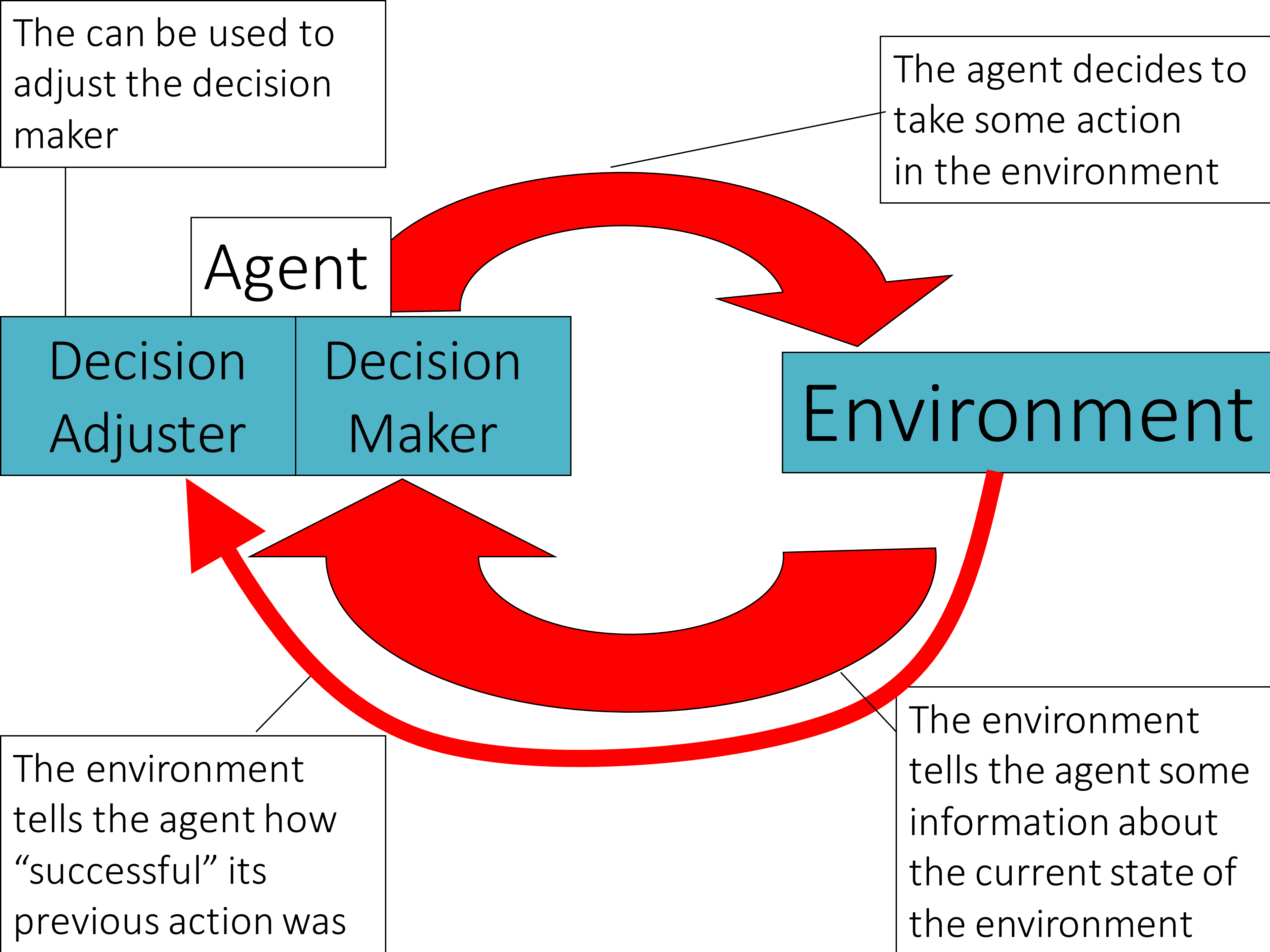
The reinforcement learning (RL) problem is the problem faced by an agent that learns behaviour through trial-and-error interactions with its environment in order to maximize its overall reward

Reinforcement learning is learning from experience to associate states and actions with desirable outcomes

In 1903 a Russian scientist called Pavlov conducted the classic stimulus/response experiment with dogs.



Stimulus-response behavior in dogs



The can be used to adjust the decision maker

The agent decides to take some action in the environment

Agent

Decision Adjuster

Decision Maker

Environment

The environment tells the agent some information about the current state of the environment

The environment tells the agent how “successful” its previous action was

Agent/Environment

An Environment can be in one of a set of states:

$$S = \{s_1, s_2, s_3, \dots\}$$

In any state, an agent can take an action:

$$A = \{a_1, a_2, a_3, \dots\}$$

For any state/action pair, the agent receives a reward signal from the environment, a numerical signal that can be used to quantify how well the agent is meeting its objective

$$r : S \times A \rightarrow \Re$$

The Agent Objective

The agent perceives a sequence of states and performs a sequence of actions. $(s_t, a_t) \in S \times A$

$$< (s_1, a_1), (s_2, a_2), \ldots, (s_t, a_t), (s_{t+1}, a_{t+1}), \ldots >$$

The agent chooses an action with a decision making strategy or policy

$$\Pi(s) : S \to A \quad \text{or} \quad \Pi(s) : S \to p(a)$$

The agent objective is to choose a strategy that maximizes some function of the total reward it receives.

e.g., the agent may try and discover the policy that maximizes total reward

$$\max \sum_{i=1}^{t} r(s_i, a_i)$$

Why Reinforcement Learning is (fiendishly) Difficult

1. The learning is unsupervised

if the agent receives a reward, it has no idea of finding out whether a different action would have yielded greater reward

2. There is usually uncertainty in the reward

If the agent takes the same action in a perceived identical environment, it cannot be sure of getting the same reward. Also, the reward function itself can change over time

Why Reinforcement Learning is (fiendishly) Difficult

3. Learning is online and bad decisions have a negative consequence

There is no historical corpus of sampled data to learn from. Each case counts towards meeting the goal and the agent must balance the need to EXPLORE for new knowledge vs the desire to EXPLOIT

4. Rewards may be delayed

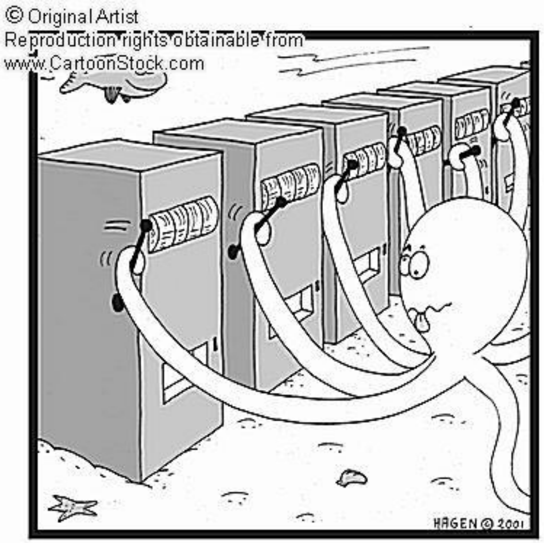
Several action/state pairs may not yield a reward but will lead to greater rewards later.

5. Actions influence future states

Choice of action now may alter the state we see next

Explore vs. Exploit Example

Example 1: n-armed bandit



© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

Compulsive gambling

The environment has a single state

You have a choice of *n* actions

Each action corresponds to choosing a fruit machine to play

Each fruit machine has a different reward function

You have to try and maximise your total expected reward over a given time interval (say 1000 goes)

n-armed bandit

Let $\bar{r}$ be the mean of the reward distribution for machine r. If we knew these means we could just choose the machine with the highest average payoff

If this were a machine learning problem, we would have a history of past pulls and could simply just estimate the mean payoff (imagine we were watching but not paying)

$\mu_1, \mu_2, \ldots, \mu_n$ with the sample means $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$

Once we have the means, our strategy then could be to just always use the fruit machine with the highest average reward

This would give us the maximum expected reward from that point

n-armed bandit

However, with reinforcement learning we have to optimise the reward while collecting the data (we have to pay for each go)

A Strategy: try each machine for a bit and then after a while only play the one that seems to pay out the most.

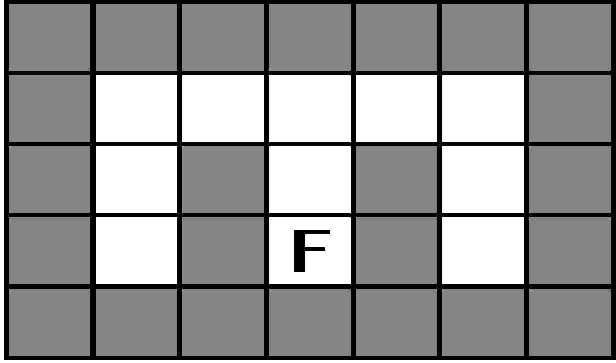
Explore: Select a machine randomly, then record the reward

Exploit: Choose the machine with the highest mean reward up until now

Balancing exploration and exploitation is a key feature of reinforcement learning

There is always a trade-off between discovering new things and doing what you already know is good

Example 2:
Maze Problems



The environment has many states

The agent is placed in a random cell and has to find its way to the food. It only has the choice of four actions (N,E,S,W)

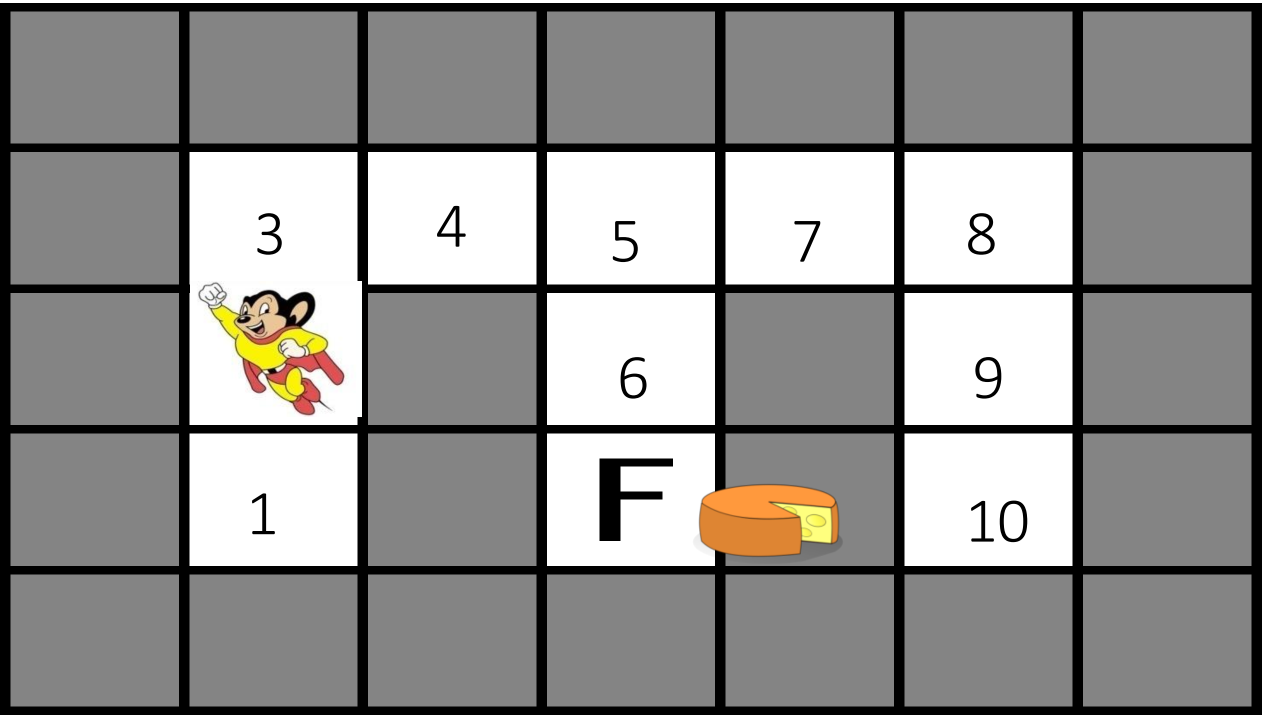
It only gets a reward when it gets to the cell marked Food.

The task is to learn the shortest route to food from any square

Note the agent may not be able to "see" exactly where it is. It may only perceive the surrounding squares. This means some squares may appear the same. These are aliasing squares

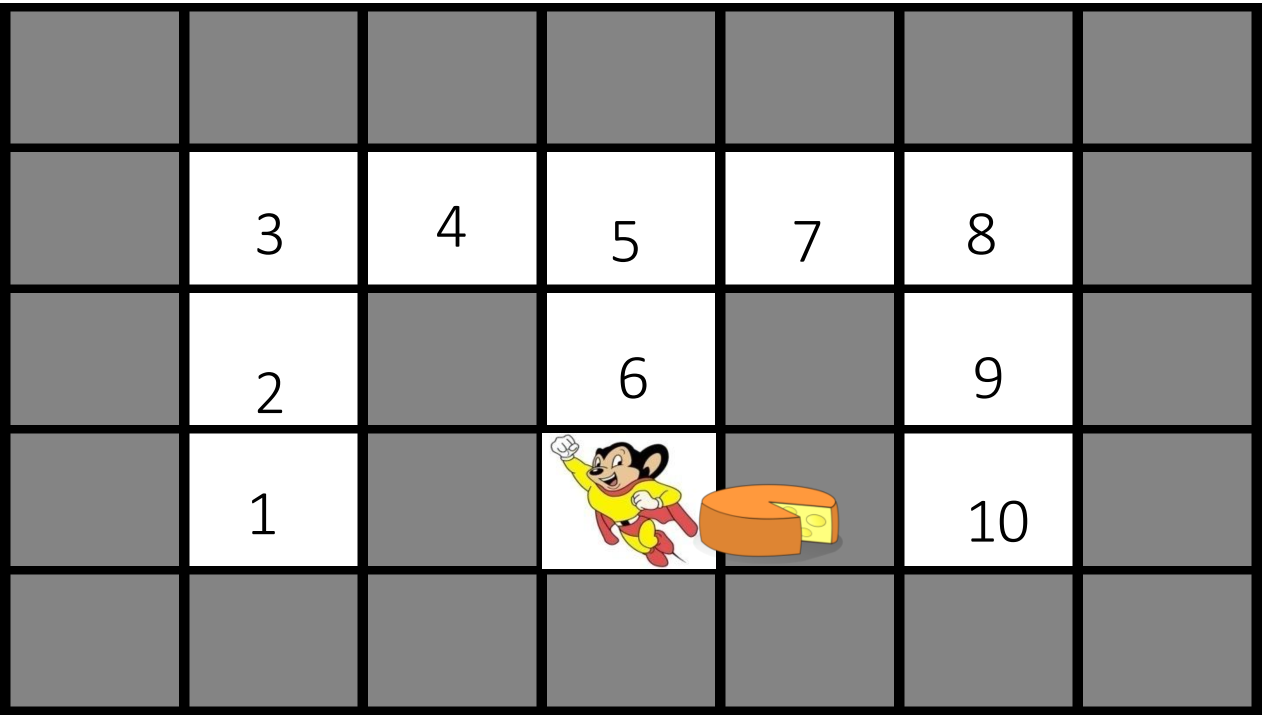
Maze woods101

Example 2:Maze Problems



Step	State	Action	Reward

Example 2:Maze Problems



Total Reward=100

Number of steps=10

Average Reward=10

Optimal Steps=5

Step	State	Action	Reward
1	2	S	0
2	1	W	0
3	1	N	0
4	2	N	0
5	3	E	0
6	4	E	0
7	5	E	0
8	7	W	0
9	5	S	0
10	6	S	100

Formulating the Problem

Strategies and Policies

A strategy is a mechanism for specifying an action in any environmental state

$$\pi(s) = a$$

π_1 Always go down

π^* Optimal Strategy

State	Action*	State	Action*
1	Up	6	Down
2	Up	7	Left
3	Right	8	Left
4	Right	9	Up
5	Down	10	Up

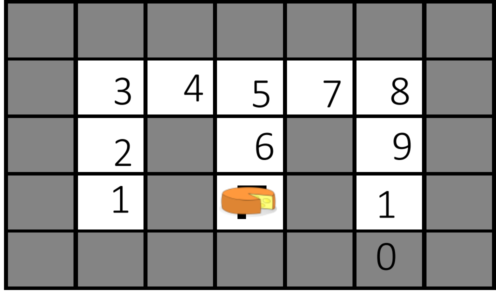
$$\pi(s) = p(a)$$

π_2 Probabilistic Strategy

State	Up	Down	Left	Right
1,2,9, 10	1	0	0	0
4,7	0	0	0.5	0.5
3,5,6, 8,10	0.25	0.25	0.25	0.25

The problem facing the agent is to find the optimal strategy

Average Rewards Strategy Measure



Assume we use average reward on a run to measure the value of the strategy from start to Terminal State T

Optimal Strategy	
State	Action*
1	Up
2	Up
3	Right
4	Right
5	Down
6	Down
7	Left
8	Left
9	Up
10	Up

$$< (s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T) >$$

$$V^*(s_1) = \frac{\sum_{i=1}^{T} r(s_i, a_i)}{T}$$

State	Action*	R*
1	Up	**16.67**
2	Up	**20**
3	Right	**25**
4	Right	**33.33**
5	Down	**50**
6	Down	**100**
7	Left	**33.33**
8	Left	**25**
9	Up	**20**
10	Up	**16.67**

Average Reward For Optimal Strategy

Discounted Rewards vs. Average Rewards

It is more common to use discounted rewards to measure the value of a strategy in reinforcement learning implementations

The principle is that rewards received more recently are worth more than rewards received some time in the future.

Discounted Reward

$$\boxed{V_d^{\pi}(s) = \sum_{i=1}^{T} \gamma^{i-1} r(s_i, \pi(s_i)) = \sum_{i=1}^{T} \gamma^{i-1} r(s_i, a_i)}$$

Average Reward

$$V_a^{\pi}(s) = \frac{\sum_{i=1}^{T} r(s_i, a_i)}{T}$$

where Gamma is the discount rate $0 \leq \gamma \leq 1$

Higher values of Gamma give greater weight to rewards received in the future.

Discounted Rewards Strategy Measure

Optimal Strategy	
State	Action*
6	Down
5	Down

Let $\gamma = 0.5$

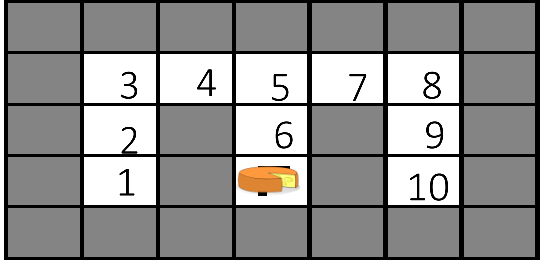
$< (6, D) >$ $\qquad r(6, D) = 100$

$$\boxed{V_d^*(6) = \gamma^0 r(6, D) = 100}$$

$< (5, D), (6, D) >$

$$\boxed{\begin{array}{l} V_d^*(5) = \gamma^0 r(5, D) + \gamma^1 r(6, D) \\ V_d^*(5) = 50 \end{array}}$$

$$V_d(s) = \sum_{i=1}^{T} \gamma^{i-1} r(s_i, a_i)$$



State	Action*	v*
1	Up	
2	Up	
3	Right	
4	Right	
5	Down	50
6	Down	100
7	Left	
8	Left	
9	Up	
10	Up	

Discounted Rewards for Optimal Strategy

Discounted Rewards for Optimal Strategy

Optimal Strategy

State	Action*
6	Down
5	Down
4	Right
7	Left

Let $\gamma = 0.5$

$$< (4, R), (5, D), (6, D) >$$

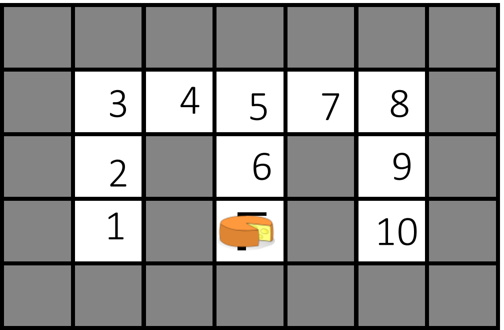
$$< (7, L), (5, D), (6, D) >$$

$$V_d^*(4) = \gamma^0 r(4, R) + \gamma^1 r(5, D) + \gamma^2 r(6, D)$$

$$V_d^*(4) = 0 + 0 + .25 * 100 = 25$$

State	Action*	v*
1	Up	
2	Up	
3	Right	
4	Right	25
5	Down	50
6	Down	100
7	Left	25
8	Left	
9	Up	
10	Up	

Discounted Rewards for Optimal Strategy



$$V_d(s_1) = \sum_{i=1}^{T} \gamma^{i-1} r(s_i, a_i)$$

Discounted Rewards for Optimal Strategy

Optimal Strategy

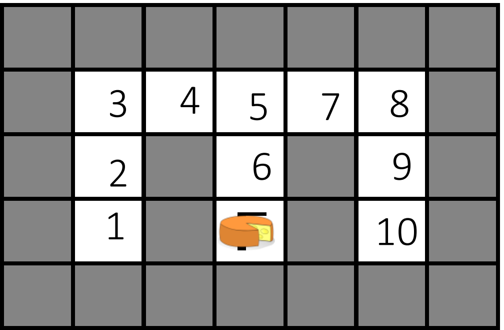
State	Action*
6	Down
5	Down
4	Right
7	Left
3	Right
8	Left
2	Up
9	Up

Let $\gamma = 0.5$

$$R_d^*(3,R) = r(3,R) + \gamma \cdot R_d^*(4,R)$$
$$R_d^*(3,R) = 0.5 \cdot 25$$

$$R_d^*(2,U) = r(2,U) + \gamma \cdot R_d^*(3,R)$$

State	Action*	R*
1	Up	3.125
2	Up	6.25
3	Right	12.5
4	Right	25
5	Down	50
6	Down	100
7	Left	25
8	Left	12.5
9	Up	6.25
10	Up	3.125



We can define the value recursively:

$$V^{\pi^*}(s_i) = r(s_i, \pi^*(s_i)) + \gamma \cdot V^{\pi^*}(s_{i+1})$$

Value Function

More generally, RL tasks may have stochastic rewards and may not terminate. Hence we use expected values and leave the sum open

$$\boxed{V^{\pi}(s) = E\left\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots \mid s_t = s, \pi\right\}}$$

The optimal value function is $V^*(s) = \max_{\pi} V^{\pi}(s)$

But we cannot search for the optimal value function by enumerating the policy space

Instead, we embed the optimal strategy in a Q-Function, that gives the value of taking an action, then following the optimal strategy

Q Function

For any state action pair *s,a* the state-value Q function is

$$Q^{\pi}(s,a) = E\left\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+2} + \ldots \mid s_t = s, a_t = a, \pi\right\}$$

The Q function for any policy π is a table that gives the expected reward in state *t* for taking action *a* then taking action dictated by $\pi(S_{t+1})$

State	Strategy
1	Up
2	Up
3	Right

State	Value
1	3.125
2	6.25
3	12.5

The Q Table

State	Left	Right	Up	Down
1	1.062	1.062	**3.125**	1.062
2	3.125	3.125	**6.25**	1.062
3	6.25	**12.5**	6.25	3.125

Q Table

The point of the Q Function is that we can work out both the strategy and the value function directly from it

$$\pi(s) = \arg\max_{s} Q^{\pi}(s,a) \qquad V^{\pi}(s) = \max_{a} Q^{\pi}(s,a)$$

We can also search the space of all Q tables for the optimal

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Crucially, we can use the current best Q table to guide the search. Hence the reinforcement learning strategy is to interact with the environment based on our current Q table, and to adjust it based on experience

Constructing the Optimal Q-Function

The optimal Q function gives us a way of finding the optimal strategy

Hence if we can evaluate the Q function we can solve the problem

We can define the Optimal Q function recursively. Suppose we are in state *s*. If we take action *a* we get reward $r(s,a)$

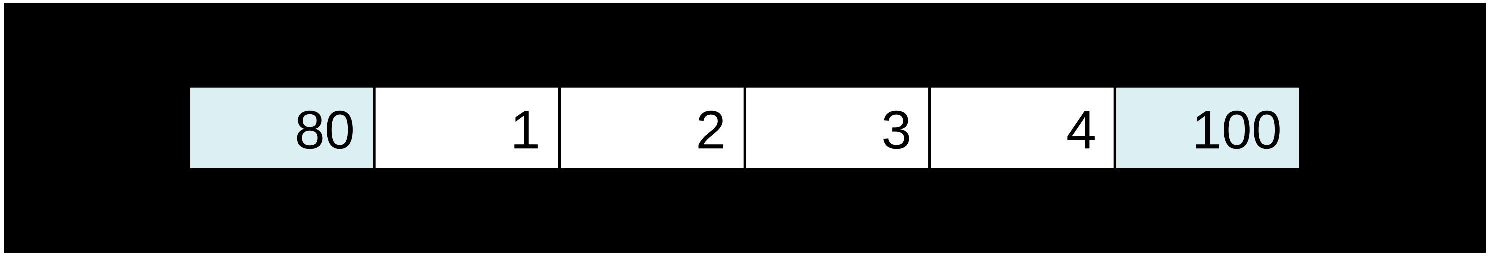
If we take action in environment *s* then we move to state s'

We can then recursively define the Q function

A Dynamic programming formulation can be used to solve this if all the rewards and transitions are known

$$Q^*(s,a) = r(s,a) + \gamma \max_{b \in A} Q^*(s',b)$$

Example 1: Enumerating the Q function



80	1	2	3	4	100

States: 4 Action Set A={E,W}

Discount rate =0.9

$$Q^*(4,E)=100 \qquad Q^*(1,W)=80$$

$$Q^*(3,E)=0+0.9*\max\{Q^*(4,E),Q^*(4,W)\}$$

$$=\max\{90,?\}$$

Assume 90 for now

State	E	W
1	72.9 (?)	80
2	81(?)	
3	90(?)	
4	100	

$$Q^*(2,E)=0+\gamma\max\{(3,E),(3,W)\}=0+0.9*\max\{90,?\}=81(?)$$

$$Q^*(1,E)=0+\gamma\max\{(2,E),(2,W)\}=0+0.9*\max\{81,?\}=72.9(?)$$

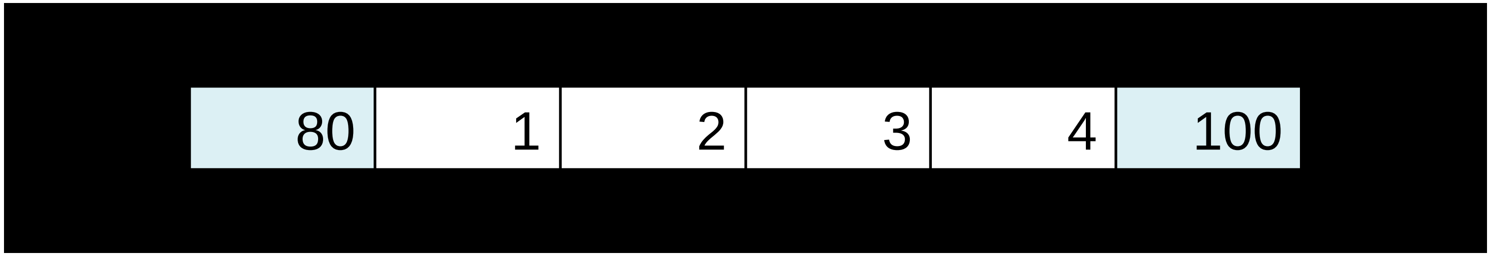
Example 1: Enumerating the Q function

$$Q^*(2,W) = 0 + 0.9 * \max\{(1,E),(1,W)\}$$
$$= 0 + 0.9 * \max\{72.9(?),80\} = 72$$

$$Q^*(1,E) = 0 + 0.9 * \max\{(2,E),(2,W)\} =$$
$$0 + 0.9 * \max\{81,72\} = 72.9$$

State	E	W
1	72.9 (?)	80
2	81(?)	72
3	90(?)	72.9
4	100	81

$$Q^*(3,W) = 0 + \gamma \max\{(2,E),(2,W)\} = 0 + 0.9 * \max\{81,72\} = 72.9$$
$$Q^*(2,E) = 0 + \gamma \max\{(3,E),(3,W)\} = 0 + 0.9 * \max\{90,72.9\} = 81$$
$$Q^*(4,W) = 0 + \gamma \max\{(3,E),(3,W)\} = 0 + 0.9 * \max\{90,72.9\} = 81$$



80	1	2	3	4	100

Example 1: Optimal Policy

$$Q^*(s,a) = r(s,a) + \gamma \max_{b \in A} Q^*(s',b)$$

State	Move
1	W
2	E
3	E
4	E

Q Function

State	E	W
1	72.9	80
2	81	72
3	90	72.9
4	100	81

80	1	2	3	4	100

Markov Decision Processes

Mazes are a specific example of a type of problem known as a Markov decision process

S: Set of States *A*: Set of Actions R: A Reward Function

T: A Transition matrix of probabilities for each action

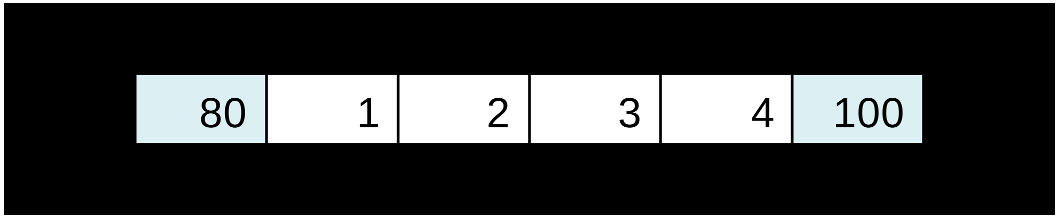
S={1,2,3,4}

A={E,W}

R(1,w)=80

R(4,e)=100

$$T^E = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T^W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



80	1	2	3	4	100

Markov Decision Processes

The central MDP assumptions are:

1. The current state is purely dependent on the previous state
2. *A,S,R and T* are known

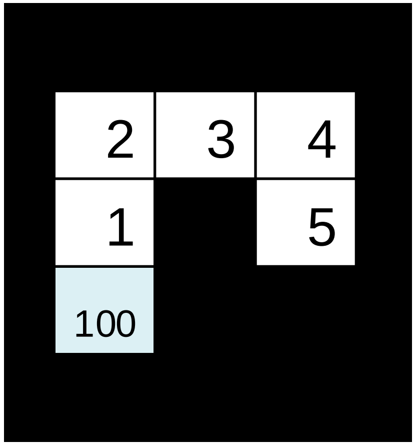
If this is true then there is guaranteed to be an optimal policy

Solving the MDP involves finding the Q function and hence the optimal policy

Finding the optimal policy is computationally expensive

Example 2: Maze as an MDP

QUESTION: Describe the following maze as a Markov Decision Process



S={1,2,3,4,5}

R(1,S)=100

A={N,E,S,W}

$$T^N = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

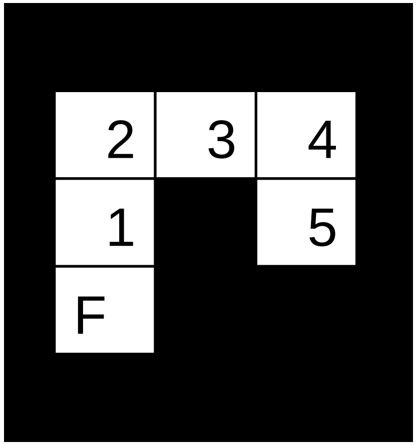
$$T^E = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^W = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Example 2: Enumerating the Q-Function (again)

QUESTION: Derive by hand the Q-Function for the following maze and hence give the optimal strategy. Assume a discount rate of 0.9 and finding food gives a reward of 100



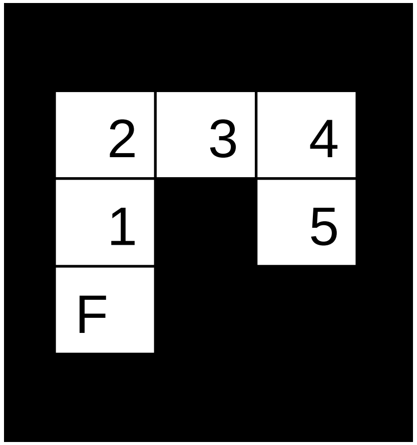
Fill in the optimal route first

State	N	E	S	W
1			100	
2			90	
3				81
4				72.9
5	65.61			

Example 2: Enumerating the Q-Function (again)

Fill in the rebounds next

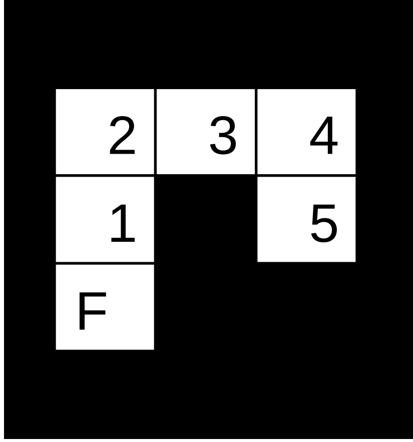
State	N	E	S	W
1		90	100	90
2	81		90	81
3	72.9		72.9	81
4	65.61	65.61		72.9
5	65.61	59.05	59.05	59.05



$$0 + \gamma \cdot \max_{b \in \{N,E,S,W\}} (Q(1,b))$$

$$Q^*(s,a) = r(s,a) + \gamma \max_{b \in A} Q^*(s',b)$$

Example 2: Finding the Q-Function



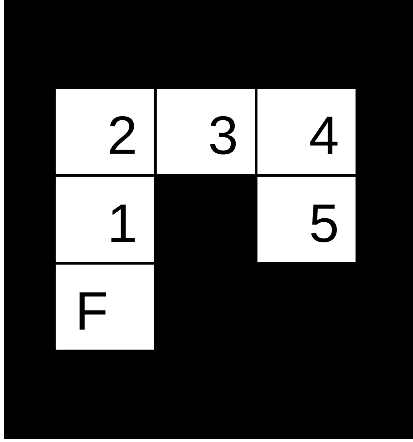
Fill in the rest

State	N	E	S	W
1	**81**	90	100	90
2	81	**72.9**	90	81
3	72.9	**65.61**	72.9	81
4	65.61	65.61	**59.05**	72.9
5	65.61	59.05	59.05	59.05

$$\gamma \cdot \max_{b \in \{N,E,S,W\}} (Q(5,b))$$

$$\arg\max_{b \in \{N,E,S,W\}} (Q(1,b))$$

Example 2: Finding the Q-Function



Fill in the rest

State	N	E	S	W
1	**81**	90	100	90
2	81	**72.9**	90	81
3	72.9	**65.61**	72.9	81
4	65.61	65.61	**59.05**	72.9
5	65.61	59.05	59.05	59.05

$$\gamma \cdot \max_{b \in \{N,E,S,W\}} (Q(5,b))$$

$$\arg\max_{b \in \{N,E,S,W\}} (Q(1,b))$$

State	Move
1	S
2	S
3	W
4	W
5	N

Planning Techniques

Of course, its not usually possible to find the Q-function by hand

There is a whole field of study on solving MDPs

Policy Iteration
Value Iteration
Linear Programming
Dynamic Programming
Function Approximators

However... in many situations you may not know *T*,*S*,*R* or *A*

Planning Techniques

Planning

In situations where we have some control, we may be able to solve the MDP

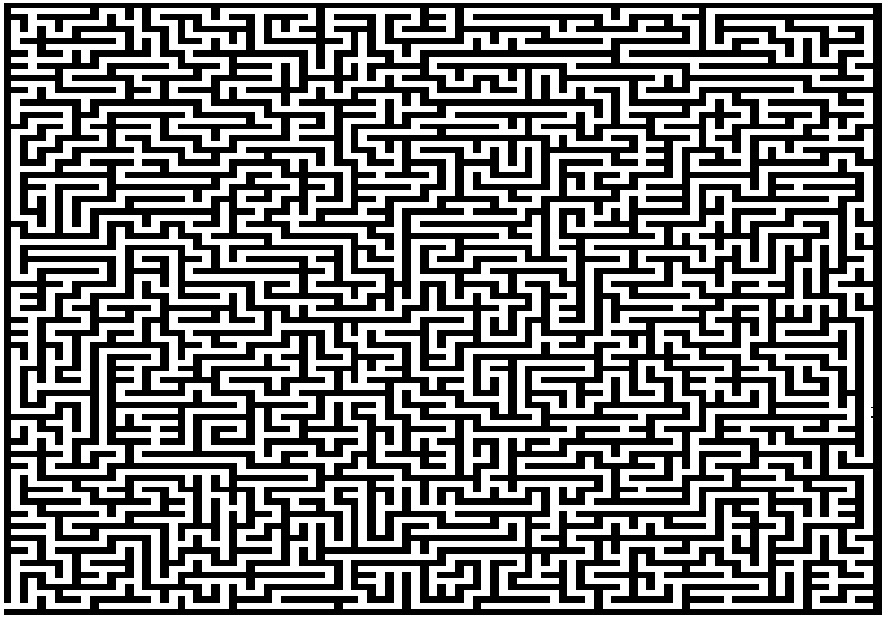
Maze solver
Robot path planning
Travel route planning
Elevator scheduling
Manufacturing processes
Network switching & routing

Learning

In complex situations, we may not know *T, R, S or A*

Rat in a maze
Agent in a market
Robot in the real world
Network packet sending algorithms

Planning a Solution to a Maze

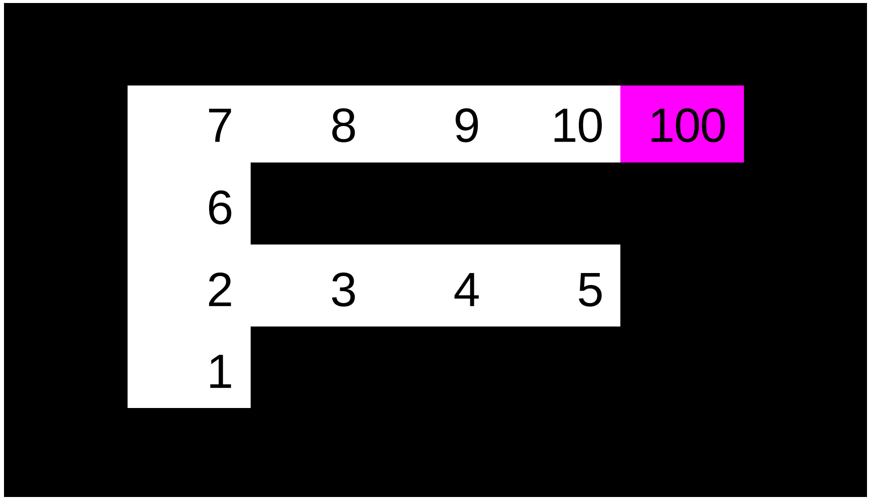


Given all the information, solve the maze

To solve, you need to model or know

1. the number of states
2. how the states are linked (transitions)
3. what the rewards are (i.e. where the exits are)

Learning a Maze



An agent is placed in a square. It has a detector that can only see the surrounding squares

1. It can only remember squares it has seen before and has no idea how many states there are
2. It can only infer linkage of states through experience
3. It has no prior knowledge as to the rewards

Watkins,C,1989.Learning from Delayed Rewards, PhD Thesis, University of Cambridge.

Q-Learning

Q-Learning is an iterative algorithm that attempts to learn the optimal Q-table through trial and error experience

It starts with a Q-table with all values set the same

It then adopts an action selection method to choose which action it should take for the state it is in.

It updates the Q-table based on any signal received through the environment by the Widrow Hoff delta update rule (**reinforcement learning**)

If a strategy is able to theoretically sample all cells in the Q-table infinitely often, it will converge to the optimal Q-Values. *If not, it will not converge*

Q-Learning Algorithm

```
Initialise $Q$ table
For every Turn
    Detect Initial State, $s$
    Until Turn terminated
            Choose an action $a$ based on current Q table
            Do action $a$ and obtain reward r
            Reinforce Q table entry $Q(s,a)$
            Detect next state $s'$
            $s=s'$;
```

Action Selection Algorithms

Greedy: Pure Exploit strategy that always choose action With the highest current Q-value

$$\boxed{a = \arg\max_{b \in A} Q(s, b)}$$

Will not converge to the optimum Q-table

Random: Pure Explore strategy that ignores the current Q-values

$$\boxed{a = random(A)}$$

Will converge (but very slowly)

Action Selection Algorithms

Roulette : Choose proportional to current Q-Values

$$\boxed{P(a) = \frac{Q(s,a)}{\sum_{b \in A} Q(s,b)}}$$

Will converge to the optimum Q-table

e-Greedy :

Either greedy with probability *(1-e)* or random with probability *e* (where *e* is usually small)

Will converge to the optimum Q-table

Action Selection Algorithms

Boltzman :

Start of with a greater probability of explore, then tend towards exploit as experience increases

$$p(a) = \frac{e^{\frac{Q(s,a)}{x}}}{\sum_{b \in A} e^{\frac{Q(s,b)}{x}}}$$

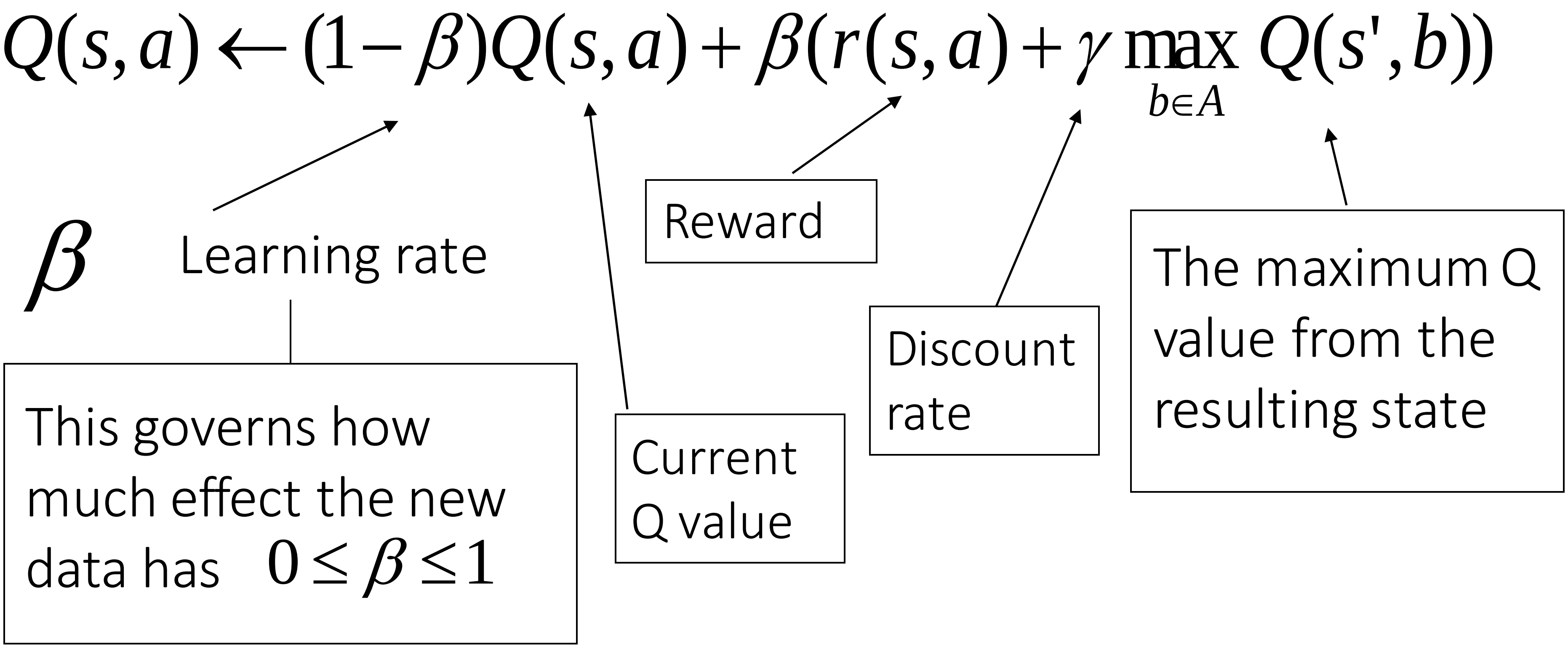
The bigger *x* is the closer this is to random selection

x then decays with time by some *cooling schedule*

May not converge to the optimum Q-table, depends on cooling schedule

Q-Learning Reinforcement

Q-Learning uses the Widrow-Hoff Update Rule



$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b\in A} Q(s',b))$$

β Learning rate

This governs how much effect the new data has $0 \le \beta \le 1$

Current Q value

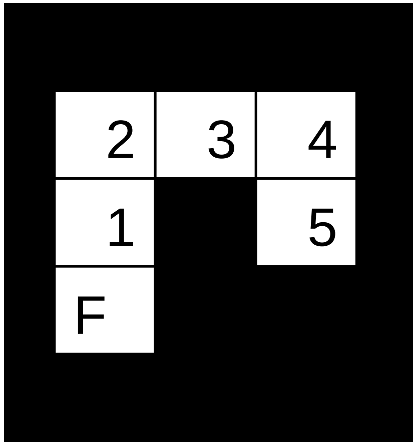
Reward

Discount rate

The maximum Q value from the resulting state

Q-Learning Example

QUESTION: An agent is placed on square 4 in the following maze. Given the current Q-Table below, perform 5 iterations of the Q-Learning algorithm using a greedy action selection algorithm and updating with the Widrow-Hoff update rule. Assume a learning rate of 0.2 and a discount rate of 0.9



State	N	E	S	W
1	10	10	95	10
2	46	43	65	55
3	40	50	60	62
4	30	50	70	68
5	49.5	44	50	41

$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b \in A} Q(s',b))$$

Iteration 1:

s=4

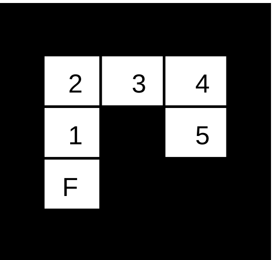
$$a = \arg\max_{b \in A} Q(s,b) = SOUTH$$

$$\Rightarrow s' = 5$$

$$Q(4,S) \leftarrow (1-0.2)Q(4,S) + 0.2(0 + 0.9 \max_{b \in A} Q(5,b))$$

$$Q(4,S) = 56 + 9 = 65$$

State	N	E	S	W
1	10	10	95	10
2	46	43	65	55
3	40	50	60	62
4	30	50	$\boxed{65}$	68
5	49.5	44	50	41



$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b \in A} Q(s',b))$$

Iteration 2:

s=5

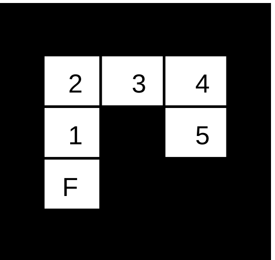
$$a = \arg\max_{b \in A} Q(s,b) = SOUTH$$

$$\Rightarrow s' = 5$$

$$Q(5,S) \leftarrow (1-0.2)Q(5,S) + 0.2(0 + 0.9 \max_{b \in A} Q(5,b))$$

$$Q(5,S) = 40 + 9 = 49$$

State	N	E	S	W
1	10	10	95	10
2	46	43	65	55
3	40	50	60	62
4	30	50	65	68
5	49.5	44	49	41



$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b \in A} Q(s',b))$$

Iteration 3:

s=5

$$a = \arg\max_{b \in A} Q(s,b) = North$$

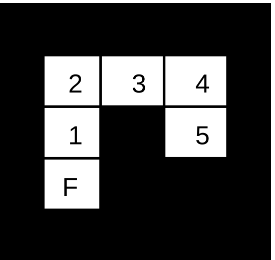
$$\Rightarrow s' = 4$$

$$Q(5,N) \leftarrow (1-0.2)Q(5,N) + 0.2(0 + 0.9 \max_{b \in A} Q(4,b))$$

$$Q(5,N) = 0.8 \cdot 49.5 + 0.2 \cdot (0.9 \cdot 68)$$

$$= 51.84$$

State	N	E	S	W
1	10	10	95	10
2	46	43	65	55
3	40	50	60	62
4	30	50	65	68
5	51.84	44	49	41



$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b\in A} Q(s',b))$$

Iteration 4:

s=4

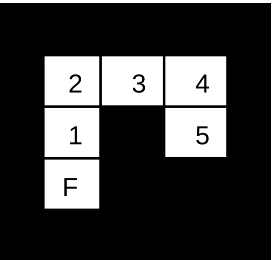
$$a = \arg\max_{b\in A} Q(s,b) = West$$

$$\Rightarrow s' = 3$$

Et cetera

The agent will now head home

State	N	E	S	W
1	10	10	95	10
2	46	43	65	55
3	40	50	60	62
4	30	50	65	68
5	51.84	44	49	41



Related Algorithms

Q-Learning is very similar to Temporal Difference learning (TD). TD attempts to learn the value function

It is even more similar to State-Action-Reward-State-Action (SARSA)

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r(s_t,a_t) + \phi(Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)))$$

Note: SARSA and TD assume a policy has been defined, whereas Q-Learning attempts to learn the whole Q-table

TD was used to create TD-Gammon, a program that plays backgammon as well as a human expert

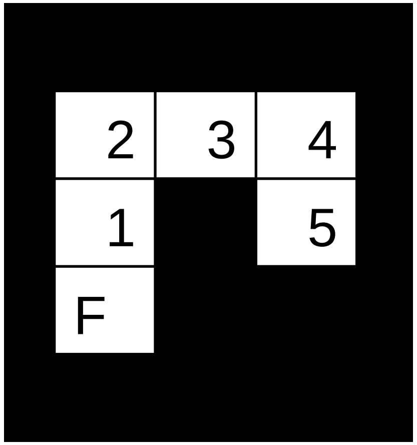
Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, March 1995 / Vol. 38, No. 3.

Generalisation

One of the problems with Q-learning is that interesting problems will have a huge number of states

The learning task then becomes not only to find the optimal policy, but also to generalise to group states with similar Q-values together

For example, for the maze states 3 and 4 have the same best actions but different Q-Values



State	N	E	S	W
1	81	90	100	90
2	81	65.61	90	81
3	65.61	59.05	65.61	72.9
4	59.05	59.05	53.14	65.61
5	59.05	53.14	53.14	53.14

Limited Perception

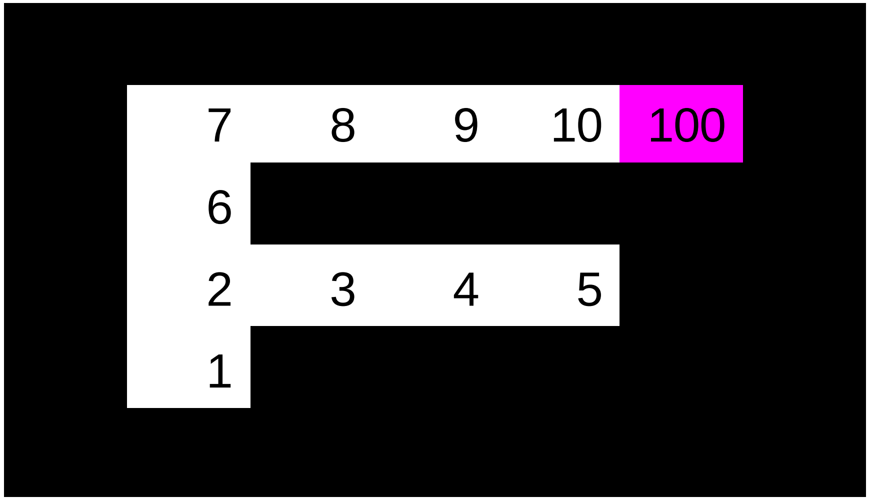
It is not realistic to assume a rat like maze solver can see the whole maze.

Similarly, it is not realistic to imagine an adaptive internet worm could map the whole internet

Instead, we assume they have a detector that perceives the environment

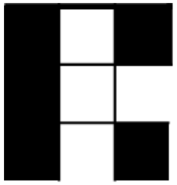
This introduces a new type of complexity, in that different states can now appear the same to the agent

Limited Perception

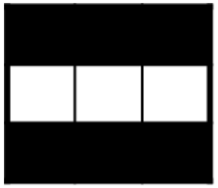


Suppose the agent can only perceive the squares around it. It needs to build the Q-table as it encounters new states

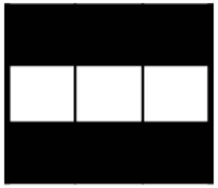
State 2



State 4

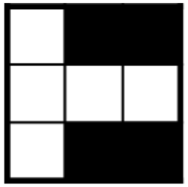


State 9



These states appear the same to the agent!

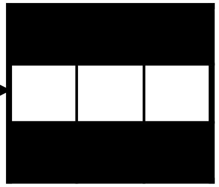
State 3



Limited Perception Agent

Start in State 4

1. Detect current environment



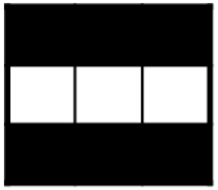
2. Find if seen before

Hash perception onto a list structure

3. If new, add a row to the Q-Table, initialised to zero

4. Proceed as usual

When the agent gets to state 9, it will perceive



The optimal strategy in state 9 is East and in state 4 it is west

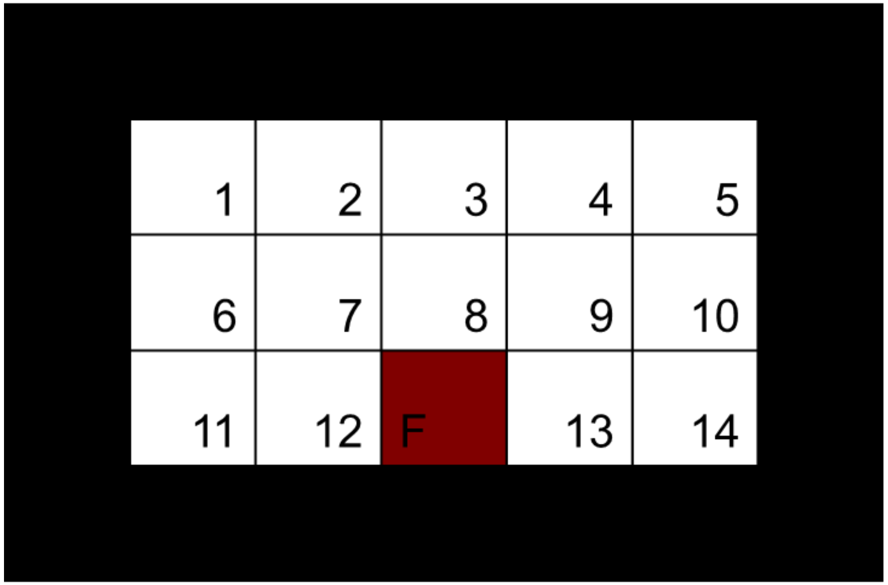
This is enough to cause Q-Learning major grief!

Limited Perception Agent

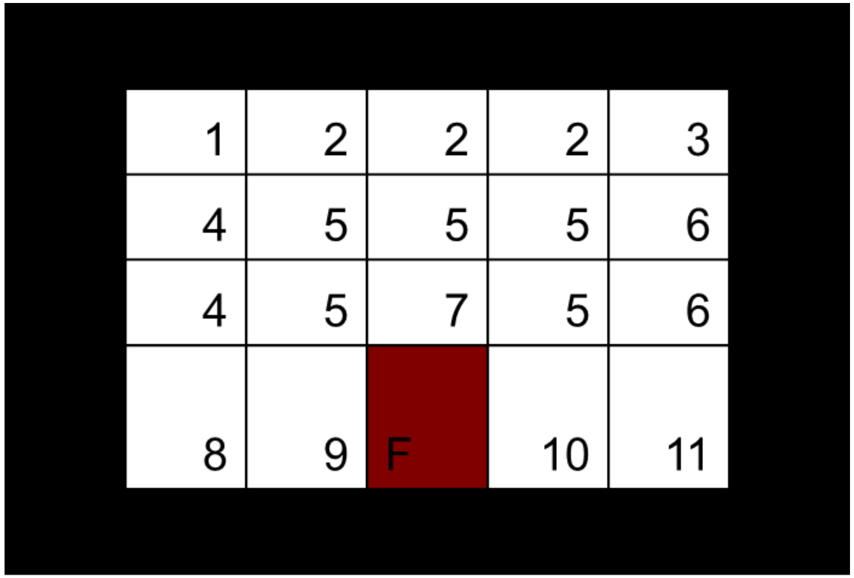
Assume the agent can only see N, S, E and W of its current position

How many distinct states will an agent with limited perception detect in the maze below?

Will it matter too much?



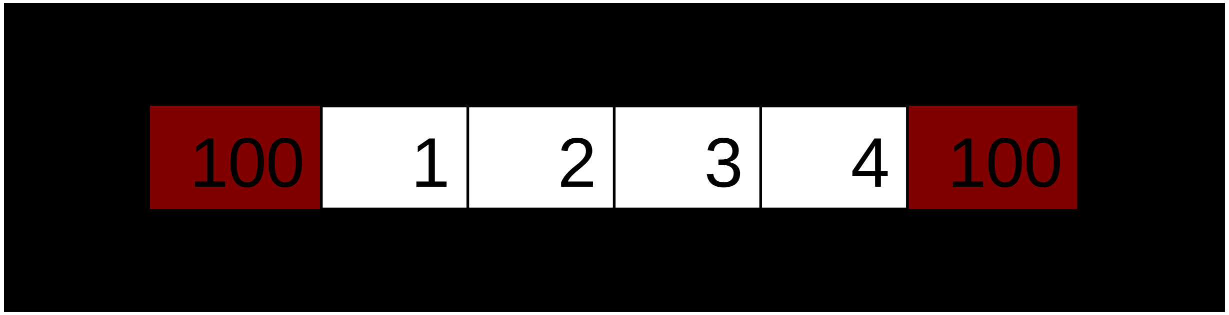
1	2	3	4	5
6	7	8	9	10
11	12	F	13	14



1	2	2	2	3
4	5	5	5	6
4	5	7	5	6
8	9	F	10	11

Limited Perception Agent

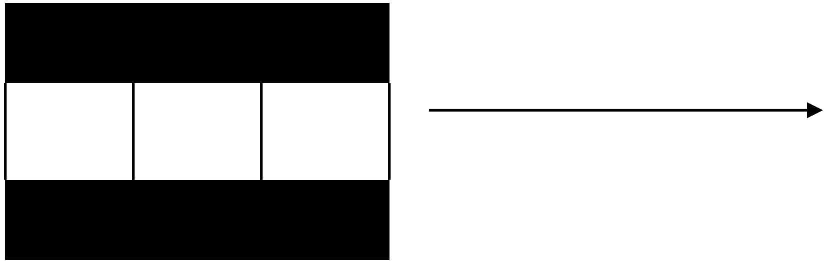
What about this one?



The aliasing of squares 2 and 3 will cause the learning algorithm great difficulty because the optimal strategy is different

Exercise: Assuming the agent can only move east or west,
perform by hand a run of a Q-Learning algorithm,
starting the first in 2 and assuming a greedy action selection mechanism and initial Q values of 50

Start in 2



Agent State 1

Agent State	W	E
2	50	50

$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r(s,a) + \gamma \max_{b \in A} Q(s',b))$$