Reinforcement Learning: An Introduction

Deep Learning Indaba
September 2017

Vukosi Marivate and Benjamin Rosman

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

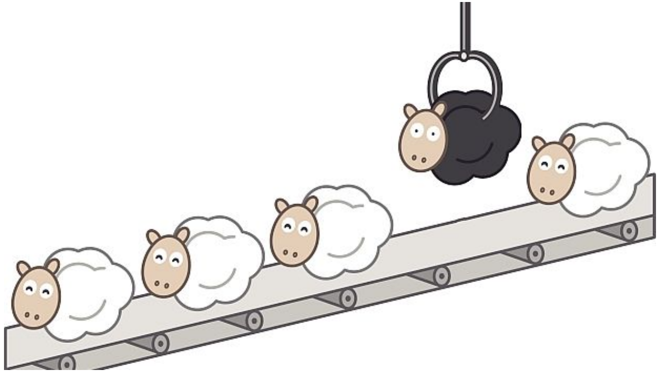
CSIR
our future through science

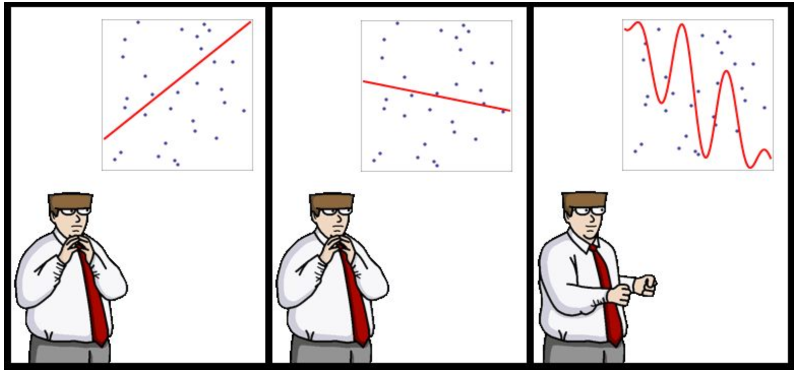
Contents

Contents

What is reinforcement learning?

We’ve seen how to solve many cool problems around supervised and unsupervised learning





But a major component of intelligence is decision making

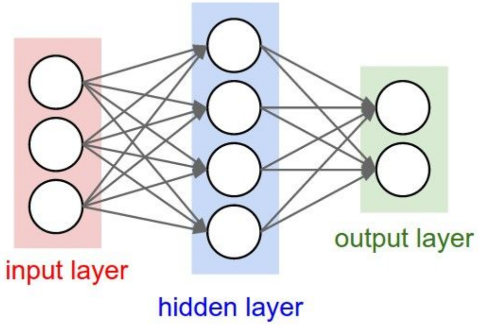
What is reinforcement learning?

Reinforcement learning is the branch of machine learning relating to learning in **sequential decision making settings**

Behaviour learning

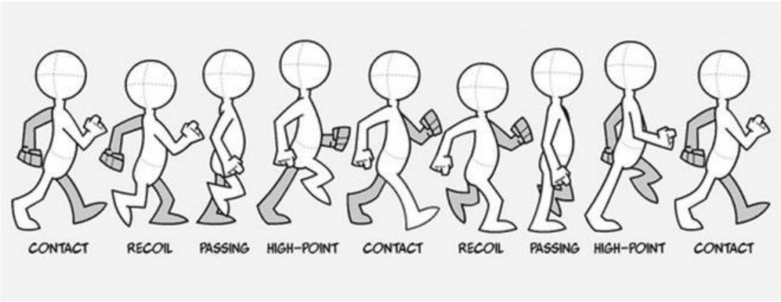
From supervised to reinforcement

Supervised learning, single decision point



Multiple decision points

- How do I know if I'm doing the right thing?
- How do my decisions now impact the future?
- Actions affect the environment!



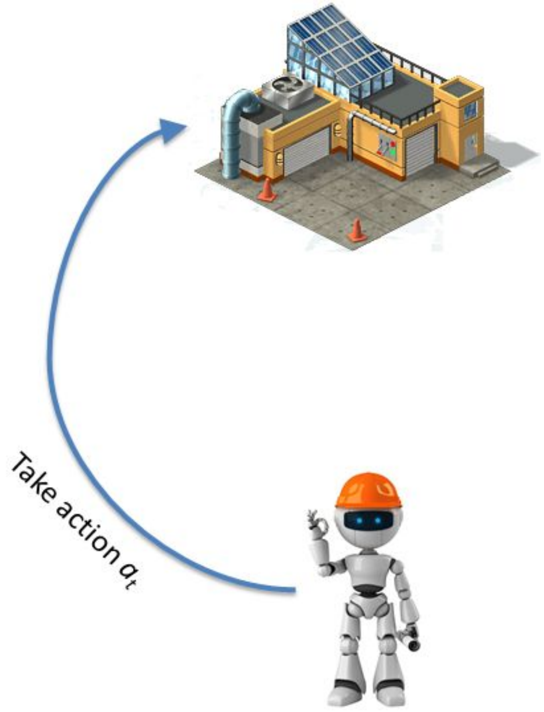
Interacting with an environment

Decision maker (agent) exists within an environment

Interacting with an environment

Decision maker (agent) exists within an environment

Agent takes **actions** based on the environment **state**



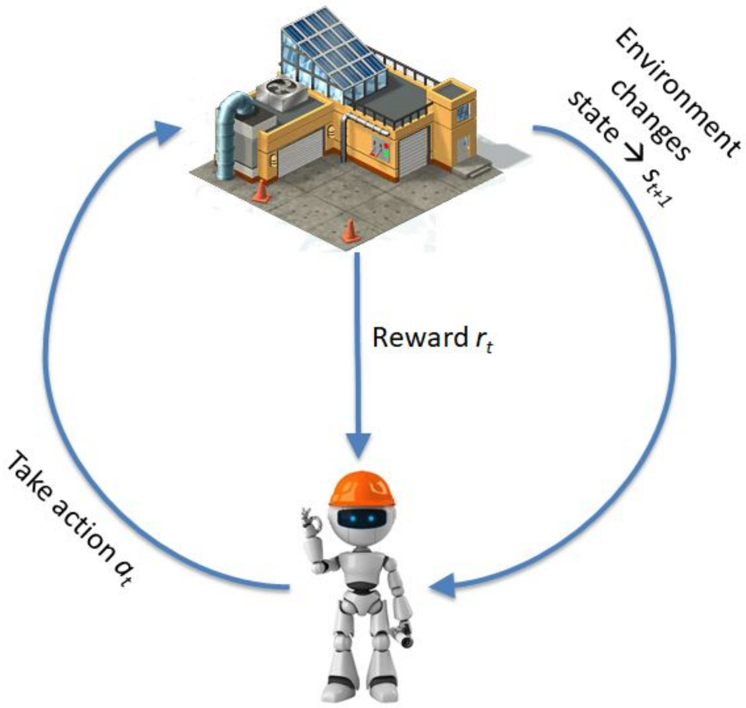
Interacting with an environment

Decision maker (agent) exists within an environment

Agent takes **actions** based on the environment **state**

Environment **state** updates

Agent receives feedback as **rewards**



A model for decision making

Markov Decision Process (MDP)

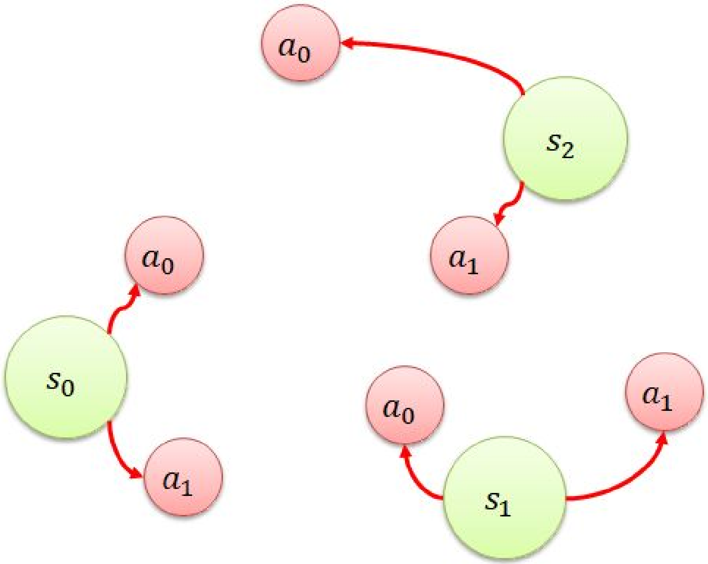
M = $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \gamma \rangle$

A model for decision making

Markov Decision Process (MDP)

M = $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \boldsymbol{\gamma} \rangle$

- States: encode world configurations
- Actions: choices made by agent



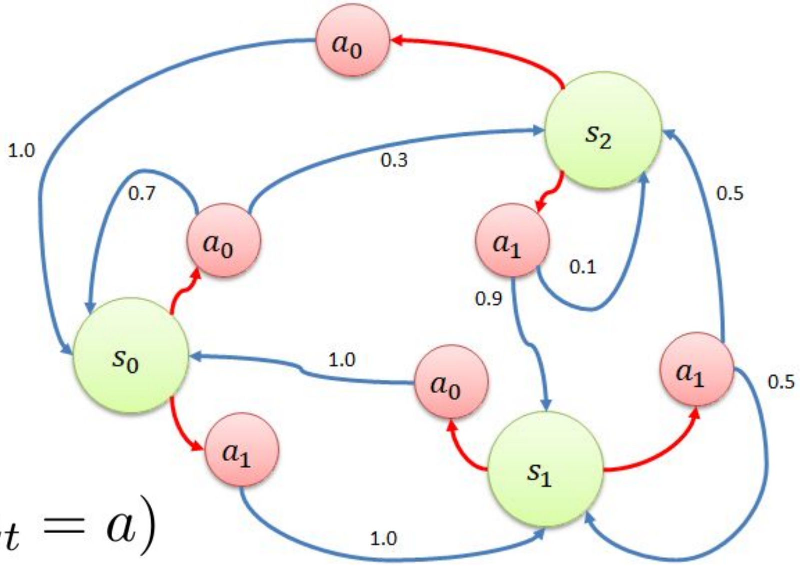
A model for decision making

Markov Decision Process (MDP)

M = ⟨S, A, T, R, γ⟩

Transition function: how the world evolves under actions

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$



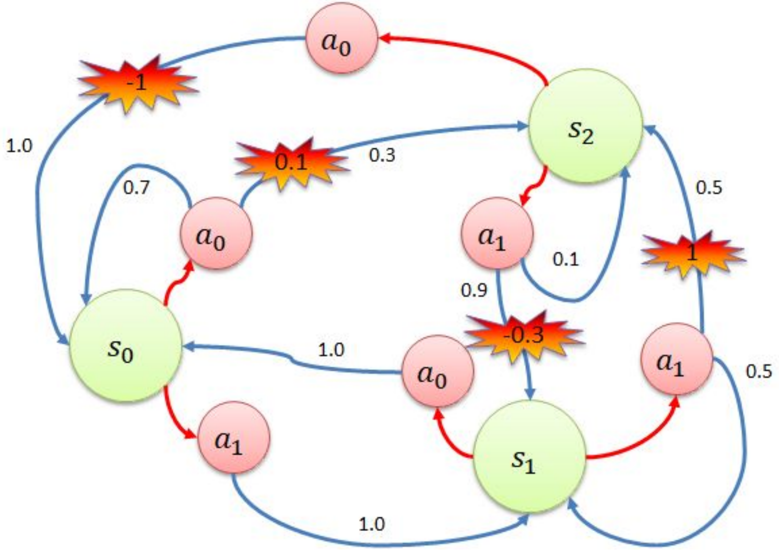
A model for decision making

Markov Decision Process (MDP)

M = $\langle \color{green}{S}, \color{red}{A}, \color{blue}{T}, \color{orange}{R}, \gamma \rangle$

Rewards: feedback signal to agent

$$R(s,a) = E[r_t | s_t = s, a_t = a]$$

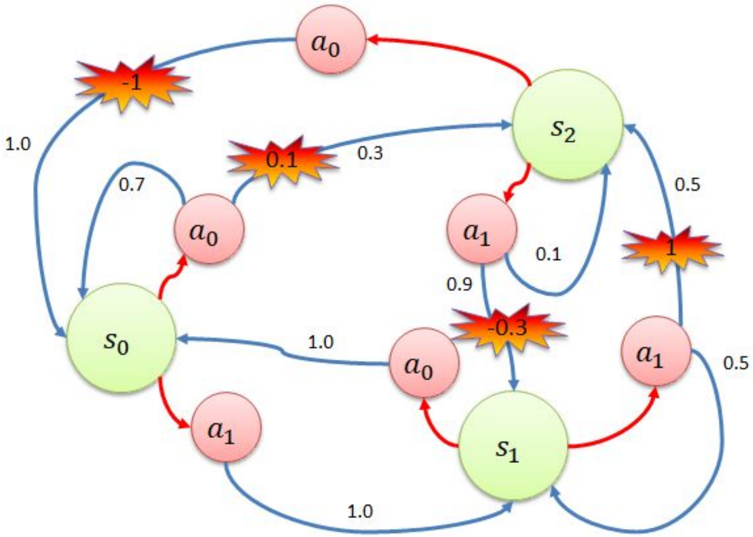


A model for decision making

Markov Decision Process (MDP)

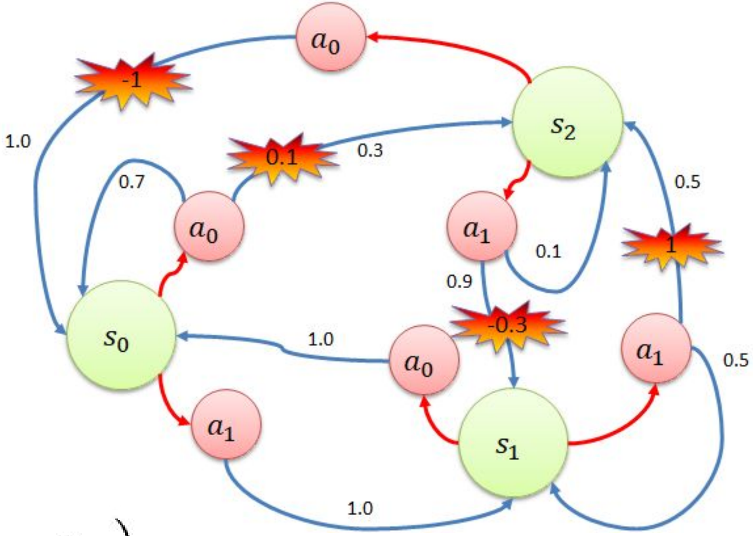
M = $\langle S, A, T, R, \gamma \rangle$

$\gamma \in [0,1]$ discounting for future rewards



A model for decision making

Markov Decision Process (MDP)



M = $\langle S, A, T, R, \gamma \rangle$

Markov:

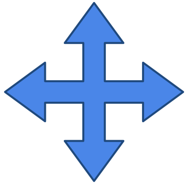
$$P(s_{t+1}|s_t) = P(s_{t+1}|s_0, ..., s_t)$$

"Future is independent of the past, given the present"

An example

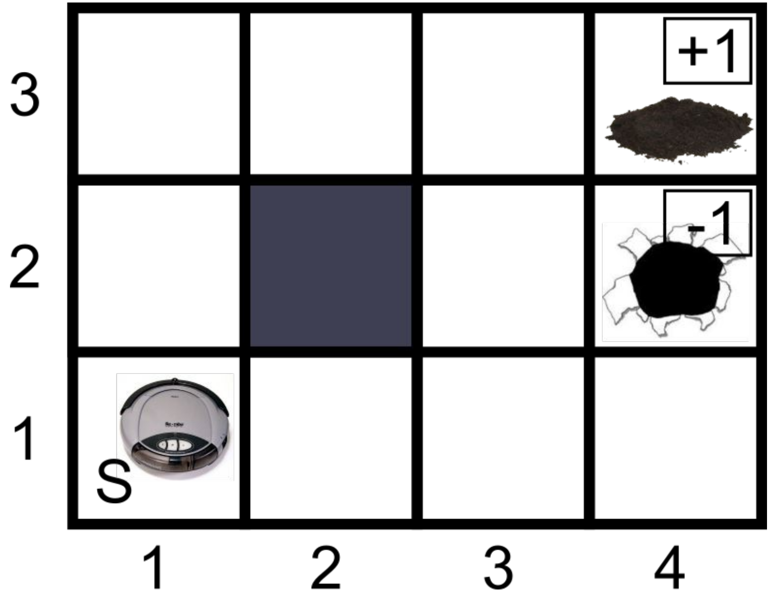
Cleaning Robot

Actions:



Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move

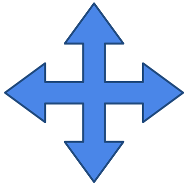


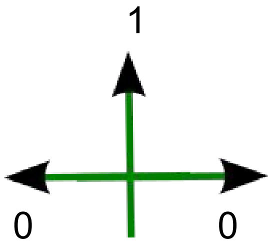
An example

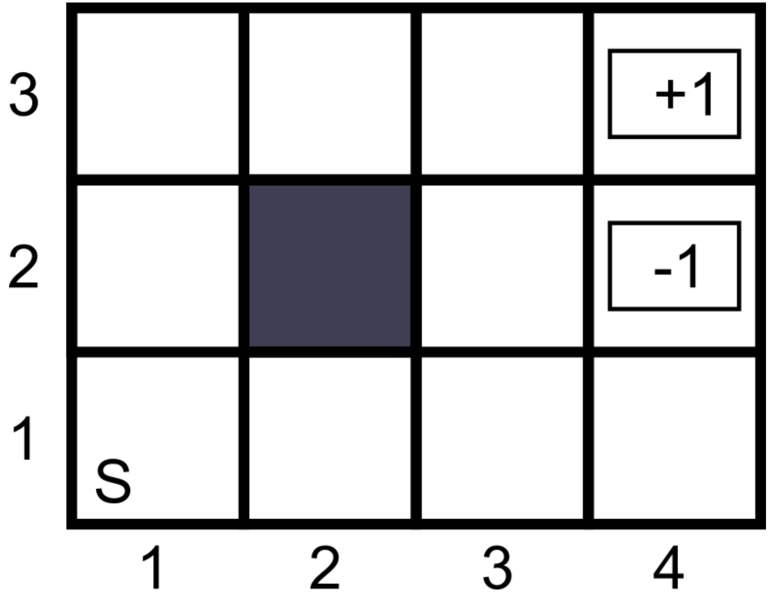
States:

- Position on grid e.g.
  - S is (1,1), goal (4,3)

Actions:



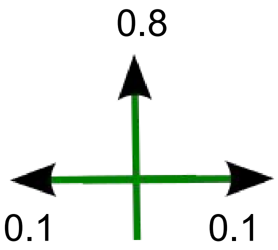


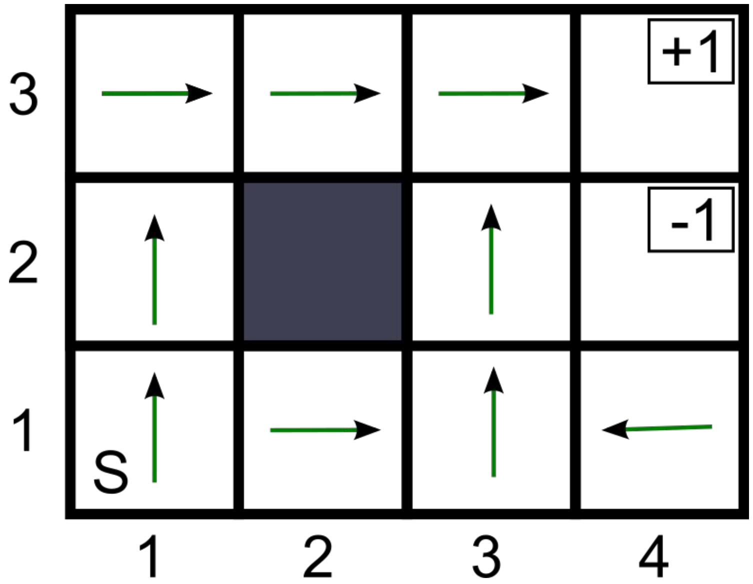


Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move

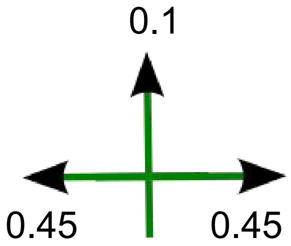
What is the optimal policy?





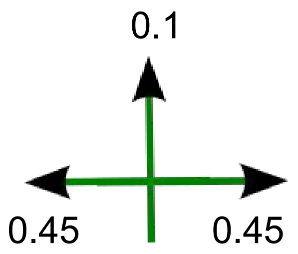
What is the optimal policy?

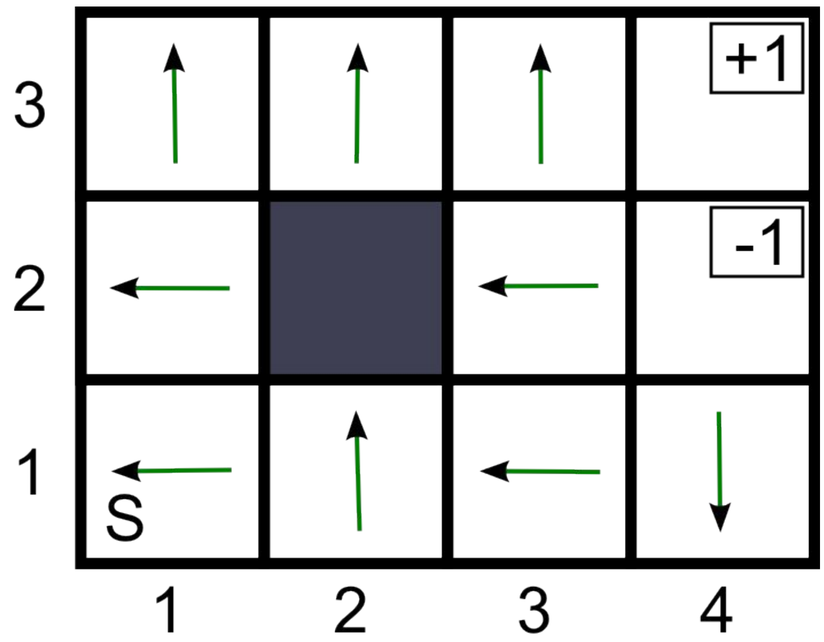
Change the action transitions?



What is the optimal policy?

Change the action transitions?



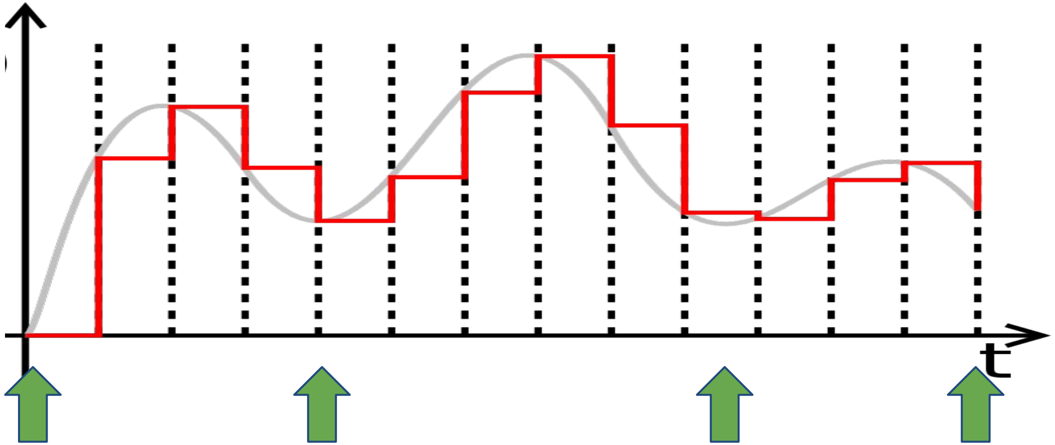


Practically, why RL?

- Treating disease in an individual
- Chronic disease (HIV, Cancer, Schizophrenia, etc.)
  - Not a single decision event

Information about:

- patient (demographics, family history)
- body (test results, etc.)
- disease (genomics, progression etc.)

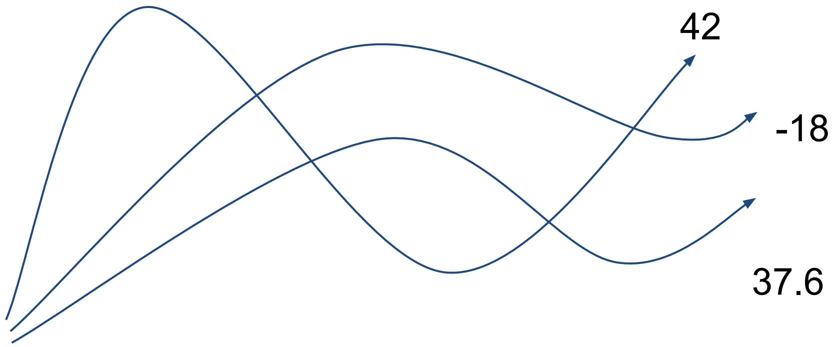


How do we find the best treatment strategy?

Evaluating behaviours

Many different trajectories
are possible through a space

Use the total **discounted accumulated rewards** to evaluate them



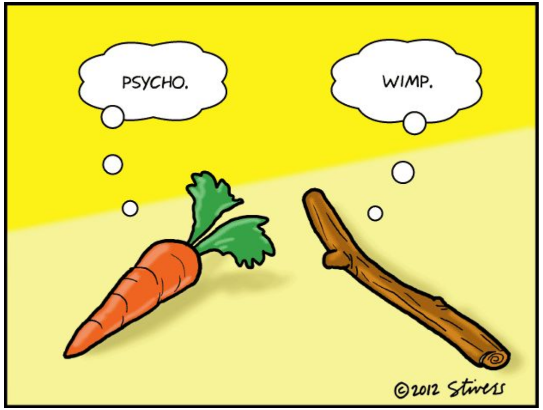
Rewards

Scalar feedback signal

Encode (un)desirable features of behaviours:

Winning/losing, collisions, taking expensive actions, ...

- Sparse
- Delayed
- Only have relative value



The Rats of Hanoi

Policies

A **policy** (or behaviour or strategy) π is any mapping from states to actions

- Deterministic or stochastic

$$\pi(a|s) = P(a_t = a | s_t = s)$$

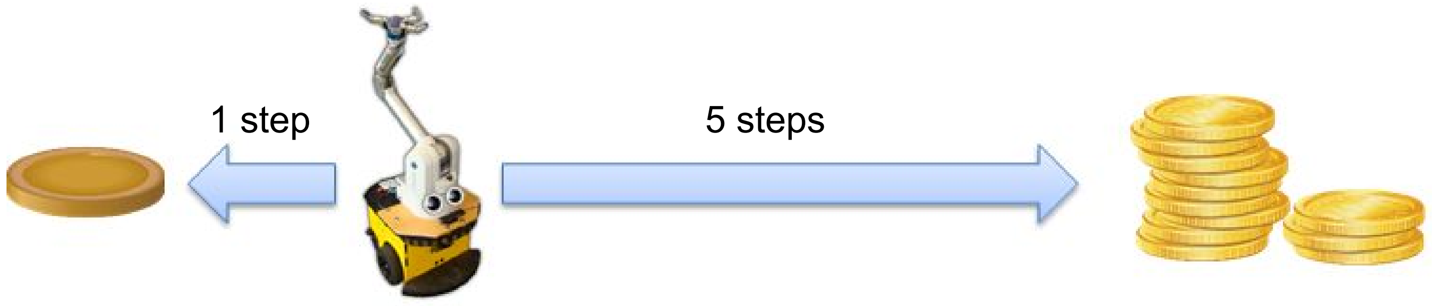
Optimal policy π^*

- Accumulates maximal rewards over a trajectory
- This is what we want to learn!

Immediate vs delayed rewards

Cannot just rely on the **instantaneous** reward function

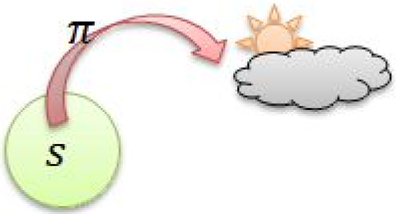
Tradeoff: don’t just act myopically (short term)



Notion of **value** to codify the goodness of a state, considering a policy running into the future

- Represented as a **value function**

Value Functions



Value function:

accumulated reward

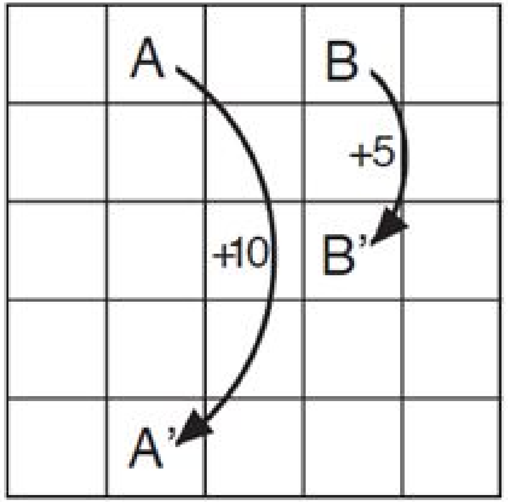
The expected return ($\boldsymbol{R}$) starting at state $\boldsymbol{s}$ and then executing policy $\boldsymbol{\pi}$

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t, s_{t+1})\}$$

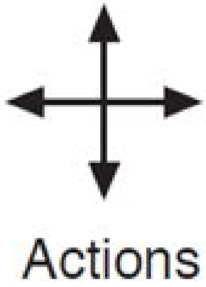
"How good is $\boldsymbol{s}$ under $\boldsymbol{\pi}$?"

Example Value Functions

Reward -1 for every move

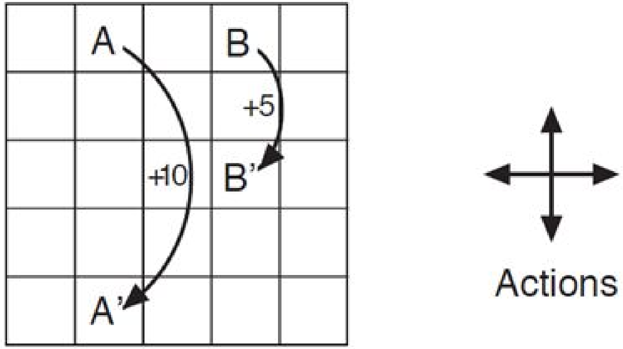


(a)



Example Value Functions

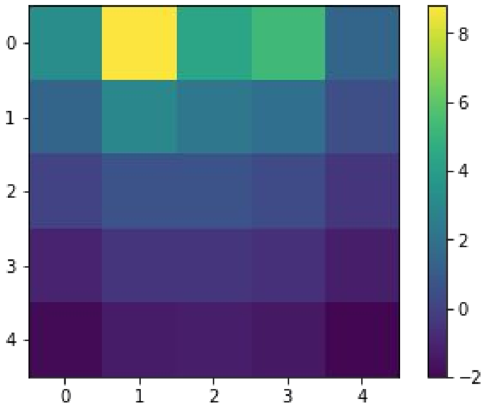
Random policy:



(a)

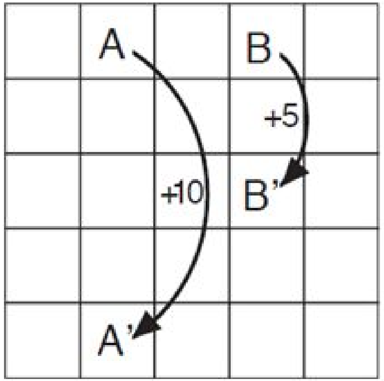
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)



Example Value Functions

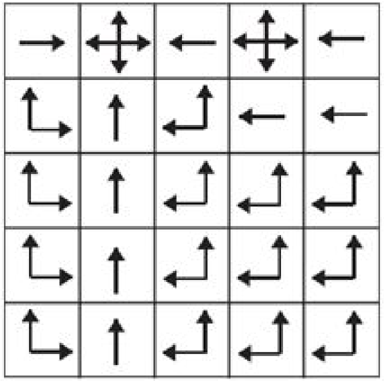
Optimal policy:



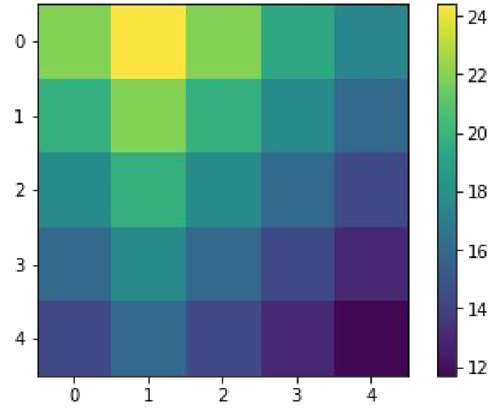
a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*



So what?

How do we use these ideas
to do something useful?

Value Functions: Recursion

$V(s)$ ⇒ expected return starting at s and following π

Suggests dependence on V(s') from next state s'

Bellman Equation:

$$\boxed{V^\pi(s)} = \boxed{R(s,\pi(s))} + \gamma \boxed{\sum_{s'}} \boxed{T(s,\pi(s),s')} \boxed{V^\pi(s')}$$

value of s

immediate reward

for all possible next states

the probability of reaching that state with π

value of s'

Value Functions: Optimality

Similarly, for an optimal policy π^* with optimal value function V^*:

Bellman Optimality Equation:

$$\boxed{V^*(s)} = \boxed{\max_a}\{R(s,a) + \gamma \sum_{s'} T(s,a,s')\boxed{V^*(s')}$$

take the best possible action

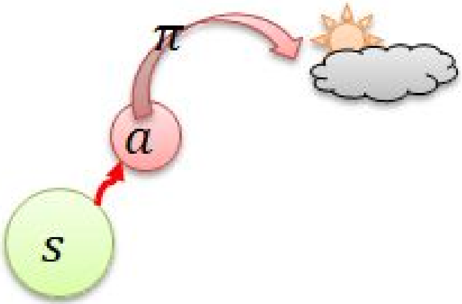
Value Functions

Action-value function:

$$\boxed{Q(s,a)} = \sum_{s'} \underbrace{\boxed{T(s,a,s')}}_{\text{transition probability}} \left(R(s,a,s') + \gamma \boxed{V(s')}\right)$$

The expected return (***R***) starting at state ***s*** and executing action ***a***, and then following policy ***π***

“How good is ***a*** in ***s*** under ***π***?”



Optimal policies and value functions

$\pi^*(a|s) :=$ $\quad$ 1 if $a = argmax\ Q^*(s,a)$,
0 otherwise

Move in direction of greatest value

Finding Q^* (or V^*) is equivalent to finding π^*

Every MDP has an optimal policy

The goal of RL

Given this formulation,

how do we learn a policy?

Solving Bellman

Given the Bellman equation

$$V^*(s) = \max_a \{R(s,a) + \gamma \sum_{s'} T(s'|s,a)V^*(s')\}$$

Solve this as a large system of value function equations

- But: non-linear (max operator)
- So: solve iteratively

What are we trying to do here?

- Learn how good each state of the world is, when looking perfectly into the future

Dynamic Programming

Value Iteration: Dynamic Programming

(T,R,S,A)

- Iteratively update V (synchronous version)
- At each iteration ***i***:
  - For all states ***s*** in ***S***:
    - Update ***V(s)***

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V_i(s') \right) \right\}$$

But: this requires the full MDP!!

In general, T and R are unknown

Value Based Methods

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

CSIR
our future through science

Algorithm setup

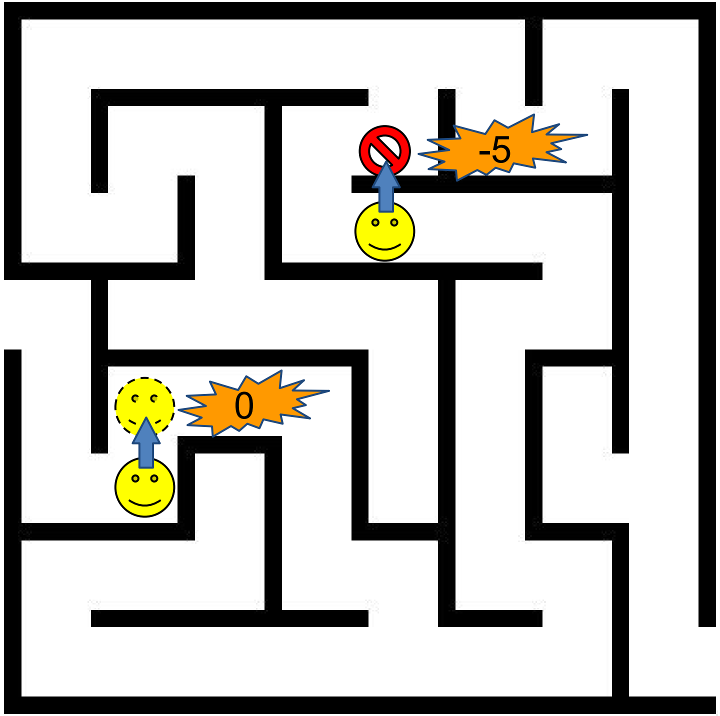
Value Based Methods:

- No Transition Model
- No Reward Model
- Access to environment for experiment or access to training data ***(s,a,r,s')***
- **Goal:** Learn Value of States, State-Actions
  - Policy through learned values

Data generation

T and R unknown!

Instead, generate samples of training data ***(s,a,r,s')*** from environment

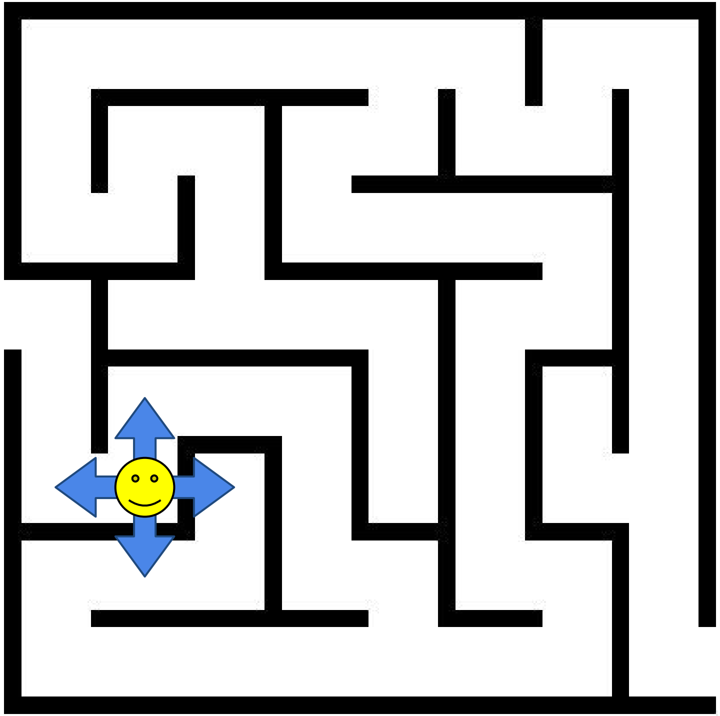


Learning from Experience

We need

- **A method to choose actions**

- Some model to keep track of and learn
  - *Value Function*



The Bandit Problem

Consider a row of one-arm bandit machines in a casino

Set of “arms” (actions) that each generate rewards from different distributions

Exploration vs exploitation

Action selection

The exploration-exploitation tradeoff!

Maximizing expected returns means balancing between:

- ***Exploiting*** gained knowledge (greedy)
  - Take the best known action
- ***Exploring*** new actions/states (random)
  - Try something new

Action selection strategies

ε-Greedy ($0 < \varepsilon \leq 1$):

- With probability $1-\varepsilon$ exploit
  - Choose the best action for a state
- With probability ε explore
  - Randomly choose action

ε usually higher at beginning of learning, decay later

Softmax

- Sample action given softmax

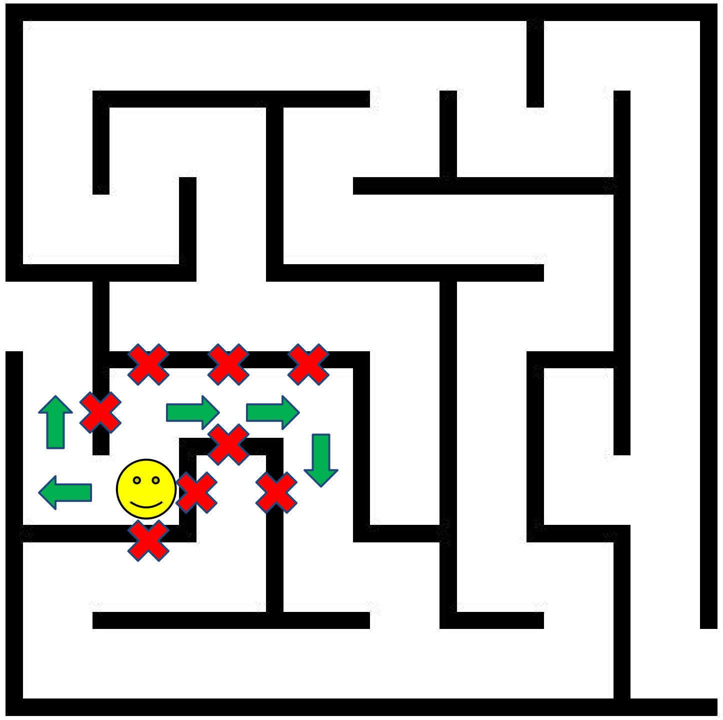
$$P_t(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(Q_t(i)/\tau)},$$

Learning from Experience

We need

- A method to choose actions

- **Some model to keep track of and learn**
  - ***Value Function***



TD Learning

(*T*,*R*,*S*,*A*)

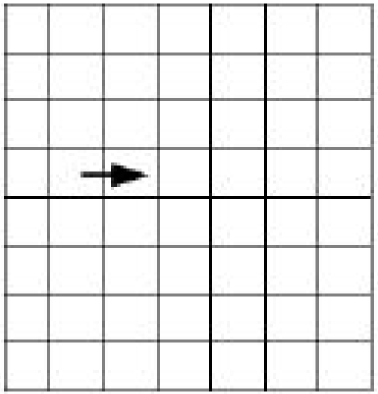
Temporal Difference (TD) Learning:

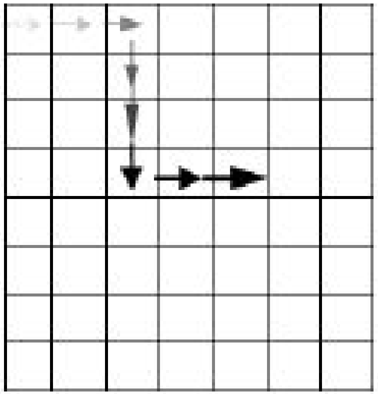
- Initialise ***V*** for all ***s*** in ***S***
- For each experience tuple (***s***,***r***,***s'***) under policy ***π***:
  - Update ***V***: $$V_{i+1}(s) \leftarrow V_i(s) + \alpha(\underbrace{\overbrace{r + \gamma V_i(s')}^{\textit{learnt value}} - V_i(s)}_{\text{TD error}})$$

estimated return
(TD target)

Eligibility traces

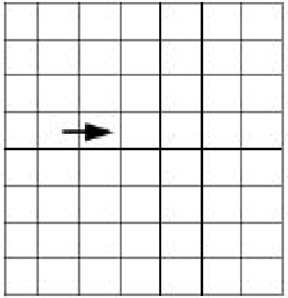
- Keep track of where agent has been
- More efficient updates





$$V_{i+1}(s) \leftarrow V_i(s) + \alpha(\overbrace{r + \gamma V_i(s')}^{learnt\ value} - V_i(s))$$

TD(0)



(T,R,S,A)

TD(0) Learning:

- Initialise V for all **s**
- For each trajectory/episode:
  - for all s
    - $e(s) = 0$
  - for each experience tuple (***s,r,s'***) under policy $\boldsymbol{\pi}$ in episode:
    - $e(s) = e(s) + 1$
    - $\delta = r + \gamma V(s') - V(s)$
    - for all ***s*** *in* ***S***
      - $V(s) \leftarrow V(s) + \alpha\delta e(s)$
    - $e(s) = 0$

We are back to normal TD Learning.

TD rollouts

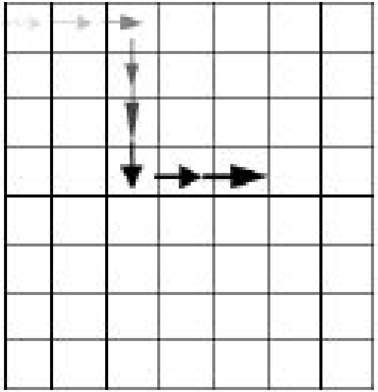
(s, a, r, s') (s', a', r', s'') (s'', a'', r'', s''')

(T,R,S,A)

1-step TD
and TD(0)

2-step TD

3-step TD



$V_{t+1}(s) \leftarrow V_t(s) +$ $\overbrace{\gamma V_t(s') - V_t}^{t \ value}$

$V(s) \leftarrow V(s) + \alpha$ $r_{t+1} + \gamma^2 V(s'') - V(s))$

$$V(s) \leftarrow V(s) + \alpha(r + \gamma r' + \gamma^2 r'' + \gamma^3 V(s''') - V(s))$$

TD(1)

TD(1) Learning: *(T,R,S,A)*

- Initialise *V* for all **s**
- For each trajectory/episode:
  - for all s
    - *e(s) = 0*
  - for each experience tuple (***s,r,s'***) under policy ***π*** in episode:
    - *e(s) = e(s) + 1* Mark whole trajectory
    - $\delta = r + \gamma V(s') - V(s)$
    - for all ***s*** *in* ***S***
      - $V(s) \leftarrow V(s) + \alpha\delta e(s)$
    - *e(s) = γe(s)* Decay trace

$$\boxed{V(s) \leftarrow V(s) + \alpha(r_t + \gamma r_{t+1} + \gamma^2 V(s'') - V(s))}$$

Tuning the decay

TD(0)

No traces

TD(1)

Traces
decay with γ

TD(λ)

Control the
decay rate

TD(λ)

TD(λ) Learning: ***(T,R,S,A)***

- Initialise V for all **s**
- For each trajectory/episode:
  - for all s
    - $e(s) = 0$
  - for each experience tuple (***s,r,s'***) under policy π in episode:
    - $e(s) = e(s) + 1$
    - $\delta = r + \gamma V(s') - V(s)$
    - for all ***s in S***
      - $V(s) \leftarrow V(s) + \alpha\delta e(s)$
    - $e(s) = \gamma\lambda e(s)$

Control the speed of decay

Intermission

15 minutes

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

CSIR

our future through science

Onwards from TD

Recap: we can now learn by estimating **V** from experience

But:

- Not using actions ***A***

- We would rather learn ***Q***,

  for easier policy extraction!

  V requires a one-step lookahead model

SARSA

Learn from **s, a, r, s', a'**

(*T*,*R*,*S*,*A*)

- Initialise ***Q*** for all **s, a**
- For each episode
  - Initialise s_0
  - Choose a_0 in s_0 from ***Q***
  - For each step *t* in episode
    - Take a_t, observe r, s_{t+1}
    - Choose a_{t+1} in s_{t+1} from ***Q***

act

look ahead

learn

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{Q(s_{t+1}, a_{t+1})}_{\text{estimate of future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

SARSA

(*T*,*R*,*S*,*A*)

Where did we get the a_{t+1} ?

- Taking the next action under **Q**
- This is an **on policy** algorithm

What about **off policy**?

- Learn about optimal policy while exploring
- Reuse experience from other policies
- Learn from observations

Q-Learning

(*T*,*R*,*S*,*A*)

- Initialise ***Q*** for all ***s, a***
- For each episode
  - Initialise s_0
  - For each step ***t*** in episode
    - Choose a_t in s_t from **Q**
    - Take a_t , observe r, s_{t+1}

act

learn

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\boxed{\max_a Q(s_{t+1},a)}}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

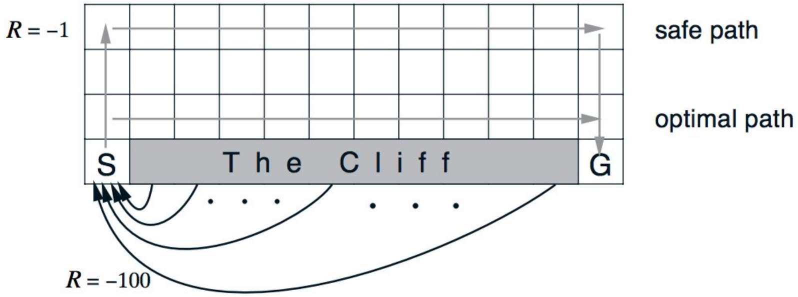
take best next action
(so far)

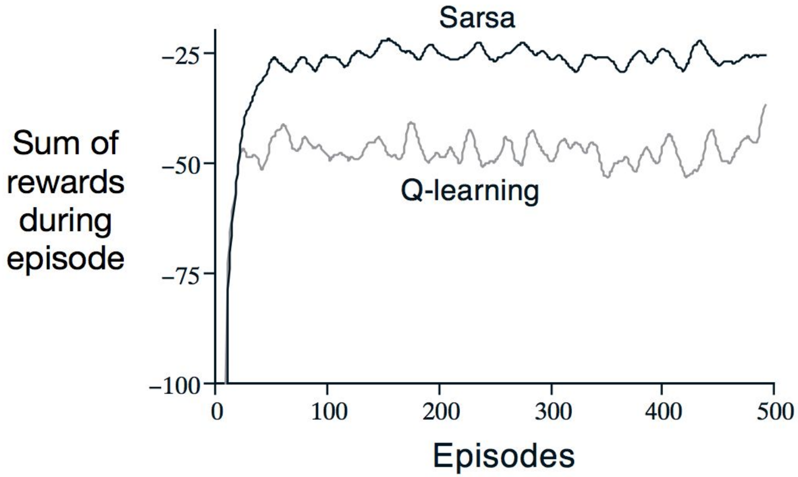
Q-Learning demo

(T,R,S,A)

Shreyas Skandan: https://www.youtube.com/watch?v=RTu7G0y4Os4

Typical Learning Curves





Generalising...

What about extending behaviour to different tasks?

What about building a simulator?

Ask questions about the domain

Solution: we need a **model**!!!

Model Based Methods

From Values to Environment Models

Model based reinforcement learning

Learn a model (***T*** and ***R***) from experience

- Supervised learning problem

Models let you predict next state and reward

- Reason about uncertainty

Algorithm setup

Model Based RL:

(T,R,S,A)

- No Transition Model
- No Reward Model
- Access to environment for experiment or access to training data ***(s,a,r,s')***
- **Goal**: Learn Transition and Reward Models
  - Policy through learned models.

Model Based RL

Learn a Transition and Reward Model

(T,R,S,A)

On receiving experience (s_t, a_t, r_t, s_{t+1}):

$$R(s_t, a_t) \Leftarrow R(s_t, a_t) + \alpha(r - R(s_t, a_t))$$

$$T(s_t, a_t, s_{t+1}) \Leftarrow T(s_t, a_t, s_{t+1}) + \alpha(1 - T(s_t, a_t, s_{t+1}))$$

$$T(s_t, a_t, \hat{s}) \Leftarrow T(s_t, a_t, \hat{s}) + \alpha(0 - T(s_t, a_t, \hat{s}))$$

$$Q(s,a) = R(s,a,s') + \gamma \sum T(s,a,s')V(s')$$

Dyna Q Algorithm

For each step ***t*** in episode

- Choose a_t in s_t from ***Q***
- Take a_t, observe r, s_{t+1}
- Update ***Q***: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\}$

Q-learning

- Given (s_t, a_t, r_t, s_{t+1})
- Update ***T*** and ***R***

model update

- Repeat *n* times:
  - Sample previously observed ***s***
  - Sample previously taken ***a*** (in ***s***)
  - Get ***r*** and ***s'*** from model
  - Update ***Q:*** $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\}$

sample model to update Q

What else can I do with a model?

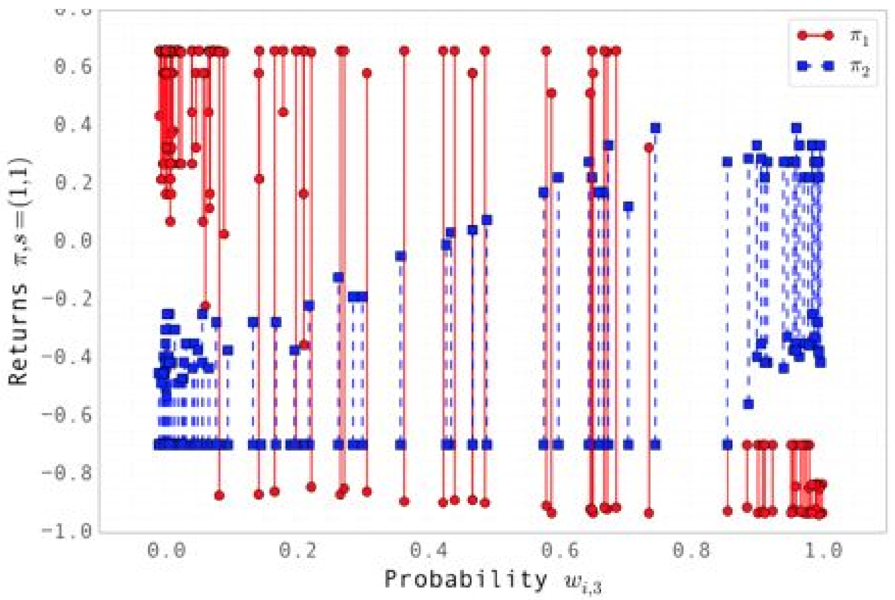
Quantify uncertainty in value functions

Uncertainty from:

- Data sparsity
- Inherent stochasticity
- Latent structure

Approaches:

- Monte Carlo sampling
- Simulation



A little bit of overkill?

Ok, so we've gone to all this trouble to learn $T, R \rightarrow Q$ …

Can't we just learn the policy?

Policy Search

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

CSIR
our future through science

Algorithm setup

Direct Policy Learning:

(T,R,S,A)

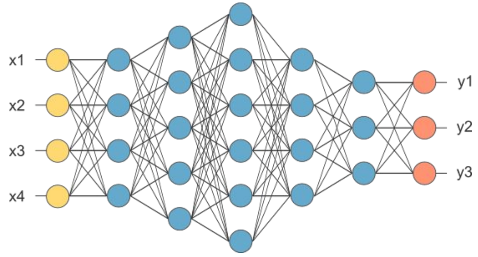
- No Transition Model
- No Reward Model
- Access to environment for experiment or access to training data ***(s,a,r,s')***
- **Goal**: Learn policy directly

Policy Gradient

Parametrise policy: $\pi_\theta(s,a) = P(a|s,\theta)$

Choices:

- Linear combination of basis functions
- Set of state features
- Deep neural network



Goal: find best θ

Optimisation problem!

Optimising the policy

Define cost function $J(\theta)$:

Start value, average reward per time step…

Find θ that maximises $J(\theta)$

e.g. gradient ascent on:

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

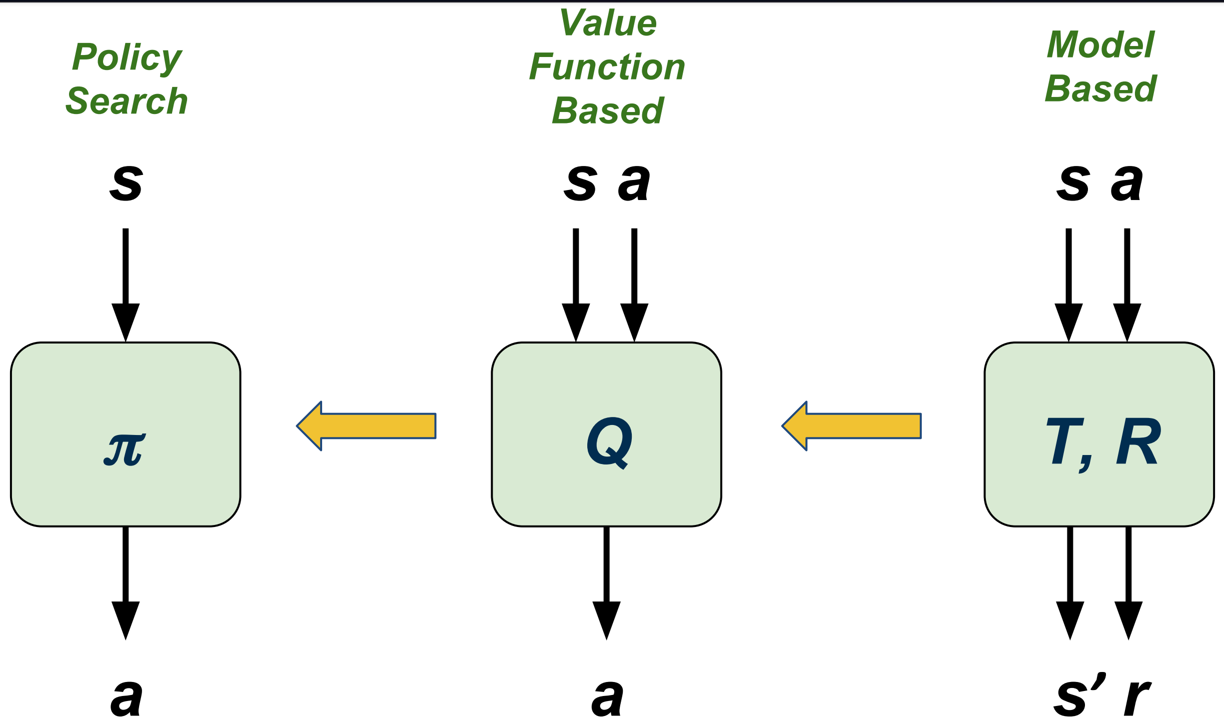
policy gradient

Why policy gradient?

+ High-dimensional action spaces
+ Continuous action spaces
+ Many recent successes in robotics

- Local convergence
- Policy evaluation high variance

Recap - RL Approaches



Inverse Reinforcement Learning

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

CSIR

our future through science

Inferring a Reward Function

Designing reward functions is hard!

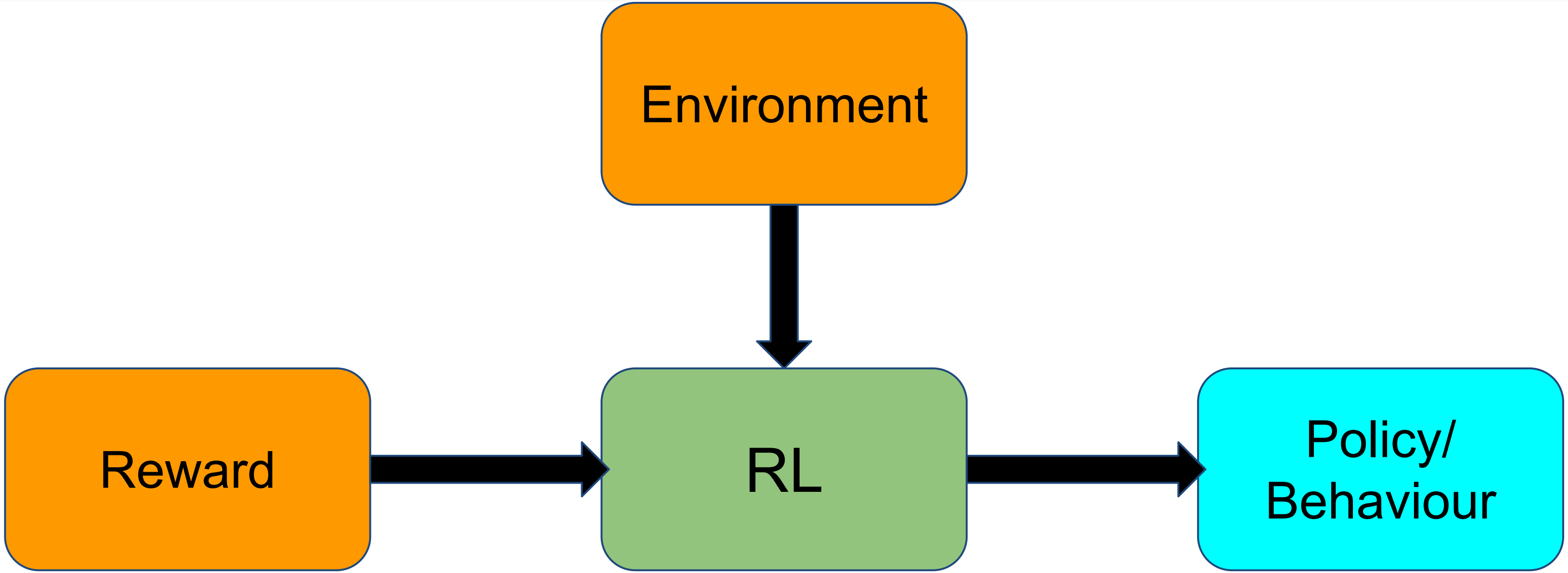
- Often not clear what should be done or how it should be rewarded
- Where do these come from?

Learn the incentives that explain observed behaviour

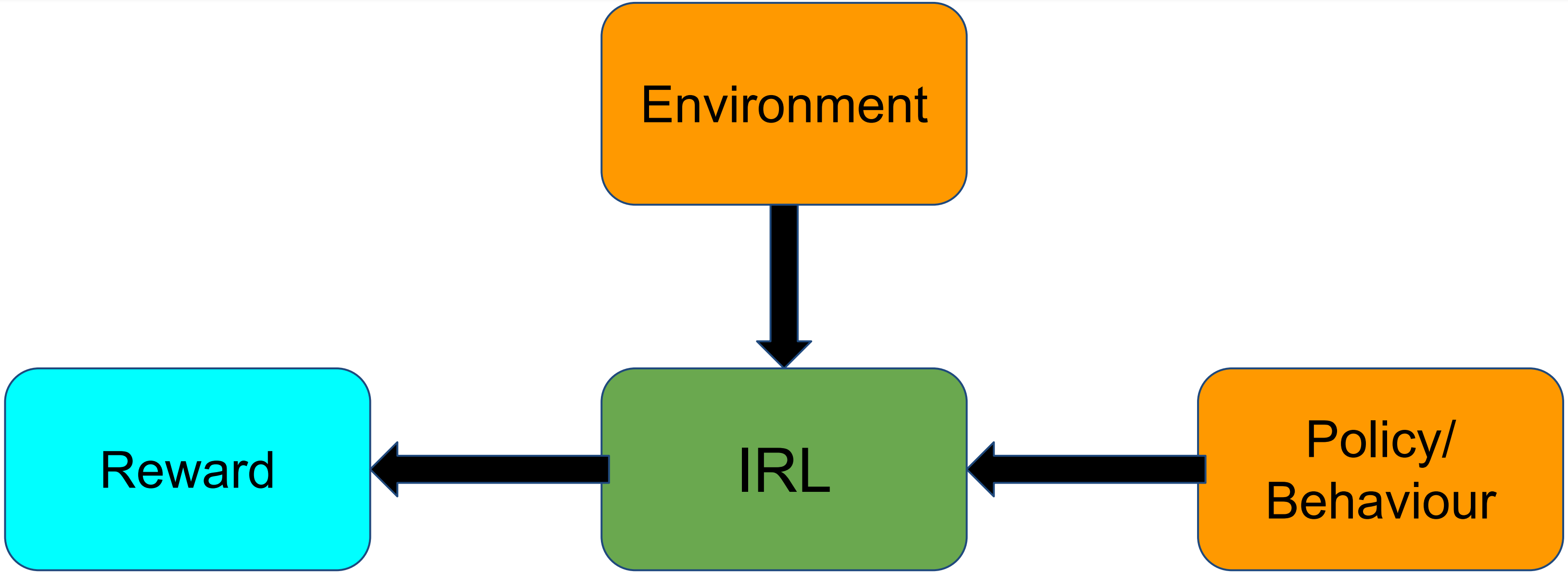
- From an “expert”

We do not observe the reward, but want to learn it

Inverse Reinforcement Learning



Inverse Reinforcement Learning



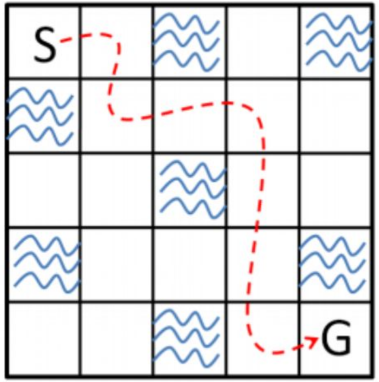
Algorithm setup

Inverse RL:

(T,R,S,A)

- Transition Model **(Can be learned)**
- No Reward Model
- Observe training data ***(s,a,s')***
- **Goal**: Learn a reward model to explain the behaviour observed through the training data

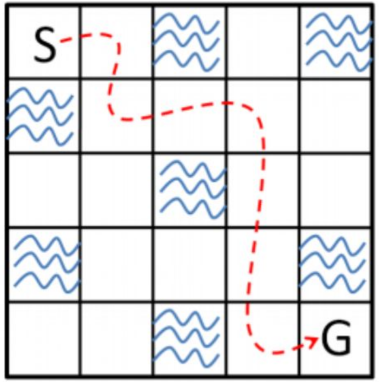
IRL: From paths to rewards



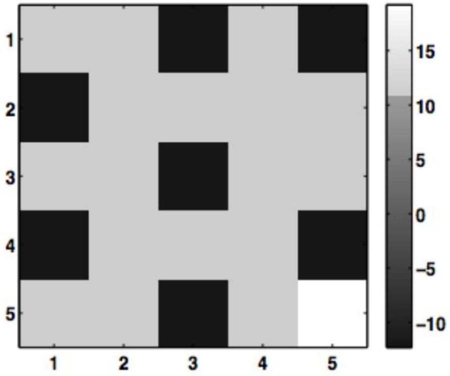
- Observe trajectory/trajectories (s,a,s')
- Would like to know:
  - What was the goal of the agent?
  - What was the reward?

Get to ***G*** and avoid water?

Maximum Likelihood IRL



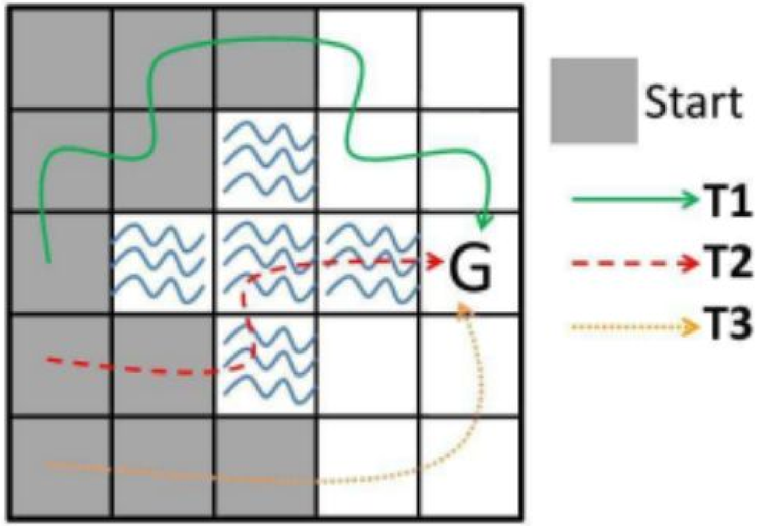
Possible reward function



ML IRL Algorithm (Intuition):

- Given sample trajectories D
- Initialise a reward function $\mathcal{R}$
- Calculate policy from $\mathcal{R}$, T
- Calculate $P(\boldsymbol{D}|\boldsymbol{\pi})$
- Calculate gradient, update $\mathcal{R}$

IRL: From paths to rewards



What about different teachers?

Information not in the data when we get it.

MLIRL with multiple intentions!!!

M Babes et. al. Apprenticeship learning about multiple intentions

IRL

Learn from demonstration

- Crowdsourcing
- Showing tasks to robots
- Learning from experts

(Some) Reinforcement Learning Applications

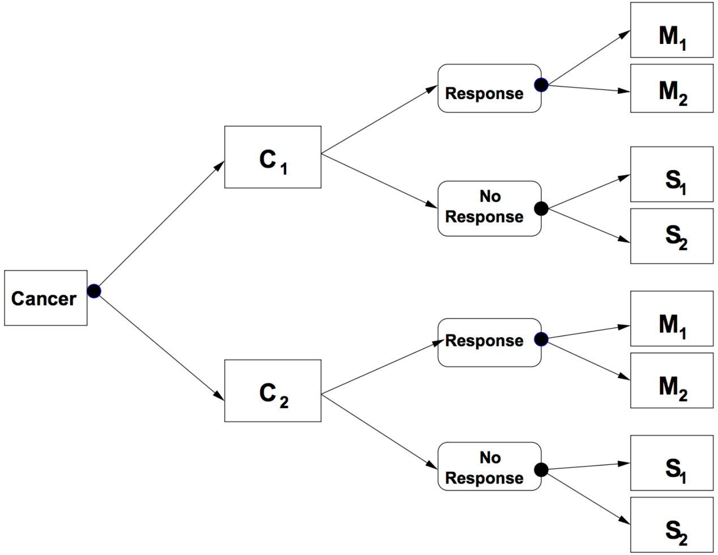
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

CSIR

our future through science

Application Areas

Randomised Controlled Trials



Efficacy in Sequential Multiple Assignment Randomized Trial

An Introduction to Dynamic Treatment Regimes: Marie Davidian

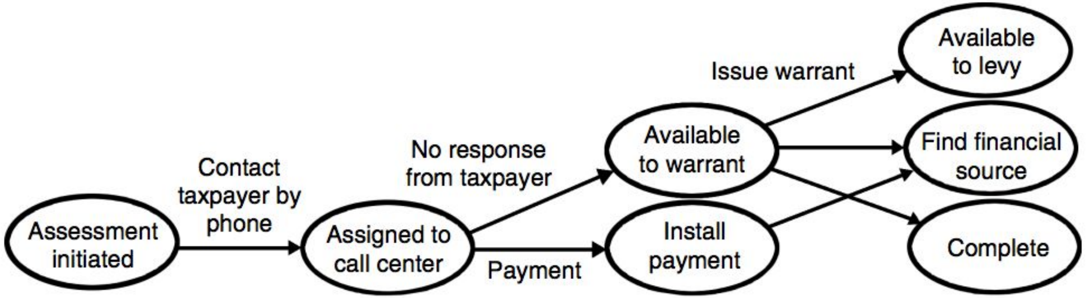
Application Areas

Advertising :(

Nuff Said!!!

Application Areas

Strategies to Improve Donations or Collecting Taxes :)



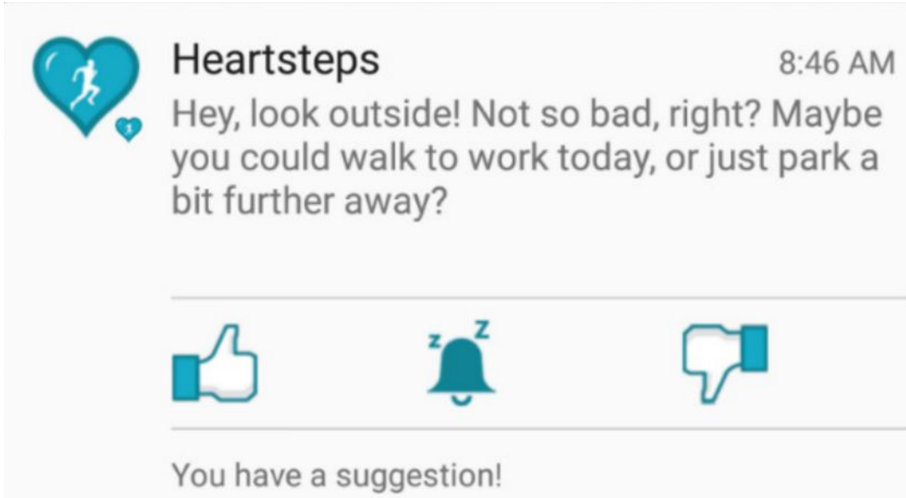
Action
Collections actions
Contact taxpayer by mail
Contact taxpayer by telephone
Create warrant
Create income execution
Create levy
Movement actions
Move to district office
Move to high-value team
Move to collection vendors
Move to indiv. case enforcement
Organization-specific action
Perform field visit
No action
Take no action

Tax Collections Optimization for New York State - Gerard Miller et. al.

Application Areas

Mobile Health Interventions

Heartsteps



Experimental Design & Machine Learning Opportunities in Mobile Health: Susan Murphy

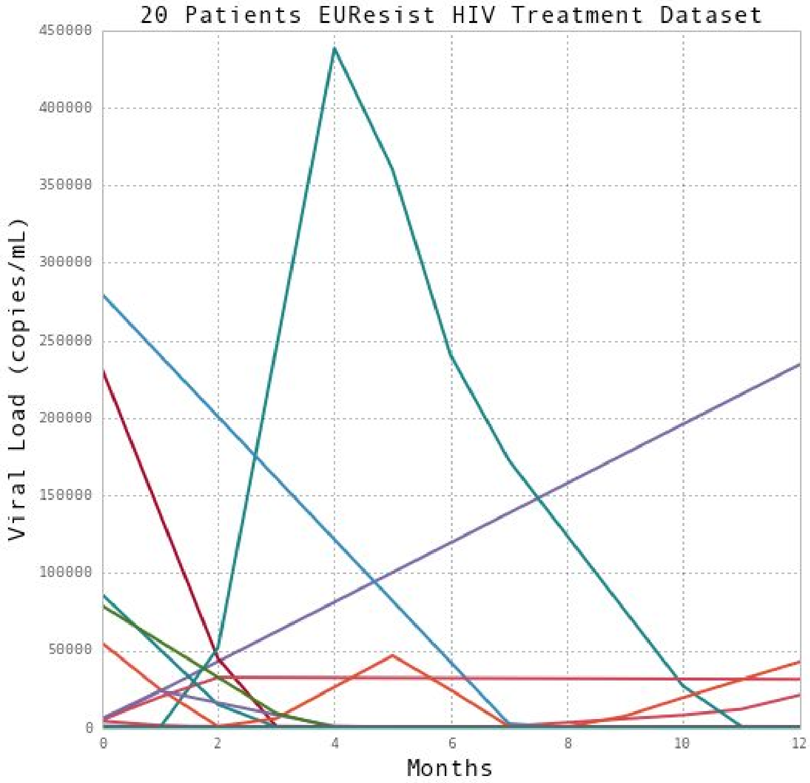
HIV Treatment: Possible Formulation

Features:
- baseline viral load, CD4 count,
- baseline CD4 percentage,
- Age, # previous treatments.

States:
- Viral Load tracked monthly over 24 months.
- Patient's treatment stage
- bins for the viral load, in copies/mL, were [0.0,50,100,1K,100K].

Actions:
- Therapy/drug cocktail groups occurring in the data set.

Reward:
- Negated AUC



V Marivate: Improved empirical methods in reinforcement-learning evaluation

Application Areas

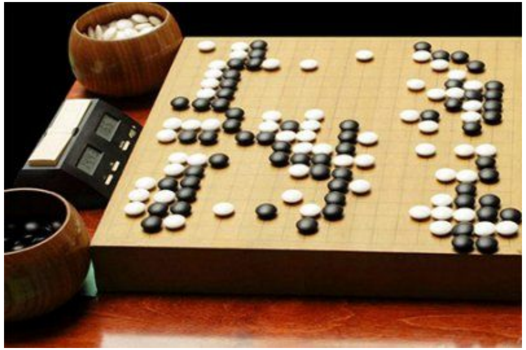
Robotics:

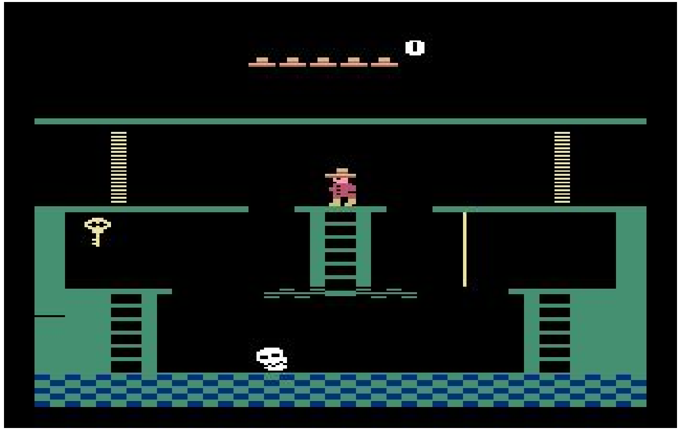
learning behaviours

RL Application Areas

Games

- Standardised testbeds
- Long decision horizons





Application Areas

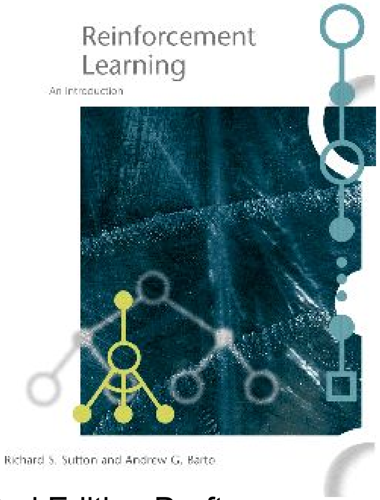
Automated Trading

1:

2: ???

3:

Thank you + Resources



2nd Edition Draft Recommended. Draft available online http://incompleteideas.net/sutton/book/the-book-2nd.html

UDACITY

RL class: https://www.udacity.com/course/reinforcement-learning--ud600

Vukosi Marivate and Benjamin Rosman

vmarivate@csir.co.za, brosman@csir.co.za