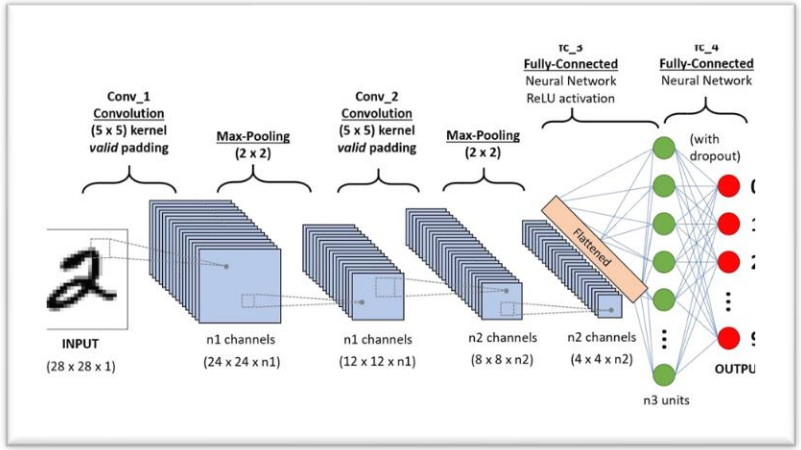
Madan Dabbeeru
Rakuten Institute of Technology, Bengaluru

What is Learning ?

$$\hat{y} = \beta_0 + \beta_1 X$$

Predicted value — Intercept — Slope — Predictor



"A child of 9 ... not only knows what 29 x 29 is, but also can work out what 17 x 17

Types of Learning

- Supervised (inductive) learning
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions

- Supervised learning
  - Decision tree induction
  - Rule induction
  - Instance-based learning
  - Bayesian learning
  - Neural networks
  - Support vector machines
  - Model ensembles
  - Learning theory
- Unsupervised learning
  - Clustering
  - Dimensionality reduction

Supervised (inductive) learning

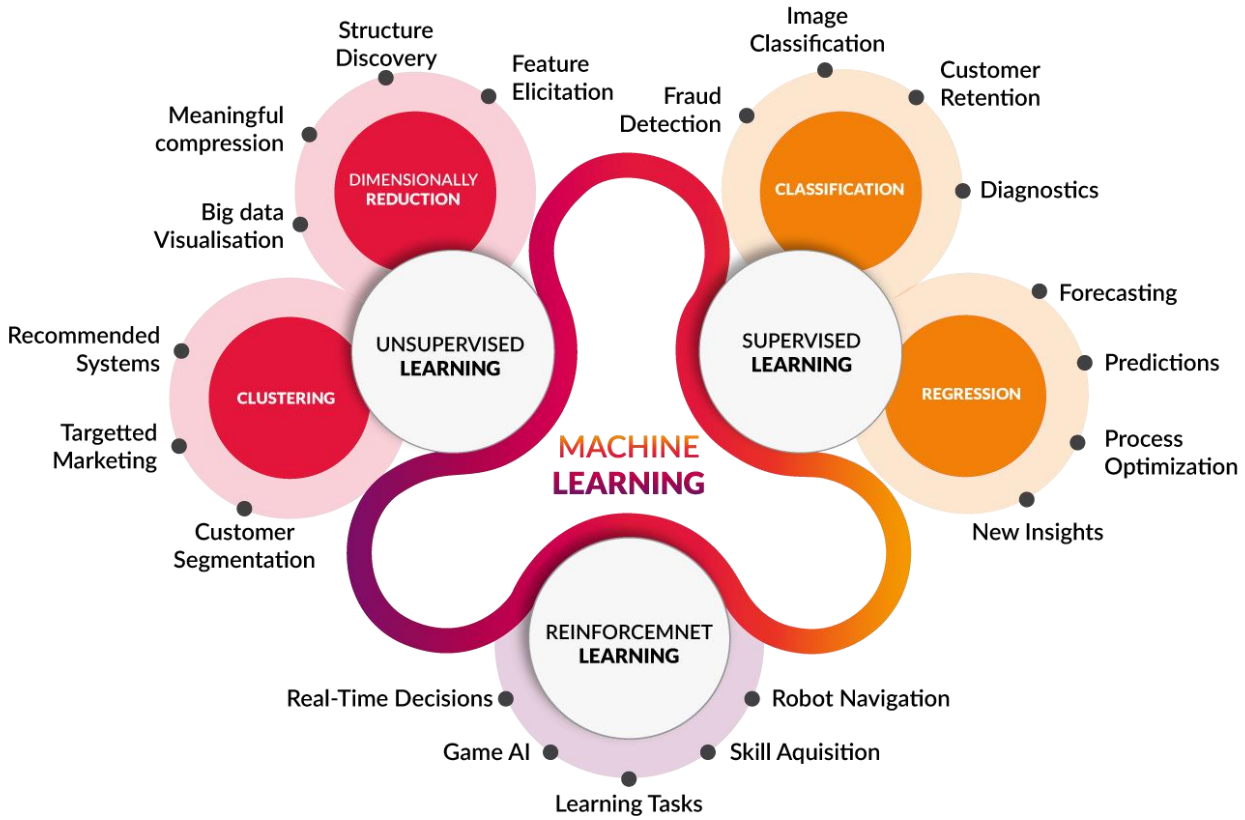
- Given examples of a function (X, F(X))
- Predict function F(X) for new examples X
  - Discrete F(X): Classification
  - Continuous F(X): Regression
  - F(X) = Probability(X): Probability estimation

ML in Practice

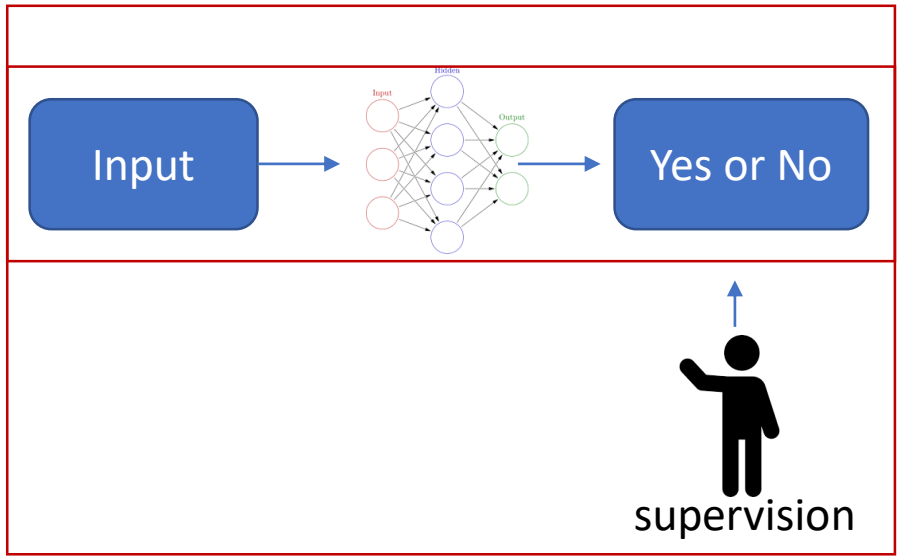
- Understanding domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learning models
- Interpreting results
- Consolidating and deploying discovered knowledge
- Loop

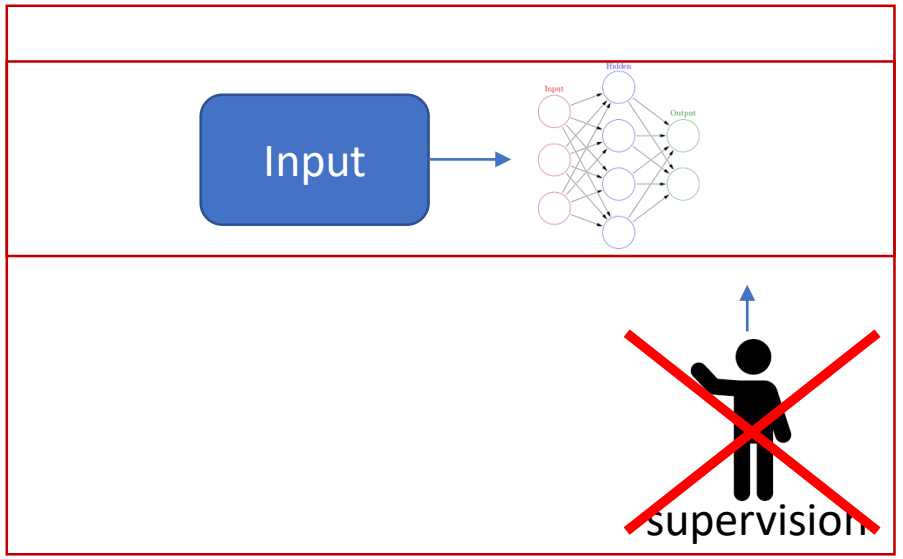
Introduction to Reinforcement Learning

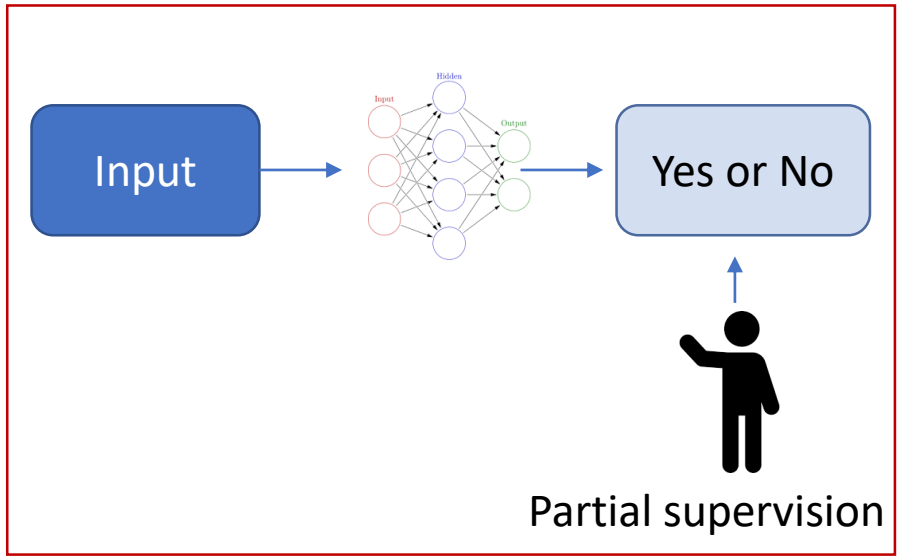
What is Machine Learning

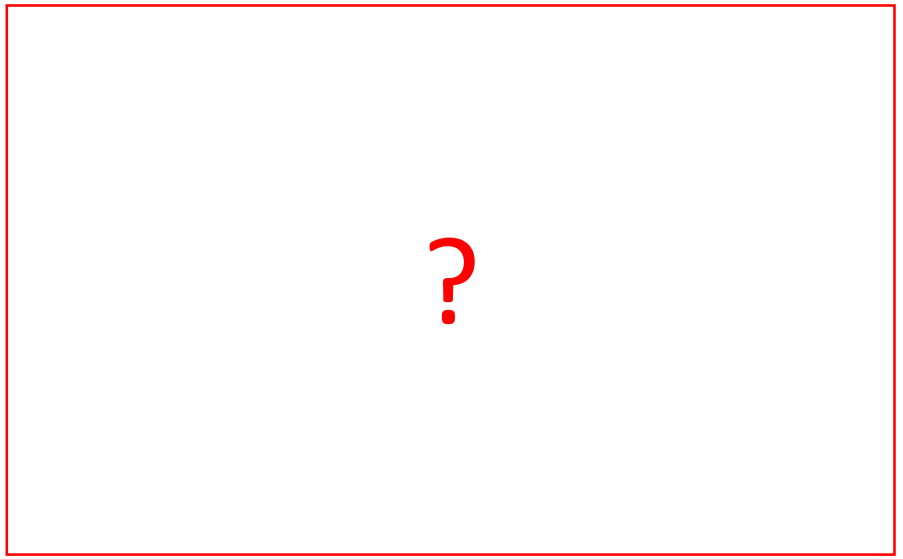


1. Supervised Learning
2. Semi-supervised Learning
3. Unsupervised Learning
4. Reinforcement Learning


Input
Hidden
Input
Output
Yes or No
supervision

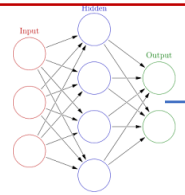

Input
Hidden
Input
Output
supervision


Input
Hidden
Input
Output
Yes or No
Partial supervision


?

Learn from examples

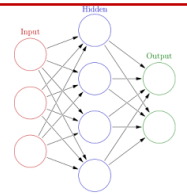
Input



Yes or No

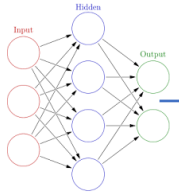
supervision

Input



supervision

Input



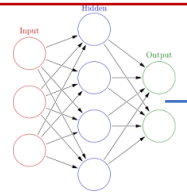
Yes or No

Partial supervision

Learn by experience

experiment

Input

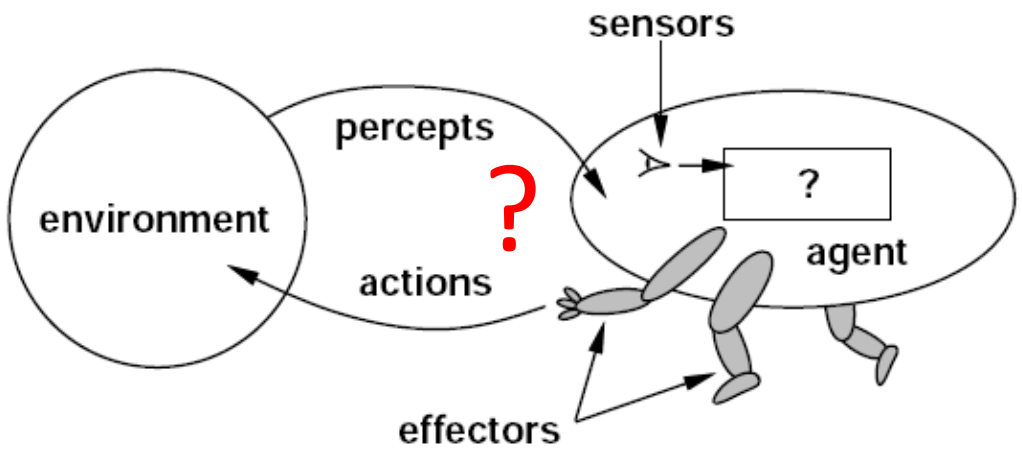


Action

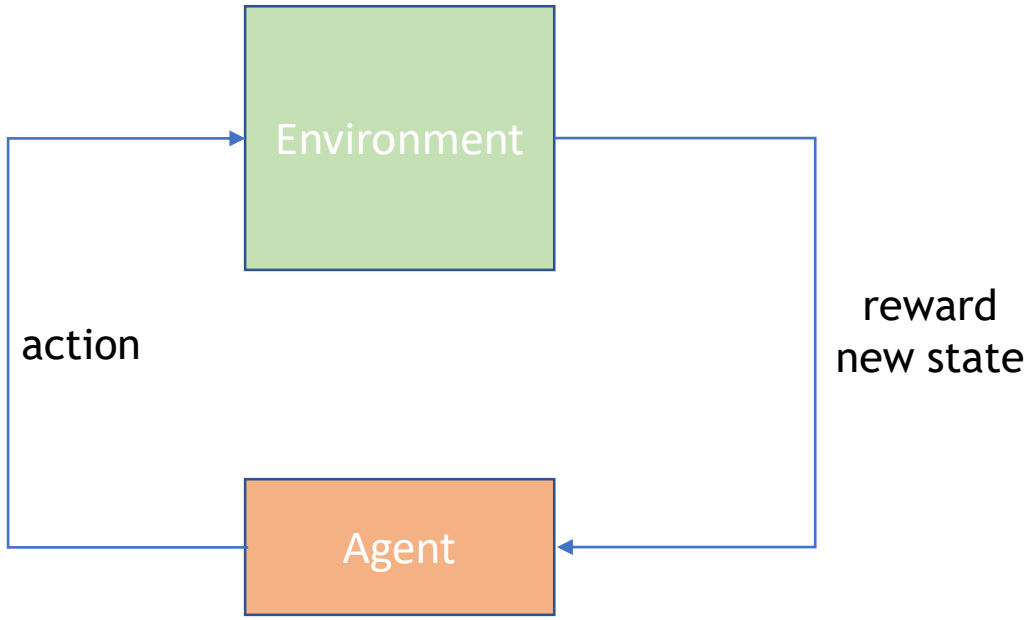
Reward

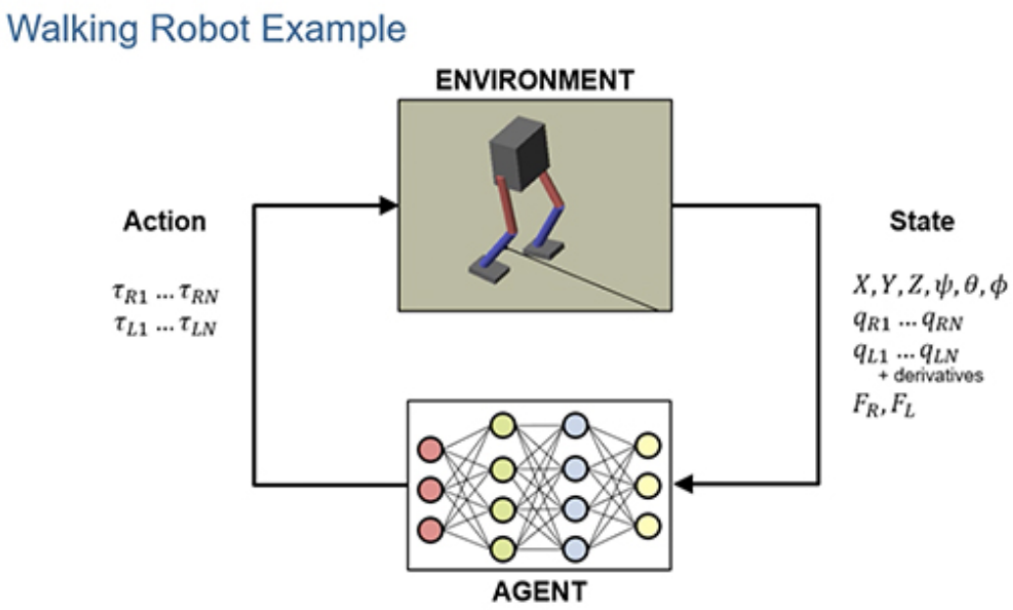
Examples:

Agent experiencing the environment



Reinforcement Learning Components



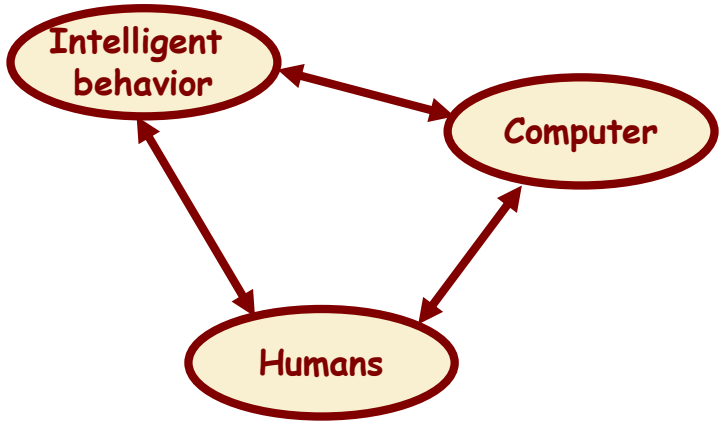


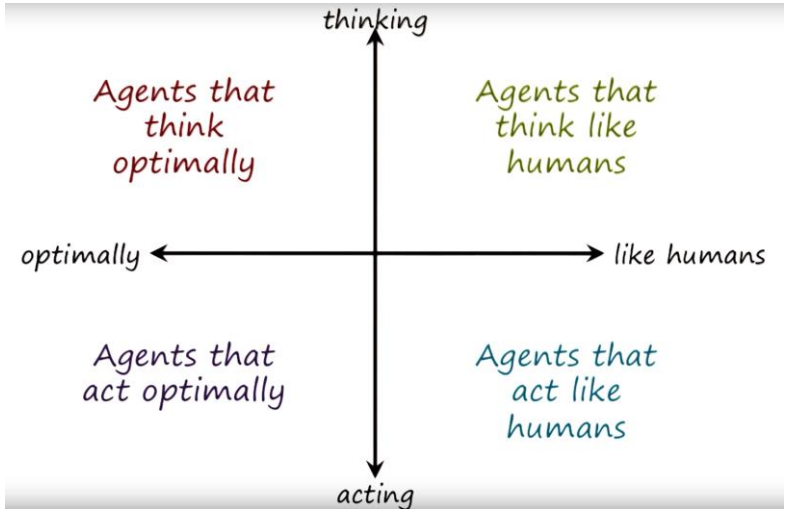
→ There is no supervisor to say what action to take, only a **reward signal**
→ A decision taken now, will reflect after few set of sequences
→ Agent doesn't know about the environment states but its own states

*Taken from Matlab

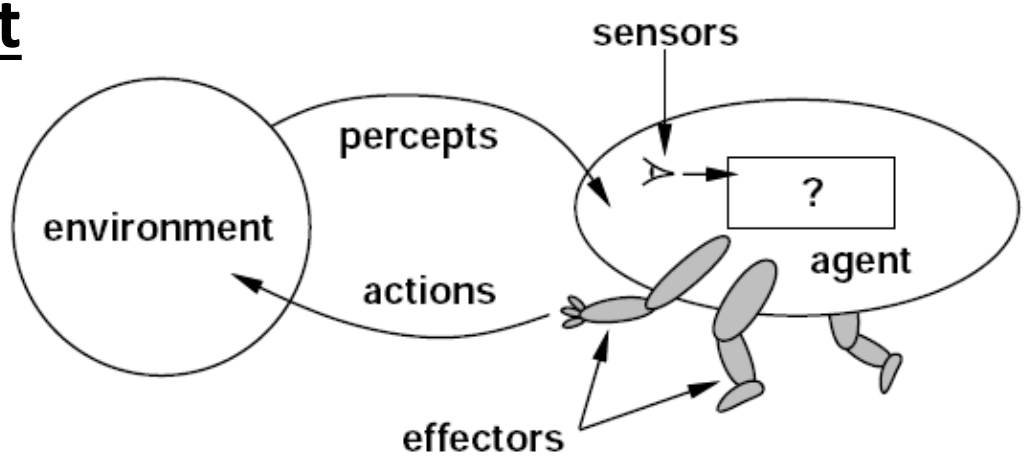
What is AI?

AI is the reproduction of human reasoning and intelligent behavior by computational methods



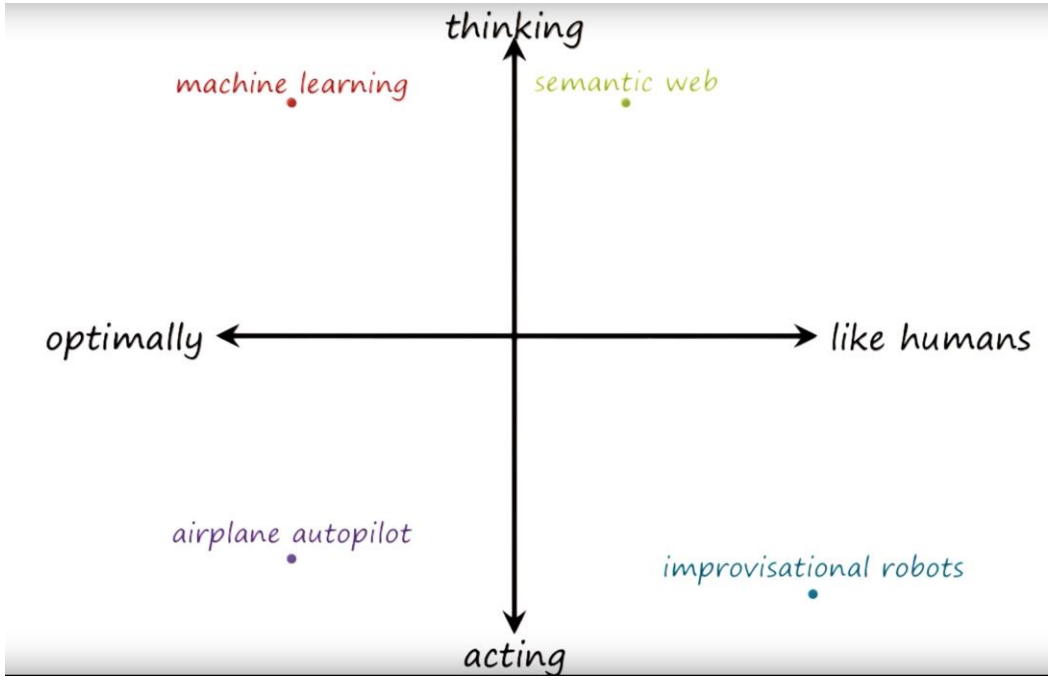


Agent



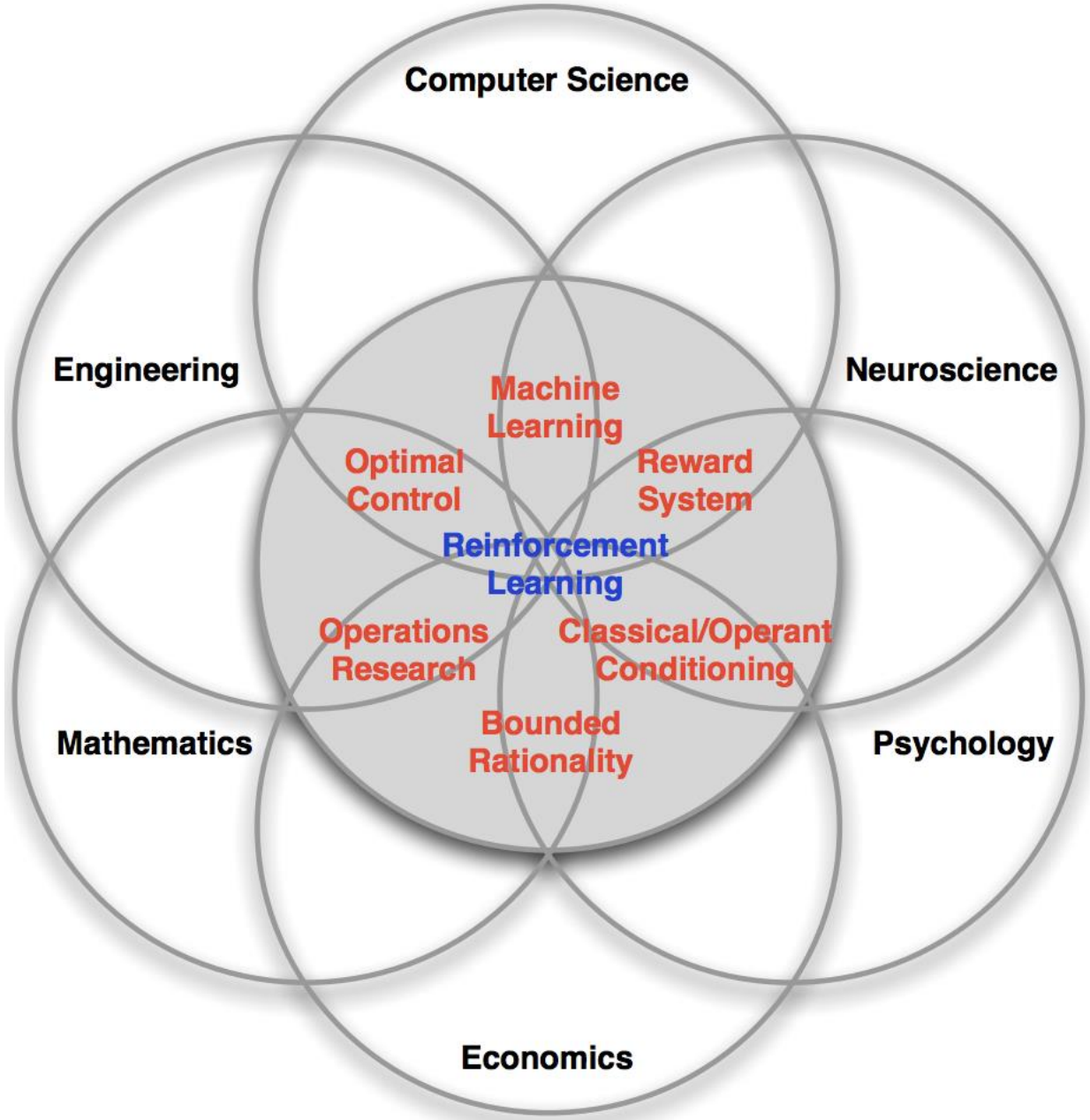
What is AI?

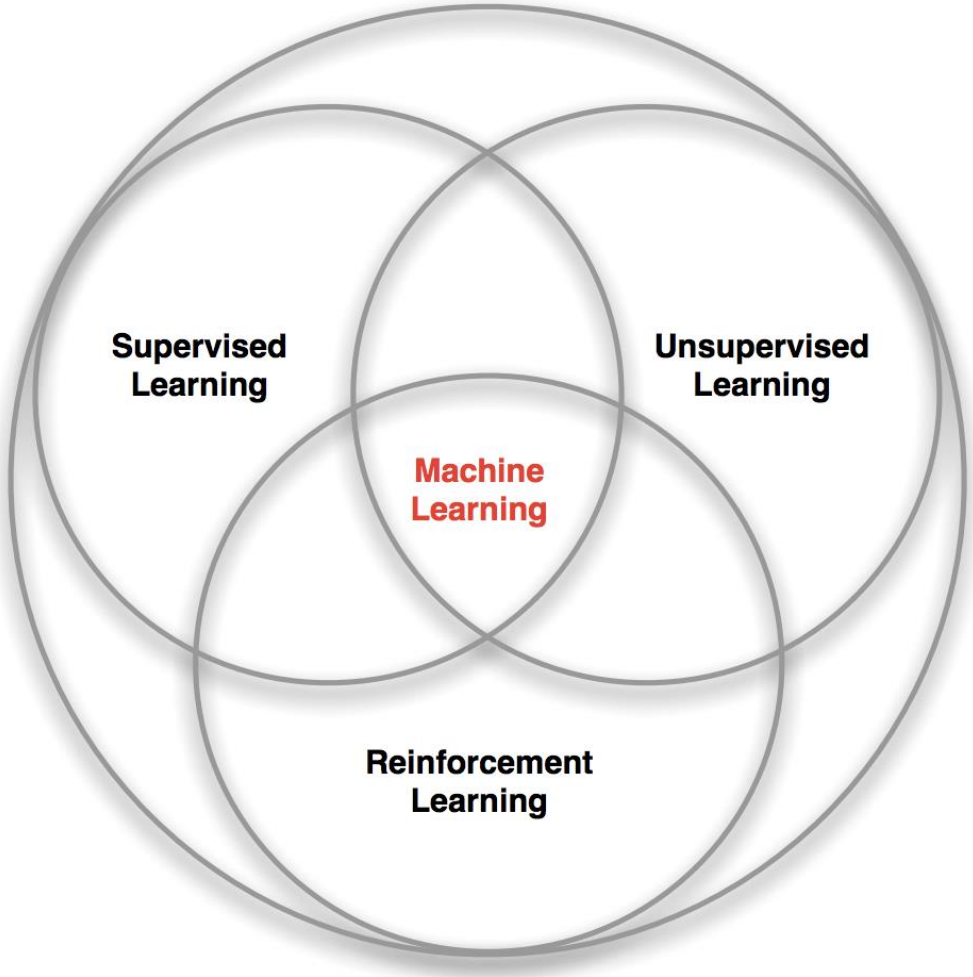
AI is the reproduction of human reasoning and intelligent behavior by computational methods



IBM Watson

Apple Siri
Unmanned Cars
Roomba Vacuum Cleaner
IBM Watson
Deep Blue
Credit Card Fraud Prediction





Taken from David Silver

Characteristics of Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Taken from David Silver

Examples of Rewards

- Fly stunt manoeuvres in a helicopter
  - +ve reward for following desired trajectory
  - −ve reward for crashing
- Defeat the world champion at Backgammon
  - +/−ve reward for winning/losing a game
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score

Taken from David Silver

Learning & Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Taken from David Silver

Exploration & Exploitation

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

Taken from David Silver

Examples: Exploration & Exploitation

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

Taken from David Silver

What is Reinforcement Learning – an example

Environment and Actions

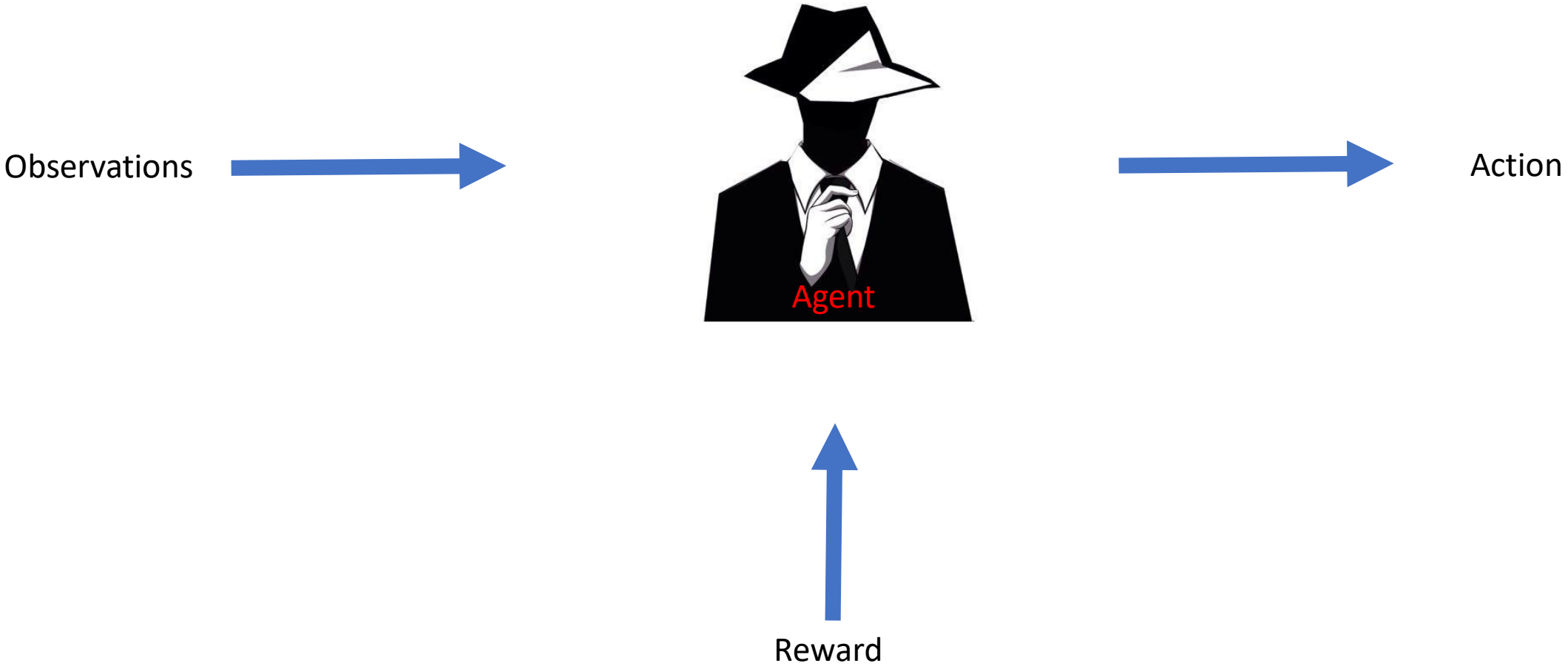
- Fully Observable (Chess) vs Partially Observable (Poker)
- Single Agent (Atari) vs Multi Agent (DeepTraffic)
- Deterministic (Cart Pole) vs Stochastic (DeepTraffic)
- Static (Chess) vs Dynamic (DeepTraffic)
- Discrete (Chess) vs Continuous (Cart Pole)

Taken from David Silver

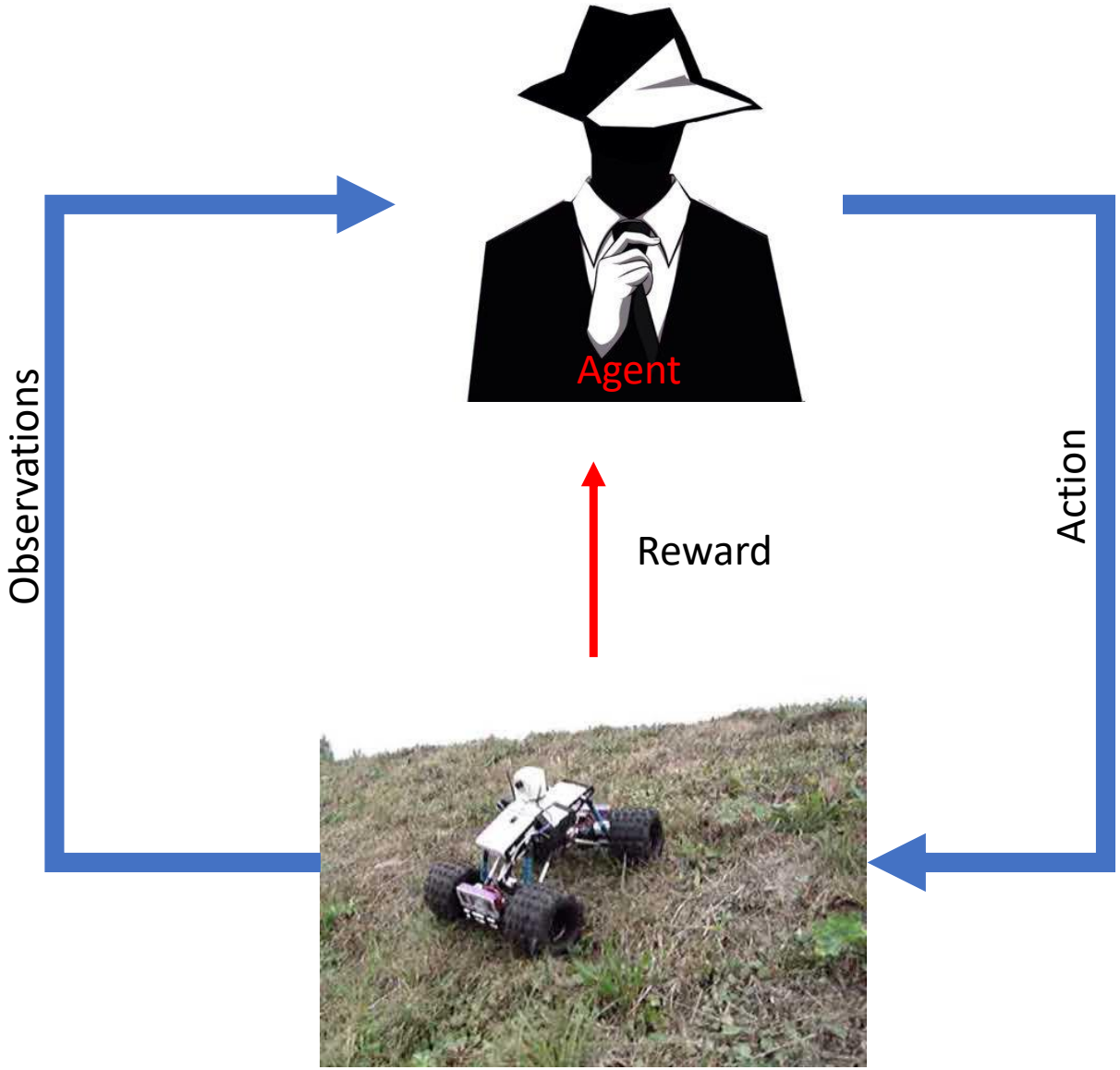
Rewards

- Reward is positive when the agent achieves the objective
- Reward is negative when the agent doesn’t achieve the objective
- Reward is a scalar signal but not a decision factor decides what action to take
- The agent has to maximize the cumulative reward

Agent



Agent



- At each time step the agent executes the action in the environment and thereby receives a reward by moving to the new state

- For example:
  - Current State: S1
  - Action : Move Upwards
  - Next State : S2 (moved upwards)
  - Reward : + 4 (Positive Reward)

- History: A set of previous States, Actions and the corresponding Rewards

- State is the information that captures the complete information of the History, for example, what happens next depends upon the ”State”

- **Markov State**: A state is “Markov” which is independent of the past given the present
- The history of the states is irrelevant when the current state is known
- **Fully Observable Environment (MDP)**
  - Fully observable environment is one in which the agent can always see the entire state of environment. In case of fully observable environments all relevant portions of the environment are observable
  - Example: Chess Game, Chinese Checker Game
- **Partially Observable Environment (POMDP)**
  - Partially observable environment is one in which the agent can never see the entire state of environment. In case of partially observable environments not all relevant portions of the environment are observable
  - Agent has to build the belief system
  - Example: Poker Game

State 1

State 2

In Partial Observable Environment, the robot knows its current location but not the absolute location – agent has to construct its own state environment

Elements of Reinforcement Learning

Policy

- Agent's behavior
- Mapping from State space to Action Space

Value Function

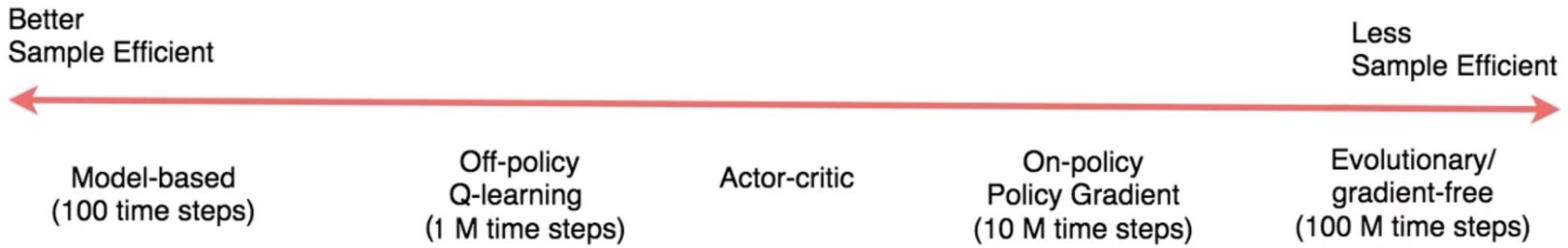
- The future reward that will be received from this state
- Evaluates the state – how good the state is (in the long run)
- It helps to select the actions

Model

- Agent's representation of the environment
- It predicts the next state and the corresponding reward

- Agent's main objective is to discover the good policy
- Agent explores the world to find the more information
- Agent exploits the information to maximize the cumulative reward

3 Types of Reinforcement Learning



Model-based

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

Value-based

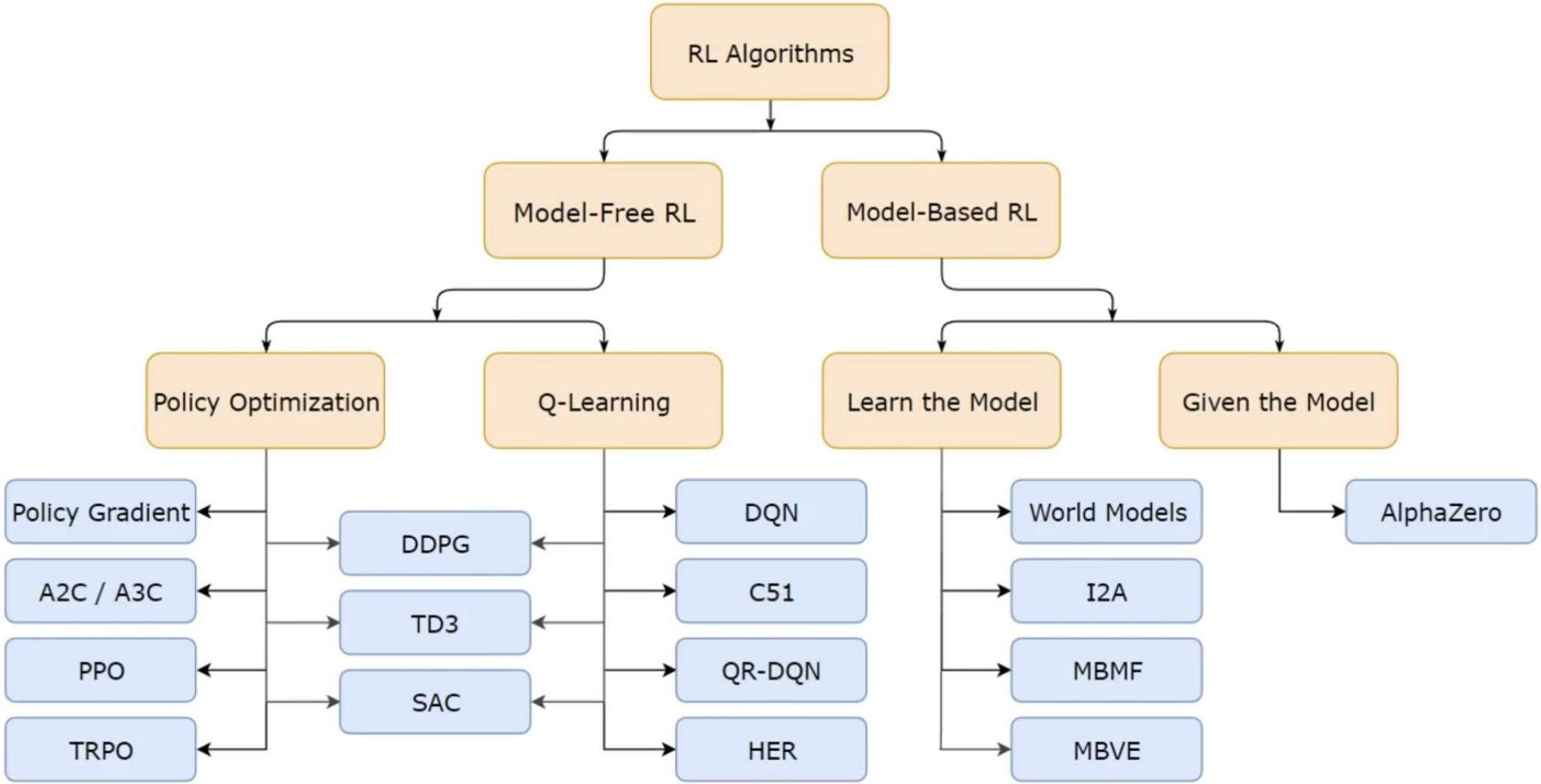
- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Ref:Deeplearning.mit.edu

Taxonomy of RL Methods



Link: https://spinningup.openai.com

RL and Planning

- Learning and Planning:
  - RL :
    - The environment is initially unknown
    - The agent interacts with the environment
    - The agent improves its policy
  - Planning:
    - A model of the environment is known
    - The agent performs computations with its model (without any external interaction)
    - The agent improves its policy a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Model-based and Model-Free Methods

- Model free methods learn directly for experience, this means that they perform actions either in the real world (ex: robots)or in computer (ex: games). Then they collect the reward from the environment, whether positive or negative, and they update their value functions.
- This is a **key difference** with Model-Based approach. Model-Free methods act in the real environment in order to learn.

Finite Markov Decision Processes

Markov Property

"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{matrix} \\ \textit{from} \end{matrix} \begin{matrix} \textit{to} \\ \end{matrix} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

where each row of the matrix sums to 1.

Taken from David Silver, Berkeley

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$

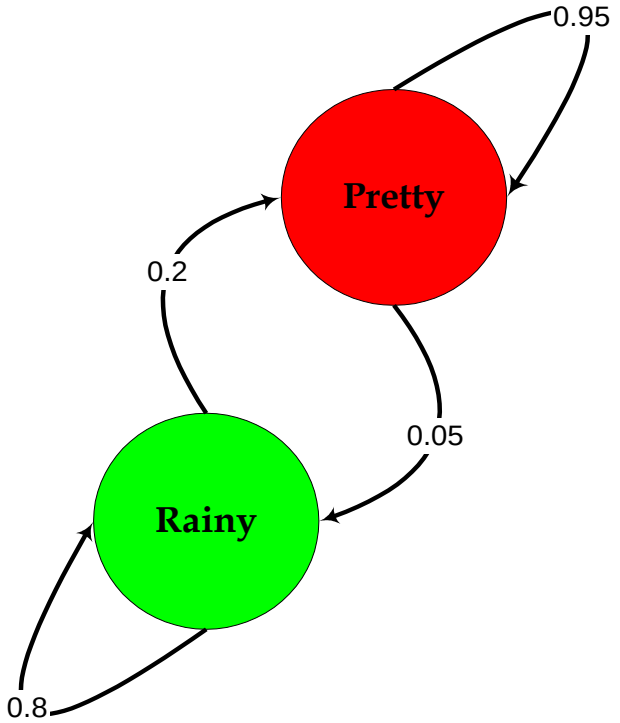
Markov Model – Working Example

- The weather in Delhi of past 26 days

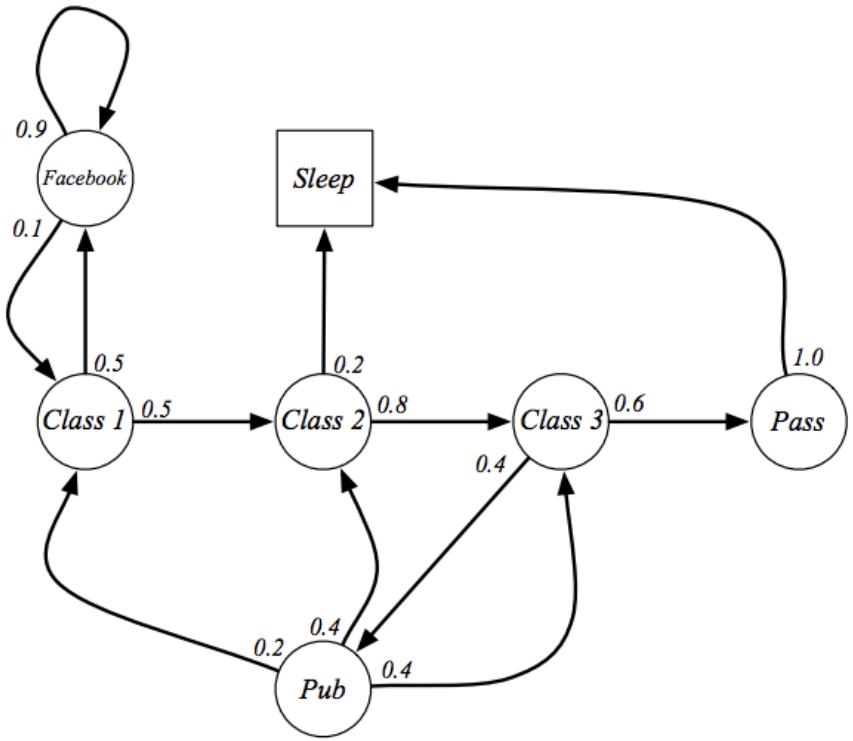
  STATES = *{ pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, rainy, rainy, rainy, rainy, rainy, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty, pretty }*

	Tomorrow's weather		
Today's Weather		Pretty	Rainy
	Pretty	0.95	0.05
	Rainy	0.2	0.8

$$\textbf{TRANS} = \begin{bmatrix} 0.95 & 0.05 \\ 0.2 & 0.8 \end{bmatrix}$$



Let us go back to Assistant Professor Salary example:



Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

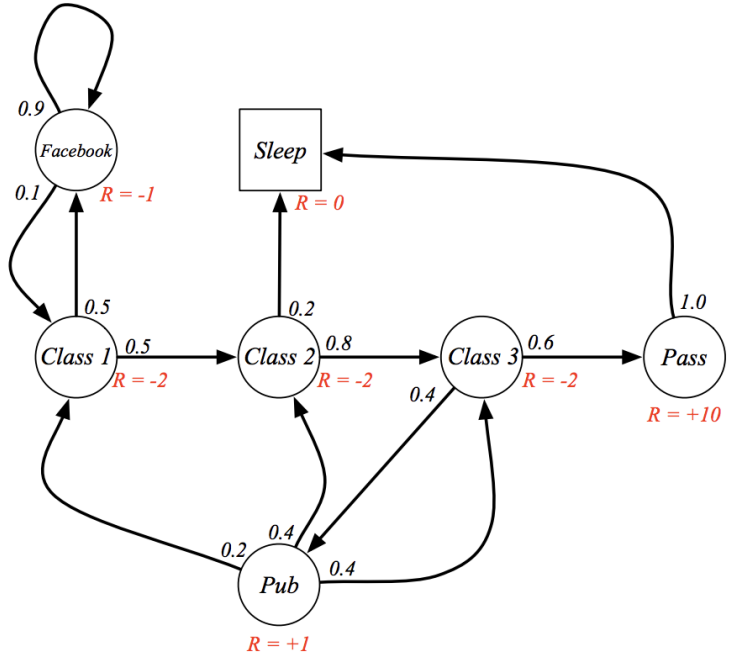
Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$



Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

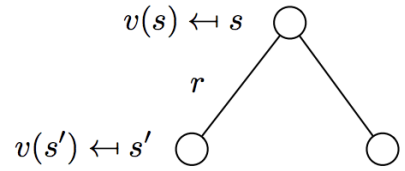
Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

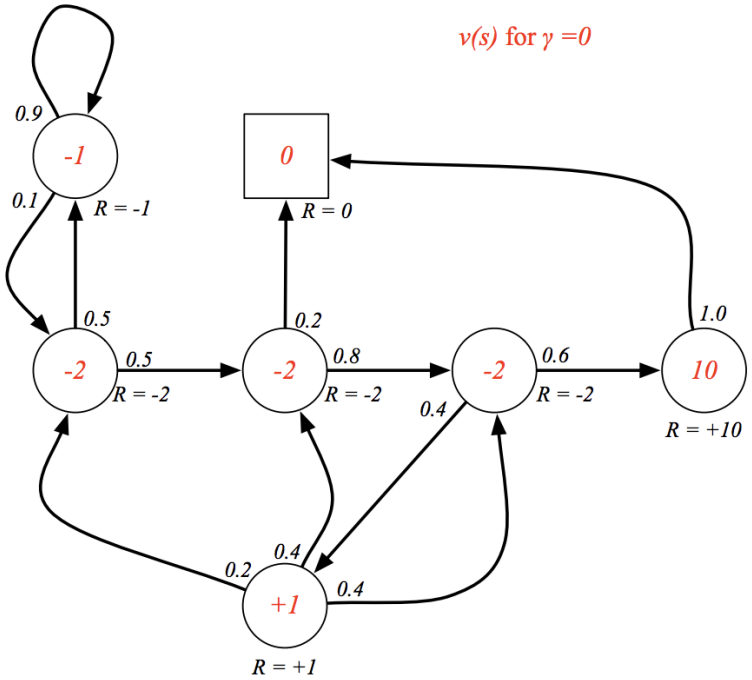
Bellman Equation

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

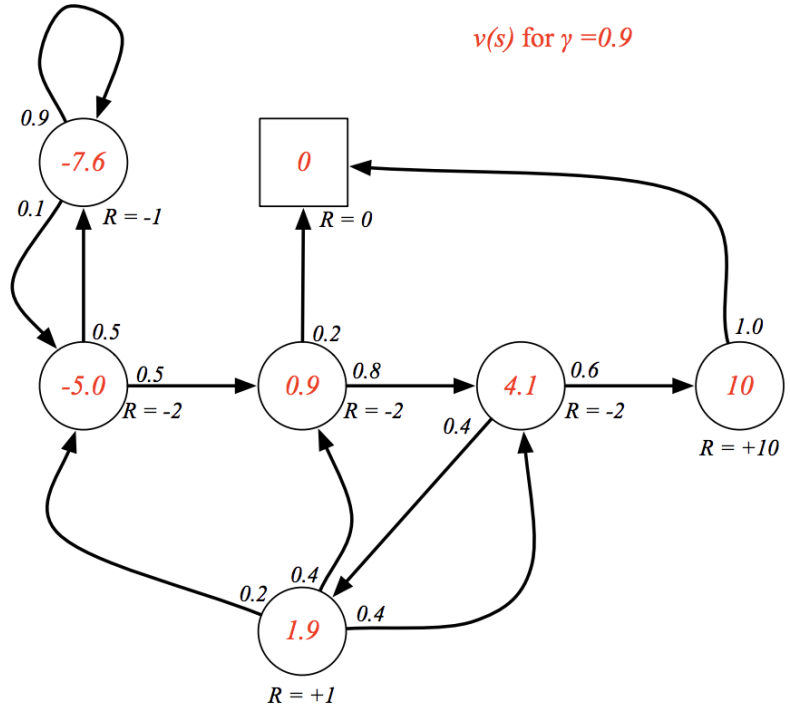


$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

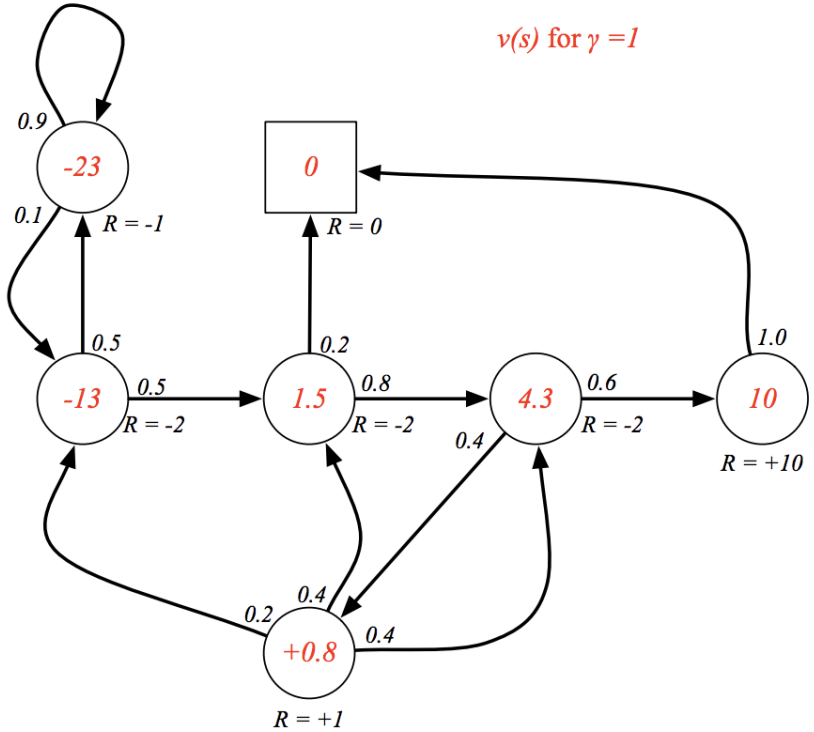
v(s) for γ =0



v(s) for γ =0.9



v(s) for γ =1



The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

- The Bellman equation is a linear equation
- It can be solved directly:

$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P})\, v &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

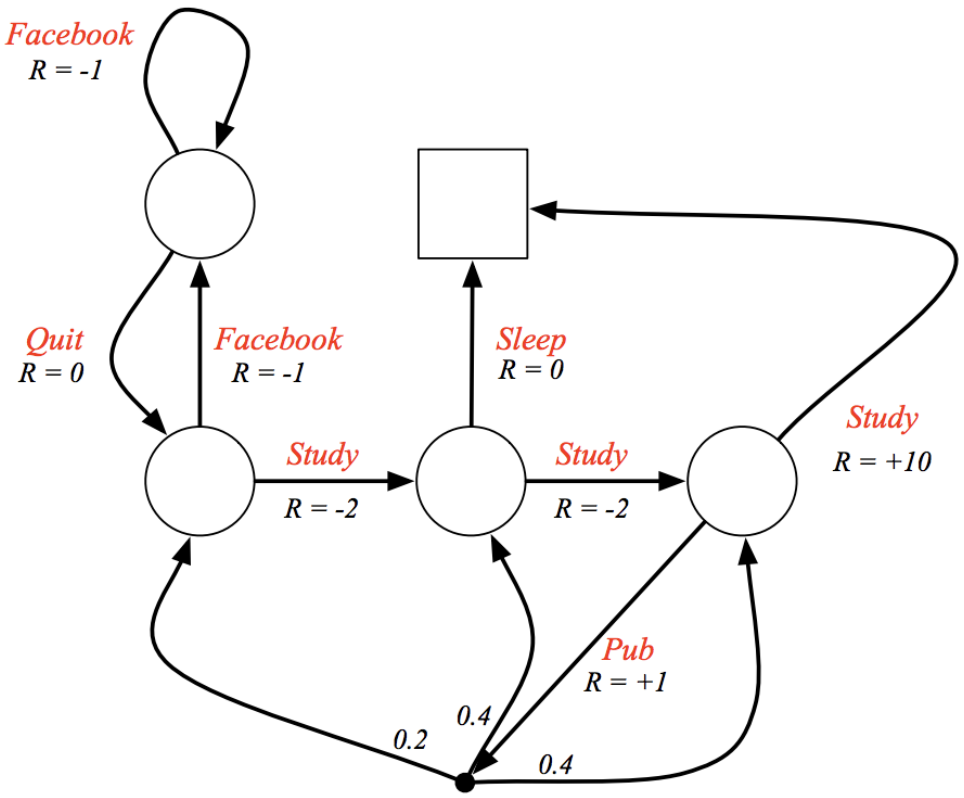
This is Markov Reward Process, what is Markov Decision Process ?

Markov Decision Process

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- where

$$\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$



Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P} \left[A_t = a \mid S_t = s \right]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Definition

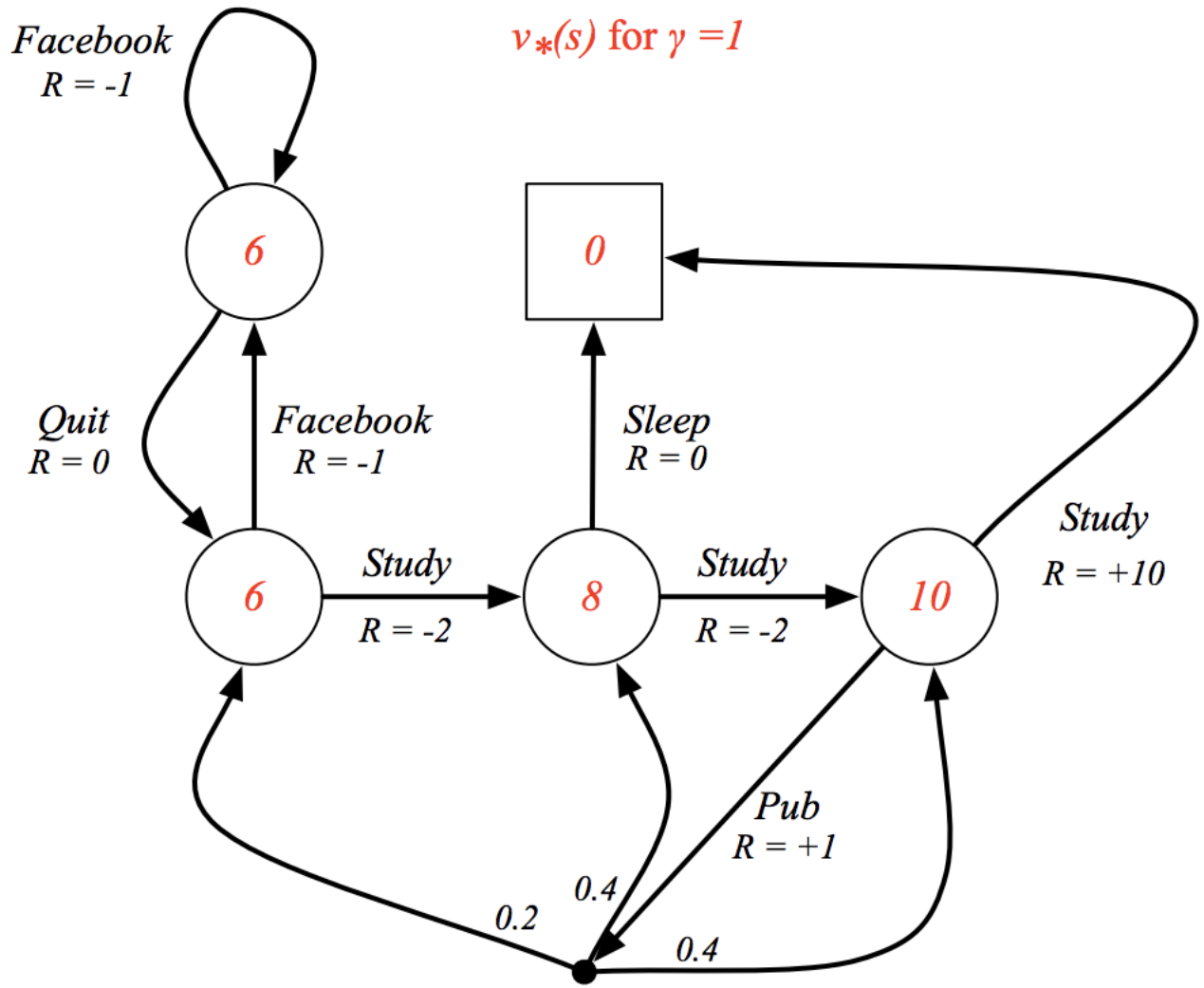
The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

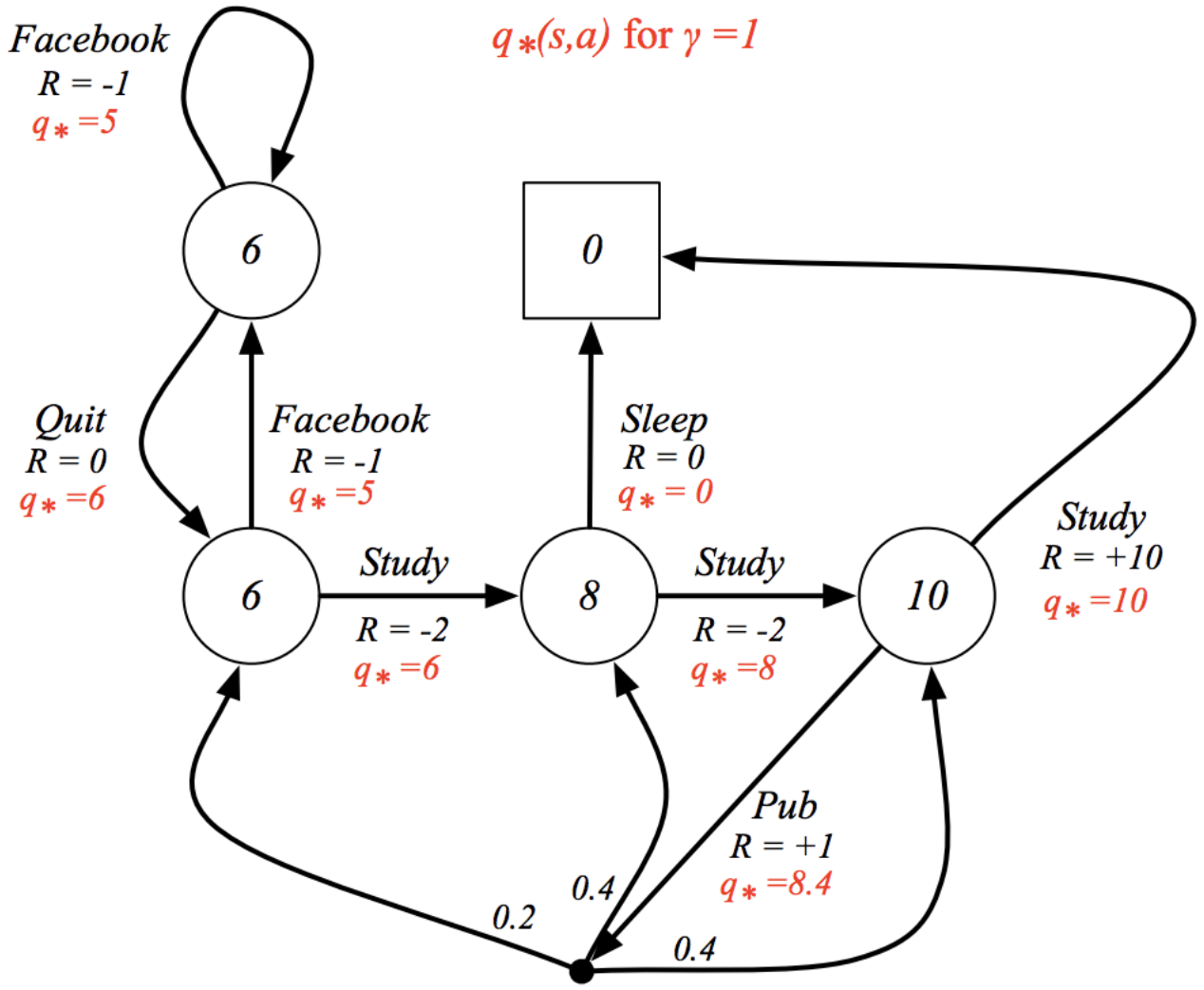
The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.


$v_*(s)$ for $\gamma = 1$
Facebook
R = -1
6
0
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
Study
R = +10
6
Study
R = -2
8
Study
R = -2
10
Pub
R = +1
0.2
0.4
0.4

Optimal Action – Value function



Optimal Policy

Define a partial ordering over policies

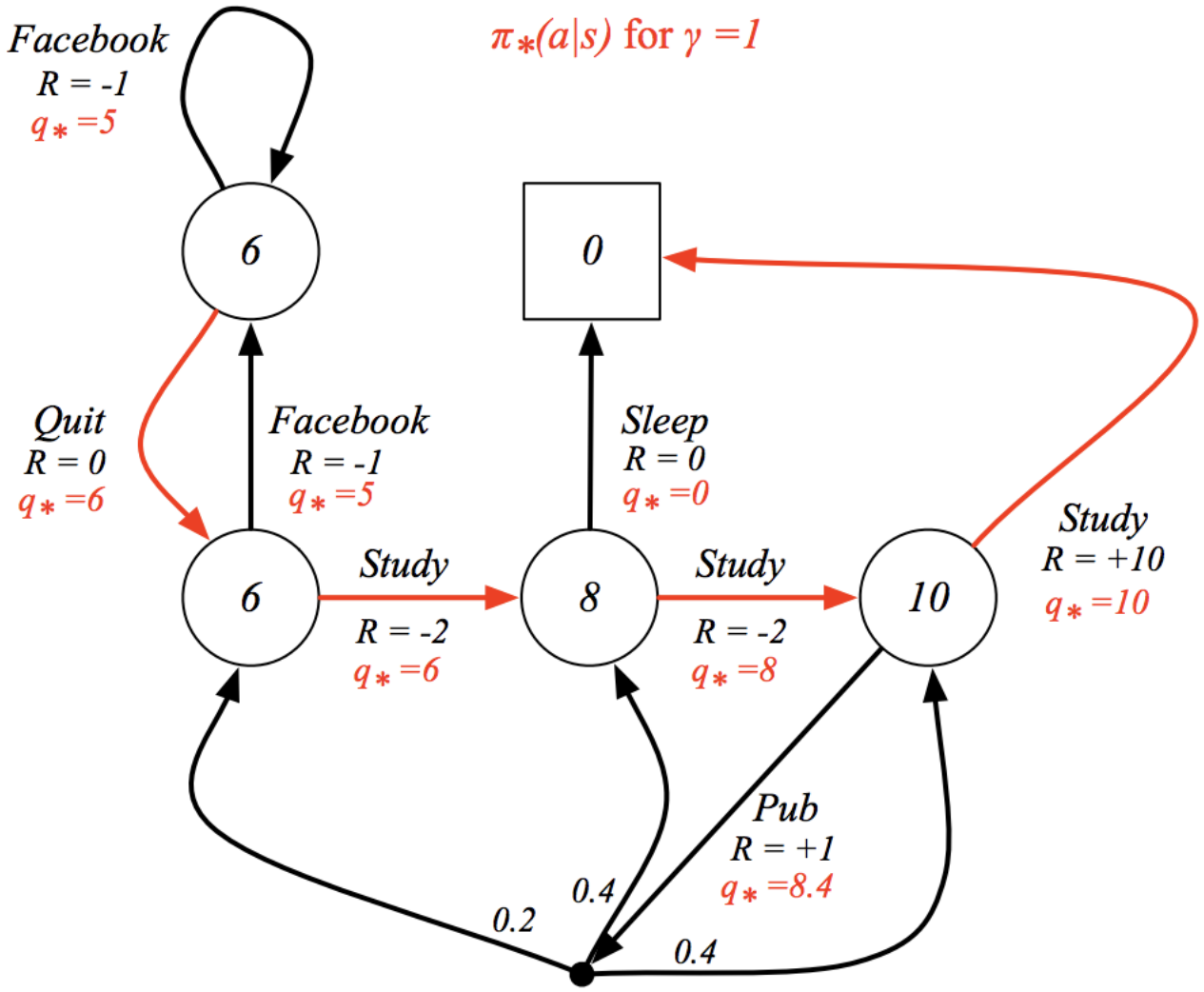
$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

Optimal Policy



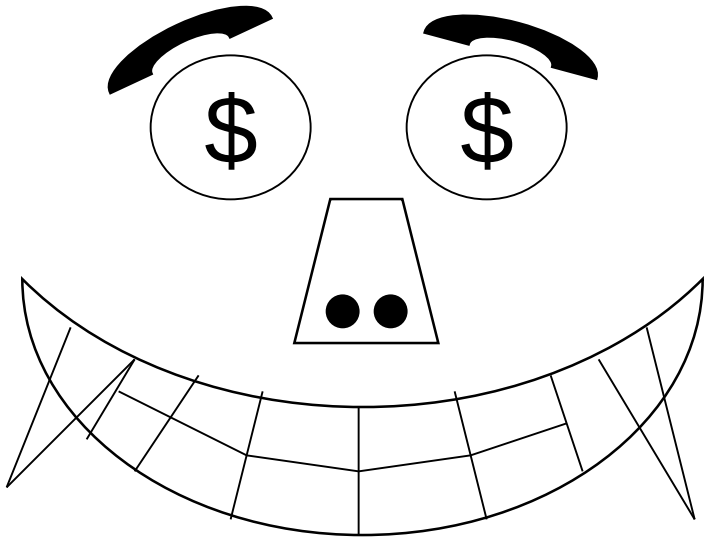
- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Discounted Rewards – why ?

An assistant professor gets paid, say, 20K per year.

How much, in total, will the A.P. earn in their life?

20 + 20 + 20 + 20 + 20 + … = Infinity



What's wrong with this argument?

Taken from Andrew W. Moore, http://www.cs.cmu.edu/~awm/tutorials

Discounted Rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

- Because of chance of obliteration
- Because of inflation

Example:

Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.

Assuming payment *n* years in future is worth only $(0.9)^n$ of payment now, what is the AP's Future Discounted Sum of Rewards ?

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor γ" is

(reward now) +

γ (reward in 1 time step) +

γ^2 (reward in 2 time steps) +

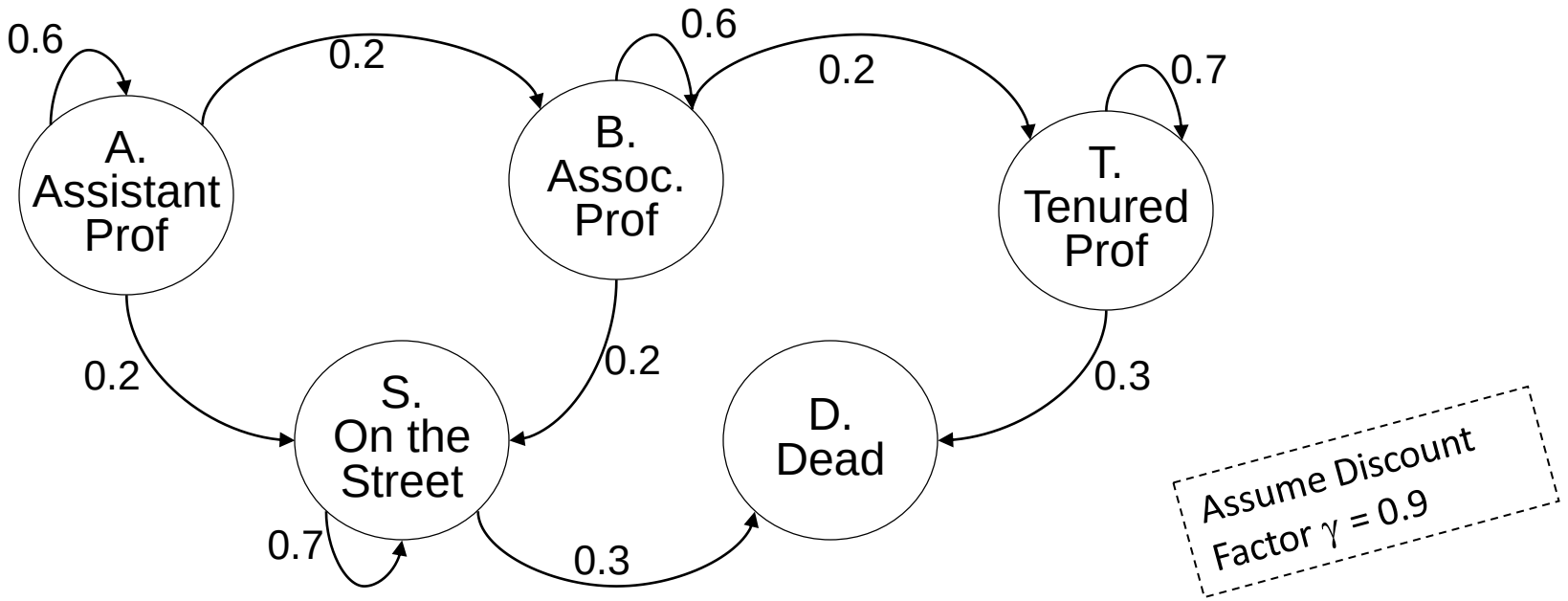
γ^3 (reward in 3 time steps) +

:

: (infinite sum)

Taken from Andrew W. Moore, http://www.cs.cmu.edu/~awm/tutorials

The Academic Life



Define:

J_A = Expected discounted future rewards starting in state A

J_B = Expected discounted future rewards starting in state B

J_T = " " " " " " " T

J_S = " " " " " " " S

J_D = " " " " " " " D

How do we compute J_A, J_B, J_T, J_S, J_D ?

Taken from Andrew W. Moore, http://www.cs.cmu.edu/~awm/tutorials

A Markov System with Rewards…

- Has a set of states $\{S_1\ S_2 \cdot\cdot S_N\}$
- Has a transition probability matrix

$$P = \begin{pmatrix} P_{11}\ P_{12} \cdot\cdot P_{1N} \\ P_{21} \\ : \\ P_{N1} \quad \cdot\cdot \quad P_{NN} \end{pmatrix} \qquad P_{ij} = \text{Prob}(\text{Next} = S_j \mid \text{This} = S_i)$$

- Each state has a reward. $\{r_1\ r_2 \cdot\cdot r_N\}$
- There's a discount factor γ . $0 < \gamma < 1$

On Each Time Step …

0. Assume your state is S_i
1. You get given reward r_i
2. You randomly move to another state
   $P(\text{NextState} = S_j \mid \text{This} = S_i) = P_{ij}$
3. All future rewards are discounted by γ

Taken from Andrew W. Moore, http://www.cs.cmu.edu/~awm/tutorials

Solving a Markov System

Write $J^*(S_i)$ = expected discounted sum of future rewards starting in state S_i

$J^*(S_i) = r_i + \gamma$ x (Expected future rewards starting from your next state)

$= r_i + \gamma(P_{i1}J^*(S_1)+P_{i2}J^*(S_2)+ \cdots P_{iN}J^*(S_N))$

Using vector notation write

$$\underline{J}= \begin{pmatrix} J^*(S_1) \\ J^*(S_2) \\ : \\ J^*(S_N) \end{pmatrix} \qquad \underline{R}= \begin{pmatrix} r_1 \\ r_2 \\ : \\ r_N \end{pmatrix} \qquad \underline{P}= \begin{pmatrix} P_{11} & P_{12} & \cdot\cdot & P_{1N} \\ P_{21} & \ddots & & \\ : & & & \\ P_{N1} & P_{N2} & \cdot\cdot & P_{NN} \end{pmatrix}$$

Question: can you invent a closed form expression for $\underline{J}$ in terms of $\underline{R}$ $\underline{P}$ and γ ?

Solving a Markov System with Matrix Inversion

- Upside: You get an exact answer

- Downside:

Solving a Markov System with Matrix Inversion

- Upside: You get an exact answer

- Downside: If you have 100,000 states you're solving a 100,000 by 100,000 system of equations.

Value Iteration: another way to solve a Markov System

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

$J^1(S_i)$ = (what?)

$J^2(S_i)$ = (what?)

:

$J^{k+1}(S_i)$ = (what?)

Value Iteration: another way to solve a Markov System

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

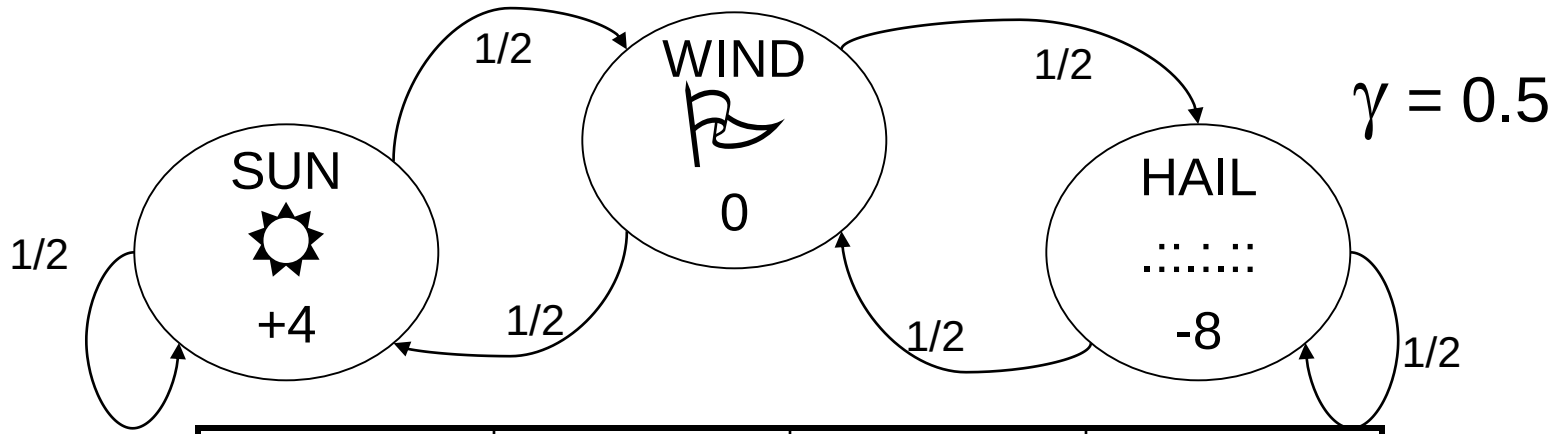
:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

$J^1(S_i) = r_i$	(what?)
$J^2(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^1(s_j)$	(what?)
:	
$J^{k+1}(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^k(s_j)$	(what?)

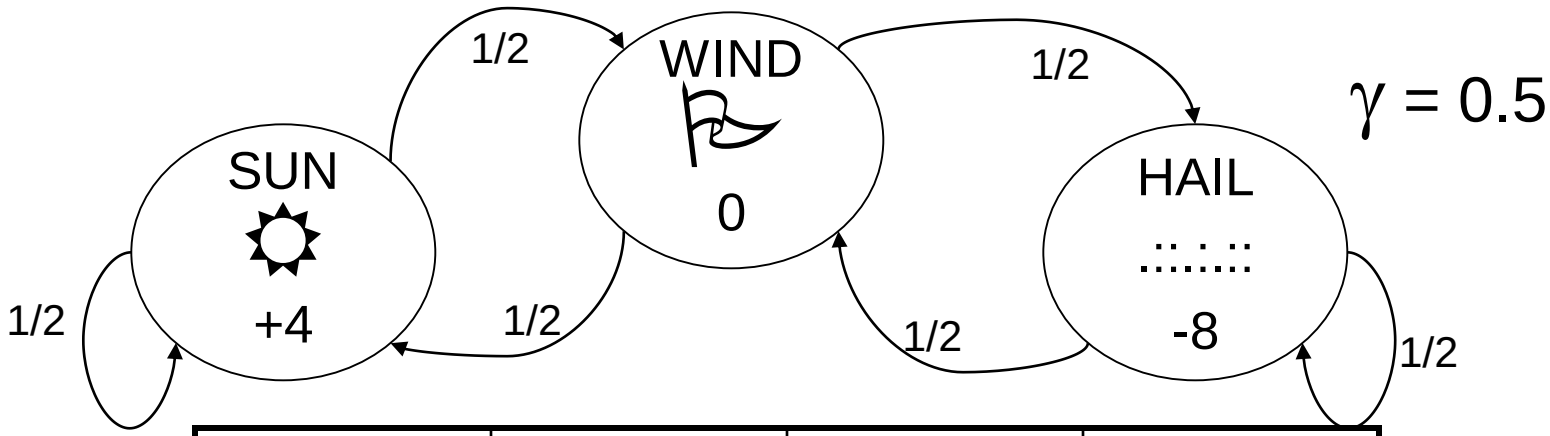
N = Number of states

Let’s do Value Iteration (do it yourself please …)



k	J^k(SUN)	J^k(WIND)	J^k(HAIL)
1			
2			
3			
4			
5			

Let’s do Value Iteration



k	J^k(SUN)	J^k(WIND)	J^k(HAIL)
1	4	0	-8
2	5	-1	-10
3	5	-1.25	-10.75
4	4.94	-1.44	-11
5	4.88	-1.52	-11.11

Value learning

Value function *V(s)* measures the expected discounted rewards for a state under a policy. Intuitively, it measures the total rewards that you get from a particular state following a specific policy.

$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[\, \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t \right]$$

Q_π(s, a=move left 1cm) > Q_π(s, a=move right 1cm)

Which Q function has the highest value ?

Can we use the value learning concept without a model?

Action-value function *Q(s, a)* measures the expected discounted rewards of taking an action.

Q_π(s, a=move left 1cm) > Q_π(s, a=move right 1cm)

Deep Reinforcement Learning: AI = RL + DL

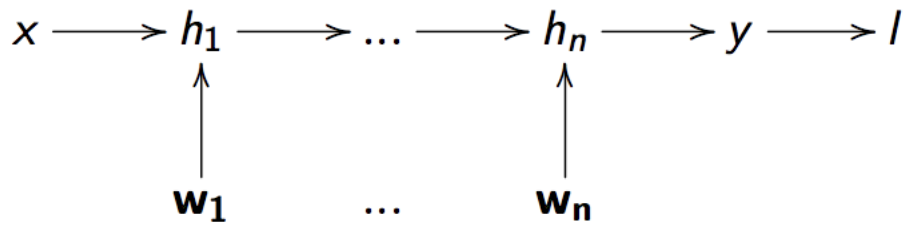
We seek a single agent which can solve any human-level task

- RL defines the objective
- DL gives the mechanism
- RL + DL = general intelligence

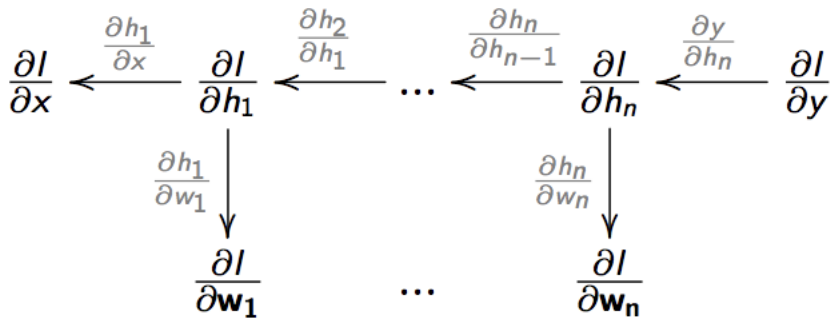
Taken from David Silver

Deep Representations

- A deep representation is a composition of many functions



- Its gradient can be backpropagated by the chain rule



Taken from David Silver

Deep Neural Network

A deep neural network is typically composed of:

- Linear transformations

$$h_{k+1} = Wh_k$$

- Non-linear activation functions

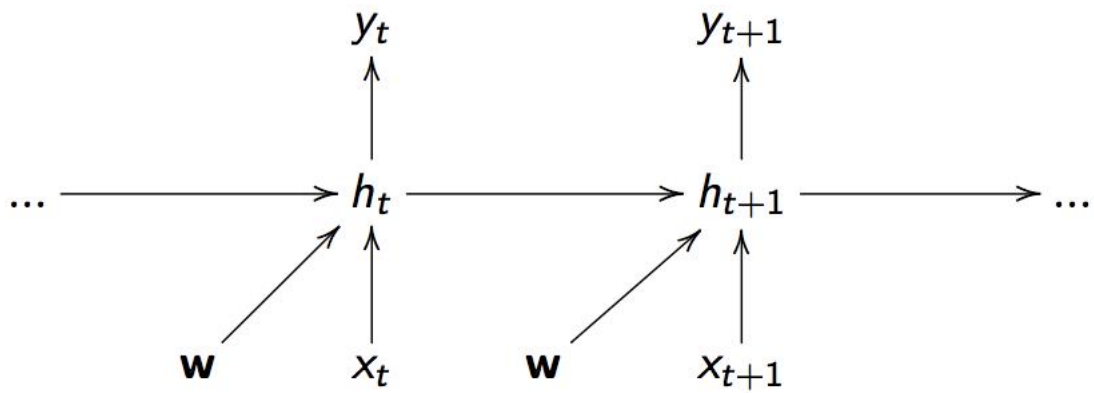
$$h_{k+2} = f(h_{k+1})$$

- A loss function on the output, e.g.
  - Mean-squared error $l = ||y^* - y||^2$
  - Log likelihood $l = \log \mathbb{P}[y^*]$

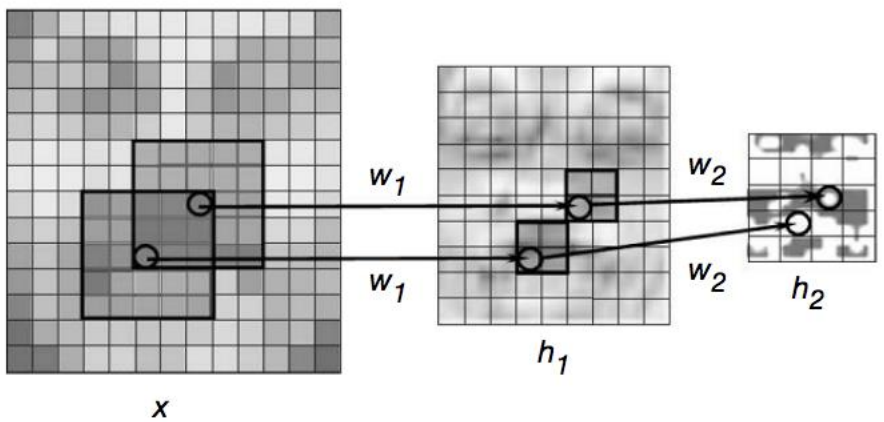
Taken from David Silver

Weight Sharing

Recurrent neural network shares weights between time-steps



Convolutional neural network shares weights between local regions



Taken from David Silver

- A policy is the agent's behaviour
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under
  - with discount factor γ

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

Taken from David Silver

Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

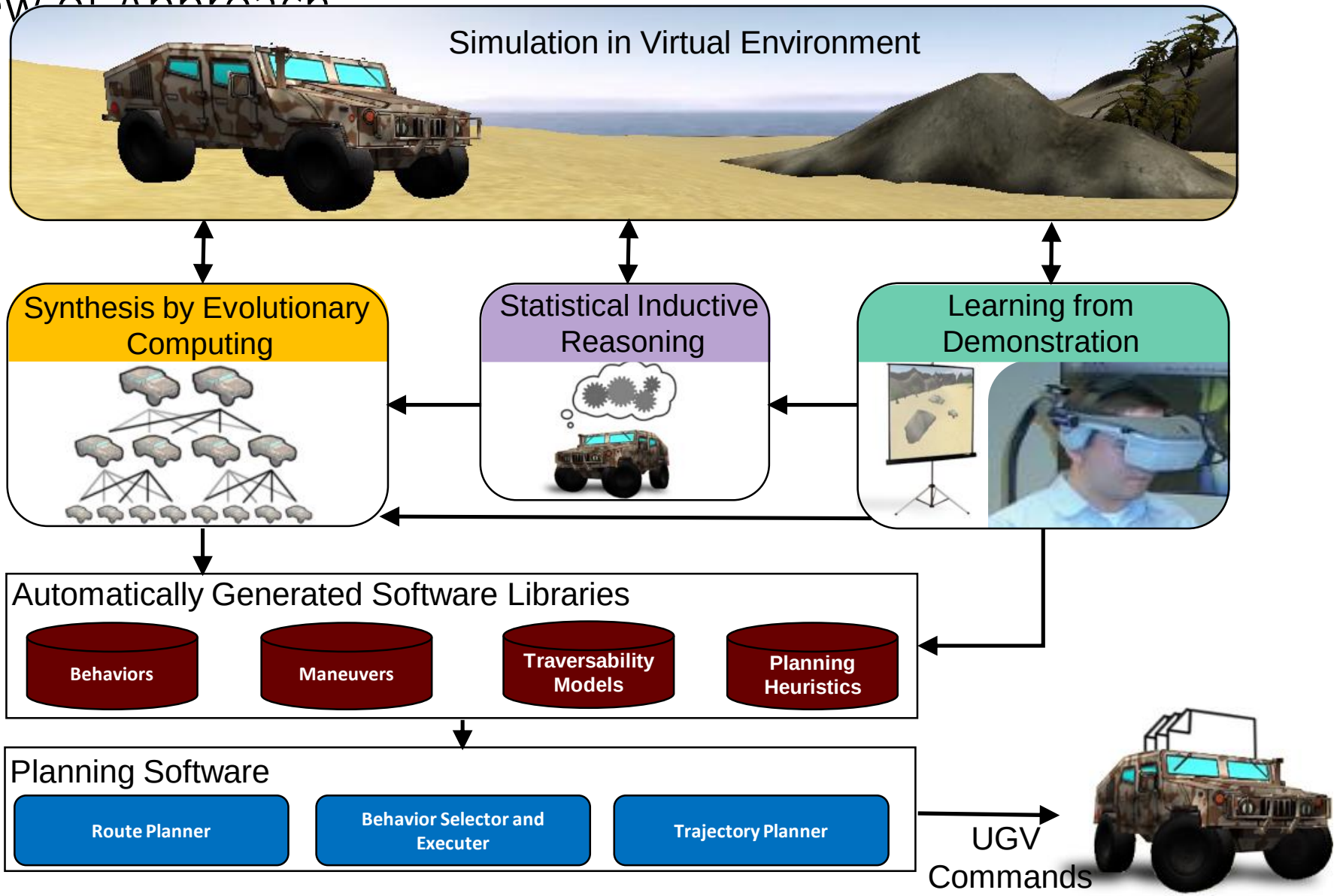
Taken from David Silver

An example

Objectives

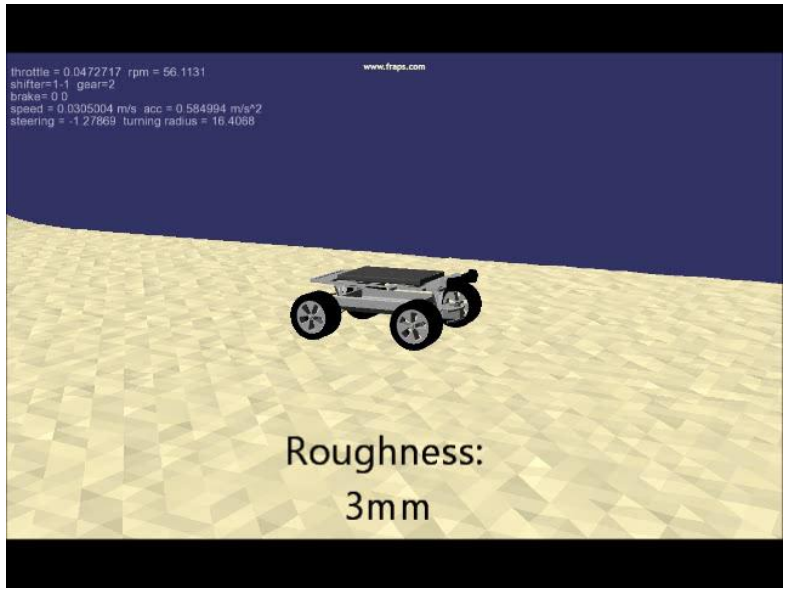
- Develop a computational framework and software for automated generation of behaviors and maneuvers using simulations for autonomous UGVs supporting small units conducting ECO/EMO
  - Behaviors are automatically generated and verified through simulations
  - Symbolic representation facilitates human verification
  - Software can be automatically updated for changes in vehicle designs
  - Software can take different terrain properties into account

Overview of Approach



Simulation Environment

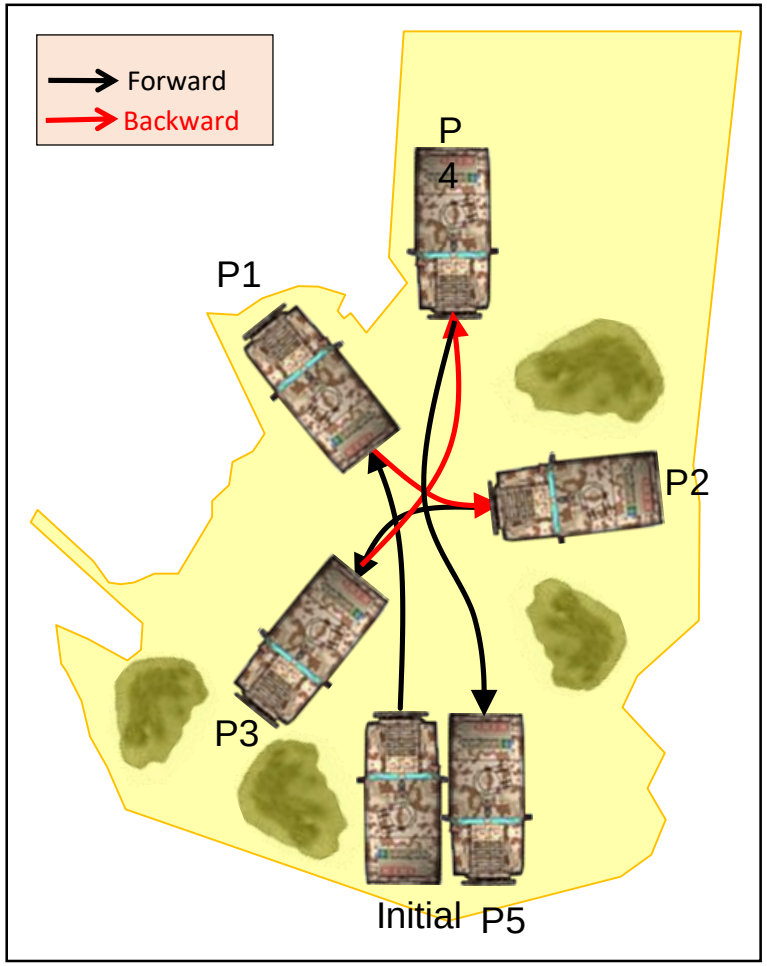
Unreal/USARSim Simulation Environment



Vortex Simulation Environment

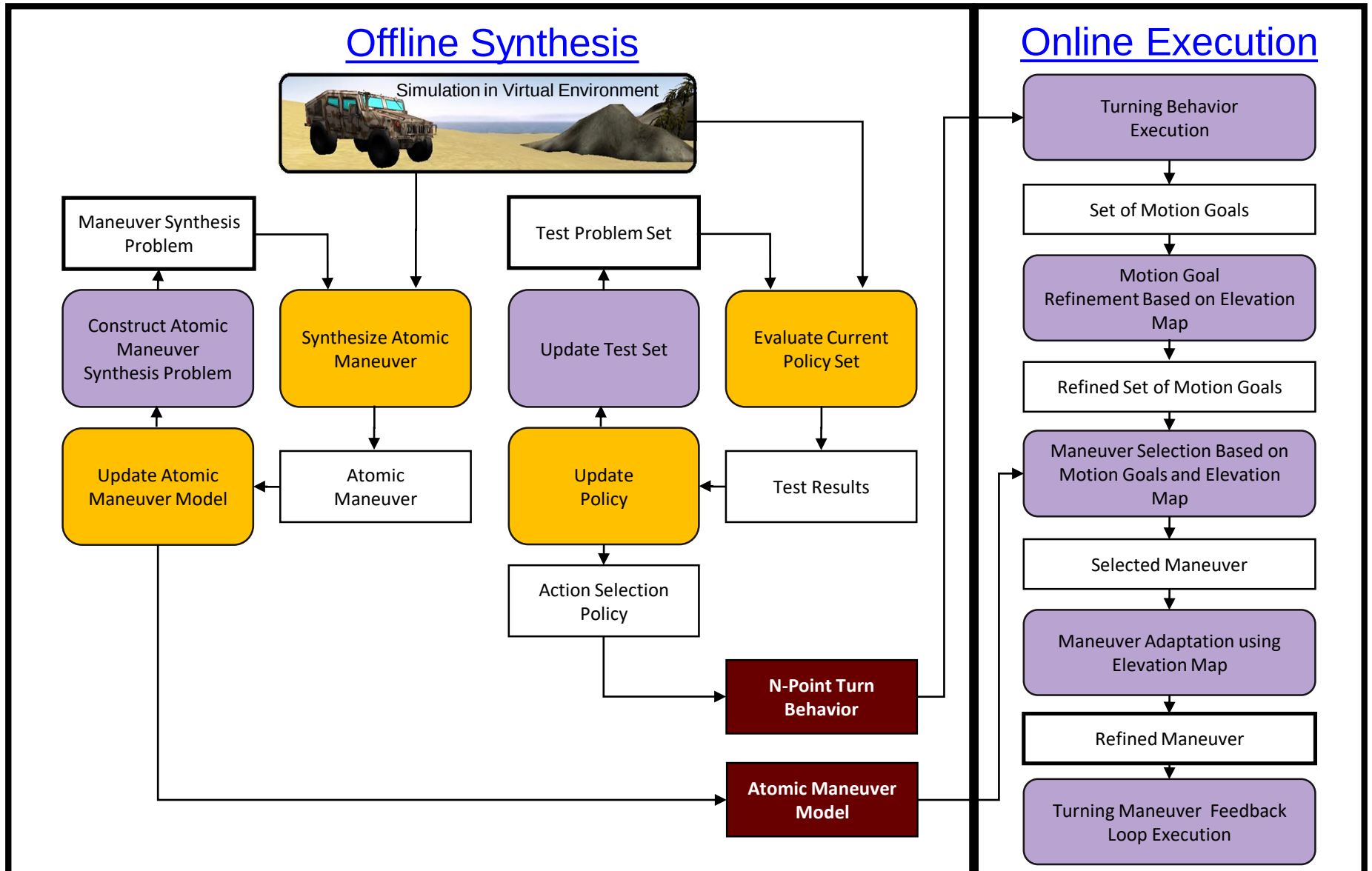
N-Point Turn in Constrained Space Behavior

- Generate N-point turn maneuver between two arbitrary states and $x_I = [x\ y\ \theta\ v]^T$ $x_G = [x\ y\ \theta\ v]^T$ that satisfies the dynamics of the vehicle $f_M(x,u,t)$ and minimizes the execution time
- Generated maneuver should reliably be executed on a rugged, highly constrained terrain



5-point turn in a cluttered environment

Approach for Generating N-Point Turn Behavior



Results

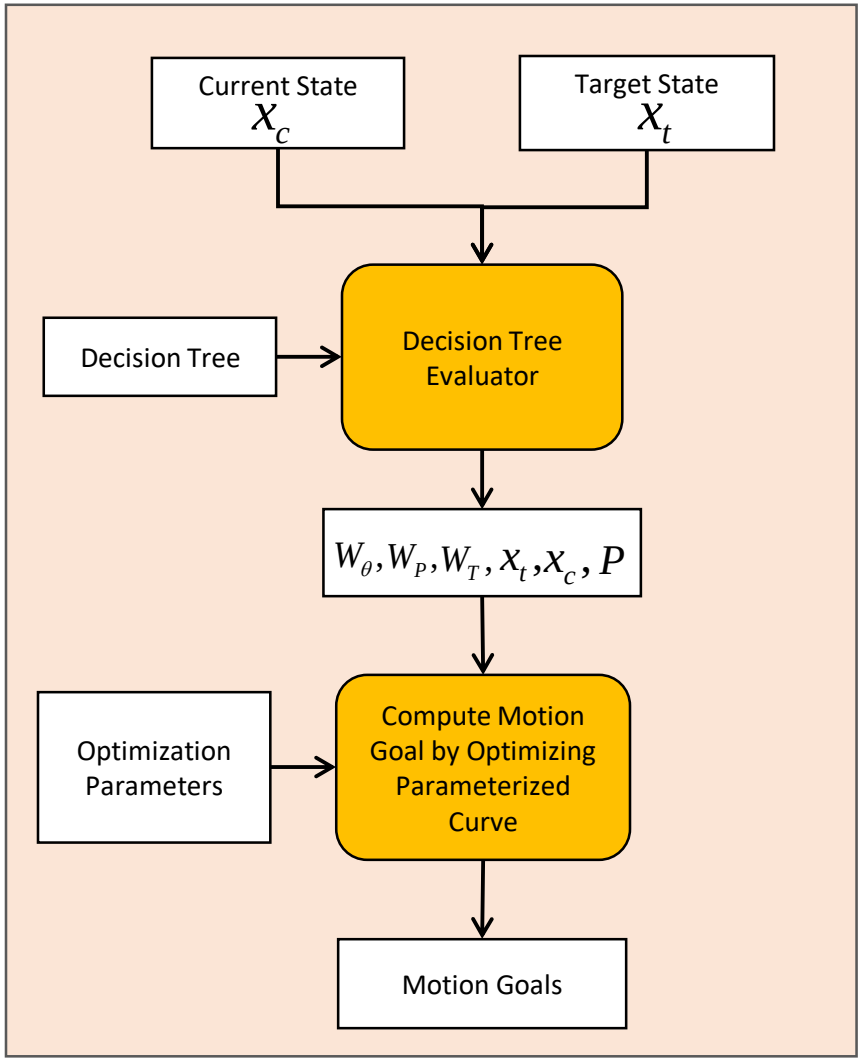
- Policy is represented as a decision tree
- Inputs:
  - Distance to target pose
  - Geometric configuration of obstacles
  - Maximum allowable clearances around the current and target poses
- Outputs:
  - Position error weight W_P
  - Orientation error weight W_θ
  - Terrain error weight W_T
  - Parameterized curve P
- It takes less that 16 second to compute motion goals for 13 point turns

$$C = W_\theta C_\theta + W_P C_p + W_T C_T$$

$$C_\theta = \frac{-dot(\vec{v}.\vec{v}')}{2} + \frac{1}{2} \longrightarrow \text{Orientation}$$

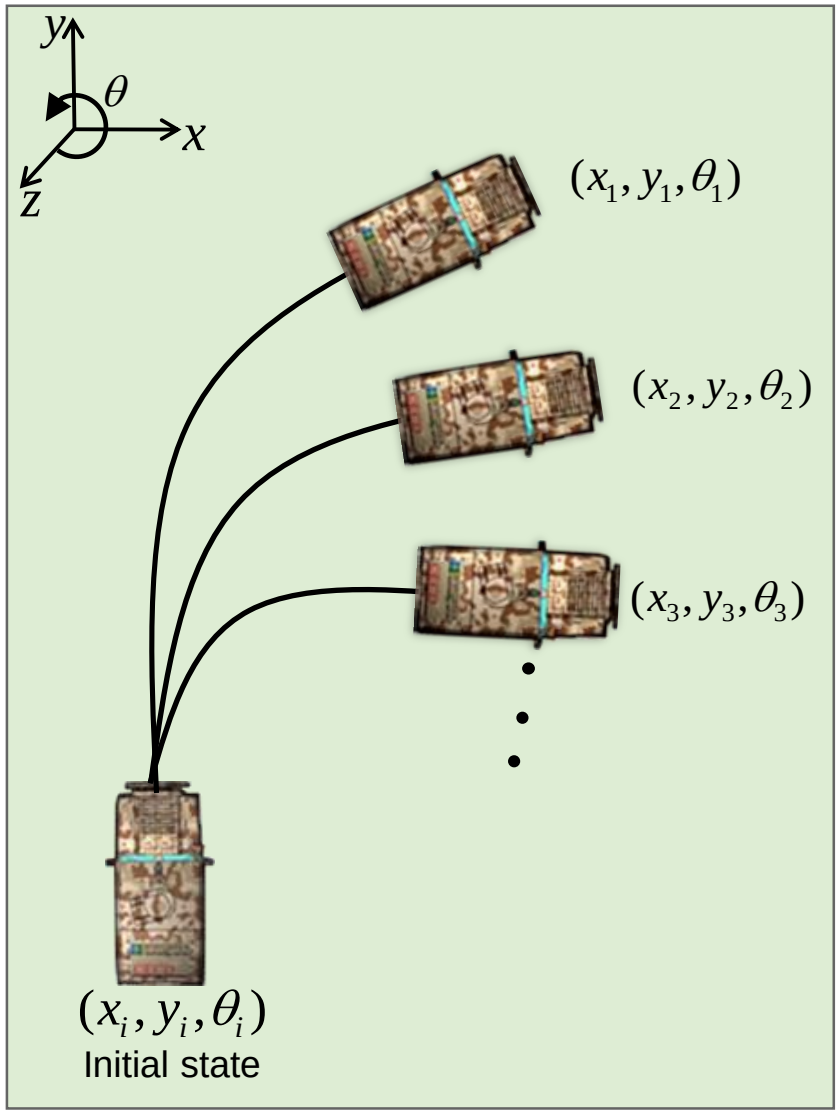
$$C_P = \|x_t - x_c\| \longrightarrow \text{Position}$$

$$C_T = \begin{cases} 1 & \text{if collision} \\ 0 & \text{otherwise} \end{cases} \longrightarrow \text{Collision}$$



Results (Cont.)

- Automatically synthesized optimal atomic maneuvers using a combination hybrid search (evolutionary and local search) of adaption of nearest neighbor
  - Represented using 5 point B-splines
    - 7 point representation does not improve accuracy
  - Output
    - Force Profile (throttle and brake)
    - Steering Angle Profile
  - Computed over 5 dimensional space
    - Δx Δy and $\Delta\theta$ are the position and orientation differences of the vehicle in the initial and final states.
    - α and β are slopes of the terrain
    - The initial and final velocities are considered zero for each atomic maneuver
- On the average it takes 16 minutes to automatically synthesize a maneuver
- Maneuvers are robust with respect to small perturbations in the terrains
  - Tested by introducing waviness in terrain



MARYLAND
ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

N-Point Turn : Planning and Execution

Created by: Madan Dabbeeru, Josh Langsfeld, and Bhushan Pai

Advisors: Satyandra K. Gupta, Petr Svec, and Robert Kavetsky

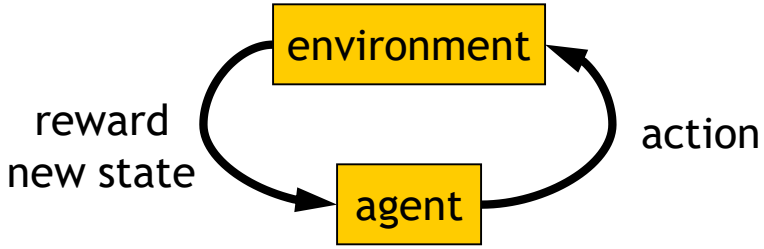
- Markov decision processes formally describe an environment for reinforcement learning

- The environment is fully observable

- Almost all RL problems can be formalized as MDPs:
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state
  - For every M.D.P. there exists an optimal policy

Markov Decision Process (MDP)

- set of states S, set of actions A, initial state S_0
- transition model P(s,a,s')
  - P([1,1], up, [1,2]) = 0.8
- reward function r(s)
  - r([4,3]) = +1
- goal: maximize cumulative reward in the long run



- policy: mapping from S to A
  - π(s) or π(s,a) (deterministic vs. stochastic)

- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment

Computing return from rewards

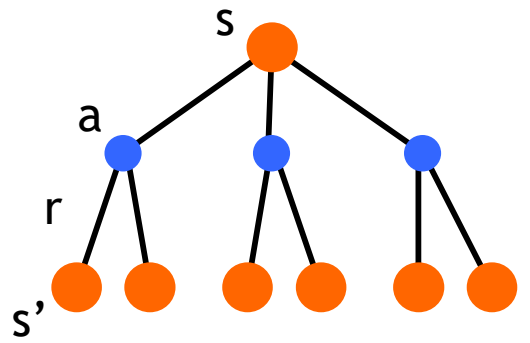
- episodic (vs. continuing) tasks
  - “game over” after N steps
  - optimal policy depends on N; harder to analyze

- additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

- discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

Value functions

- state value function: $V^{\pi}(s)$
  - expected return when starting in *s* and following π
- state-action value function: $Q^{\pi}(s,a)$
  - expected return when starting in *s*, performing *a,* and following π
- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$



$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

- Bellman equation

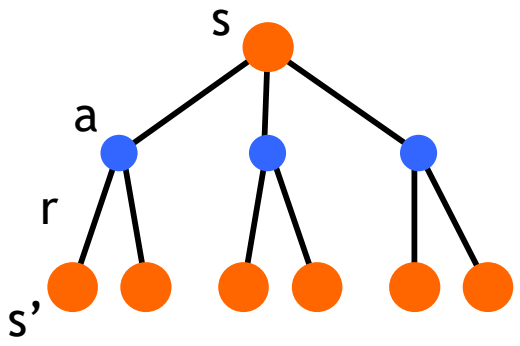
Optimal value functions

- there's a set of *optimal* policies
  - V^{π} defines partial ordering on policies
  - they share $V^*(s) = \max_{\pi} V^{\pi}(s)$ lue function

- Bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^*(s') \right]$$

  - system of n non-linear equations
  - solve for V*(s)
  - easy to extract the optimal policy



- having Q*(s,a) makes it even simpler

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Dynamic Programming

There are two types of tasks in RL

1.Prediction : This type of task predicts the expected total reward from any given state assuming the function **π(a|s)** is given.

(in other words) **Policy π** is given, it calculates the **Value function Vπ** with or without the model.

ex: **Policy evaluation** (we have seen in the last story in DP)

2. **Control** : This type of task finds the policy **π(a|s)** that maximizes the expected total reward from any given state.

(in other words) **Some Policy π** is given , it finds the **Optimal policy π***.

ex: **Policy improvement**

In RL problems we have two different tasks in nature.

1.Episodic task : A task which can last a finite amount of time is called **Episodic task (an episode)**

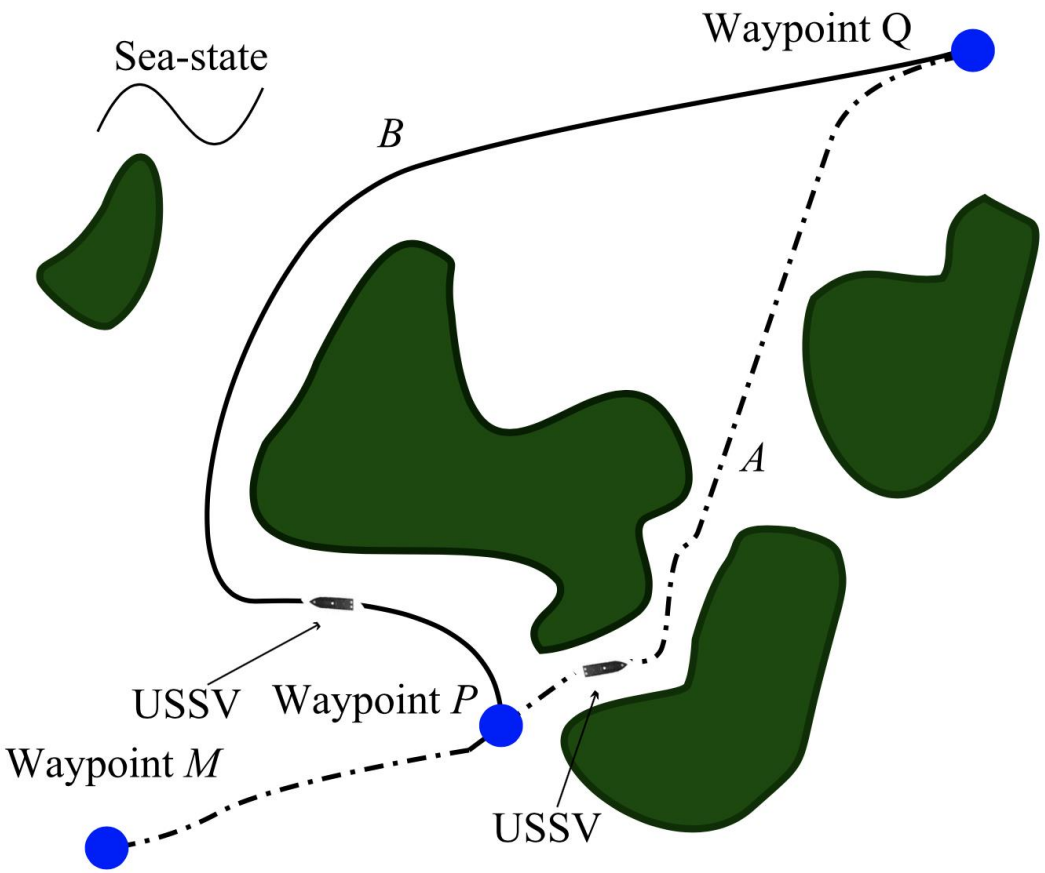
Ex : Playing a game of chess (win or lose or draw)

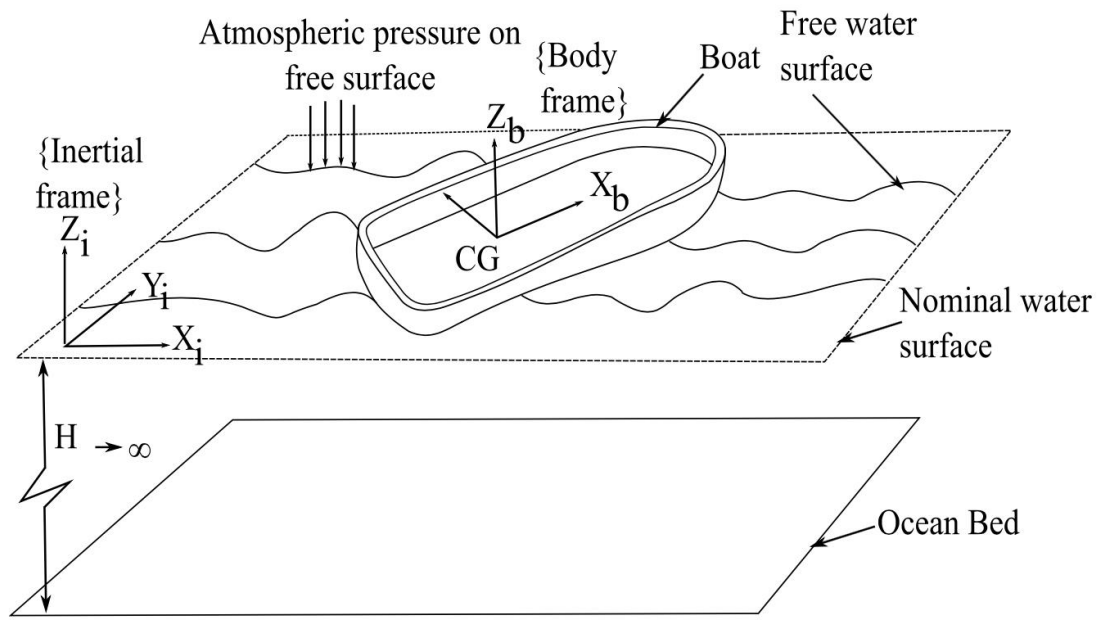
we only get the reward at the end of the task or another option is to distribute the reward evenly across all actions taken in that episode.

Ex: you lost the queen (-10 points), you lost one of the rooks (-5 points) etc..

2. **Continuous task** : A task which never ends is called **Continuous task**

Ex: Trading in the cryptocurrency markets or learning Machine learning on internet.


Sea-state
Waypoint Q
B
A
USSV
Waypoint P
Waypoint M
USSV


Atmospheric pressure on free surface
{Body frame}
Boat
Free water surface
{Inertial frame}
Z_b
X_b
CG
Z_i
Y_i
X_i
Nominal water surface
H → ∞
Ocean Bed

Dynamic sequential or temporal component to the problem

Programming optimising a "program", i.e. a policy

- c.f. linear programming

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - *Principle of optimality* applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused
- Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function v_π
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function v_*
  - and: optimal policy π_*

- Given a policy π
  - Evaluate the policy π

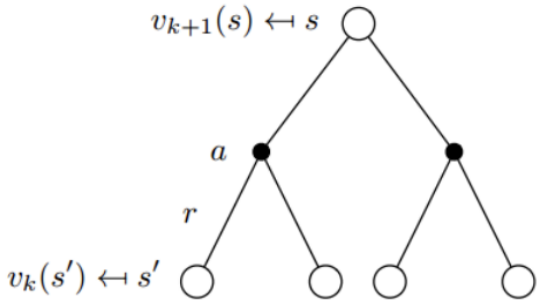
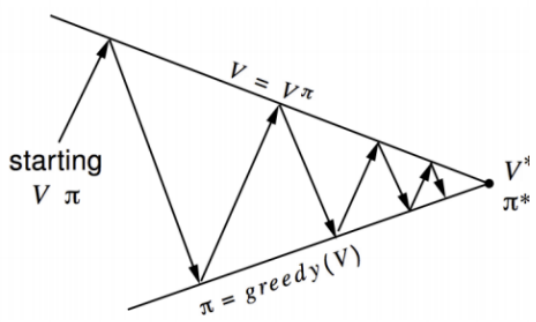
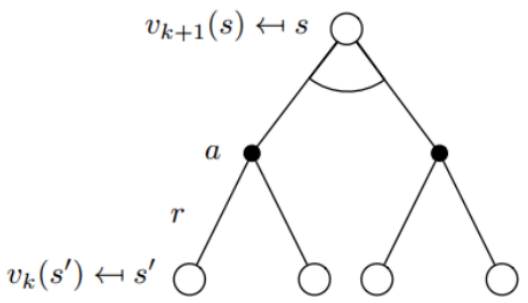
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

Dynamic Programming Algorithms

Algorithm	*Iterative Policy Evaluation*	*Policy Iteration*	*Value Iteration*
			
Bellman Equation	Bellman Expectation Equation	Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement	Bellman Optimality Equation
Problem	Prediction	Control	Control

Model-free Reinforcement Learning Algorithms

- In Model-free , we just focus on figuring out the value functions directly from the interactions with the environment
- All model free learning algorithms learn value functions directly from the environment.

*T*here are few approaches for solving these kind of problems

1.Monte carlo approach

2.Temporal-Difference approach

Monte Carlo Reinforcement Learning

The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward.

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
  - All episodes must terminate

MC methods learn directly from episodes of experience

MC is model-free: no knowledge of MDP transitions / rewards

MC learns from complete episodes: no bootstrapping

MC uses the simplest possible idea: value = mean return

Caveat: can only apply MC to episodic MDPs

All episodes must terminate

First-visit Monte Carlo policy evaluation

First-visit MC policy evaluation (returns $V \approx v_\pi$)

Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s
        Append G to $Returns(s)$
        $V(s) \leftarrow$ average($Returns(s)$)

By the law of large numbers, $V(s) \to^{\perp} v_{\downarrow}\pi(s)$ as number of episods $\to^{\perp} \infty$

Pros and cons of MC

MC has several advantages over DP:

- Can learn V and Q directly from interaction with environment (using episodes!)
- No need for full models (using episodes!)
- No need to learn about ALL states (using episodes!)

However, there are some limitations:

- MC only works for episodic (terminating) environments
- MC learns from complete episodes, so no bootstrapping
- MC must wait until the end of an episode before return is known

Next lecture

Solution: Temporal-Difference

- TD works in continuing (non-terminating) environments
- TD can learn online after every step
- TD can learn from incomplete sequences

Temporal Difference Learning (MonteCarlo + DP)

Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Temporal difference (TD) learning, which is a model-free learning algorithm, has two important properties:

- It doesn't require the model dynamics to be known in advance
- It can be applied for non-episodic tasks as well

The TD learning algorithm was introduced by the great Richard Sutton in 1988. The algorithm takes the benefits of both the Monte Carlo method and dynamic programming (DP) into account:

- Like the Monte Carlo method, it doesn't require model dynamics, and
- Like dynamic programming, it doesn't need to wait until the end of the episode to make an estimate of the value function

Instead, temporal difference learning approximates the current estimate based on the previously learned estimate. This approach is also called bootstr

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

MC & TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

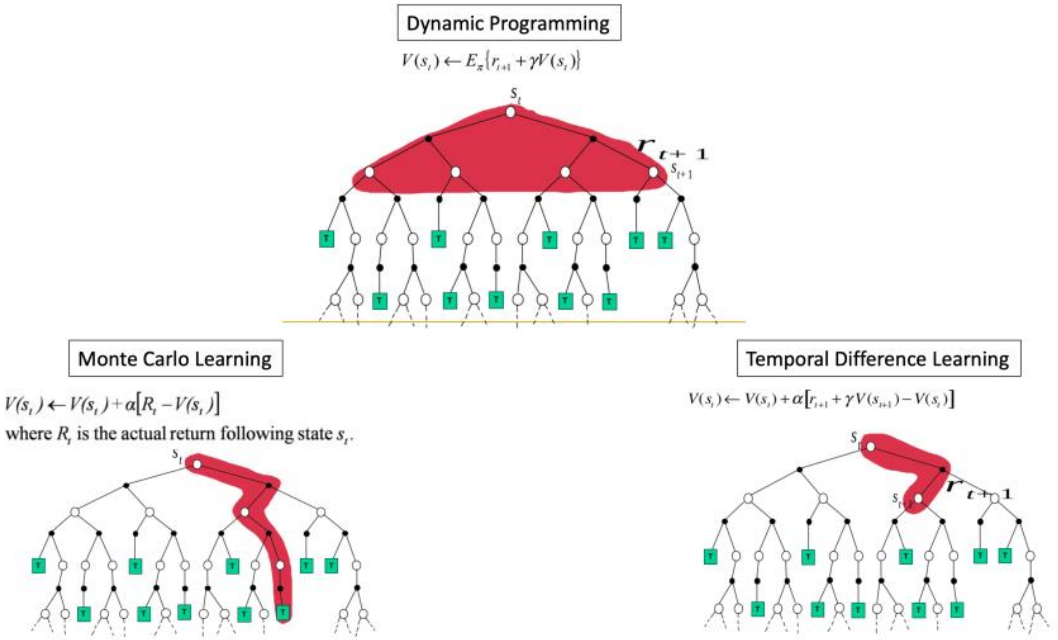
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Advantages of MC vs TD

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Temporal Difference Learning

When we learn a value function with a batch (Monte-Carlo) algorithm, we need to wait until N steps are done, then we can update.
Temporal difference learning is a iterative way of learning a value function, such that you can change the value function every step.



Temporal difference (TD) learning, which is a model-free learning algorithm, has two important properties:
•It doesn't require the model dynamics to be known in advance
•It can be applied for non-episodic tasks as well

Multi-Armed Bandit Problem

- Robust Trajectory Selection for Rearrangement Planning as a Multi-Armed Bandit Problem

Exploration & Exploitation

Restaurant Selection

- Exploitation Go to your favourite restaurant
- Exploration Try a new restaurant

Online Banner Advertisements

- Exploitation Show the most successful advert
- Exploration Show a different advert

Oil Drilling

- Exploitation Drill at the best known location
- Exploration Drill at a new location

Game Playing

- Exploitation Play the move you believe is best
- Exploration Play an experimental move

Gather enough information to make the best overall decisions

- Robust Trajectory Selection for Rearrangement Planning as a Multi-Armed Bandit Problem

Background: Bandits

μ_1

μ_2

μ_3

Bandit "arms"

(unknown reward probabilities)

Pull arms sequentially so as to maximize the total expected reward

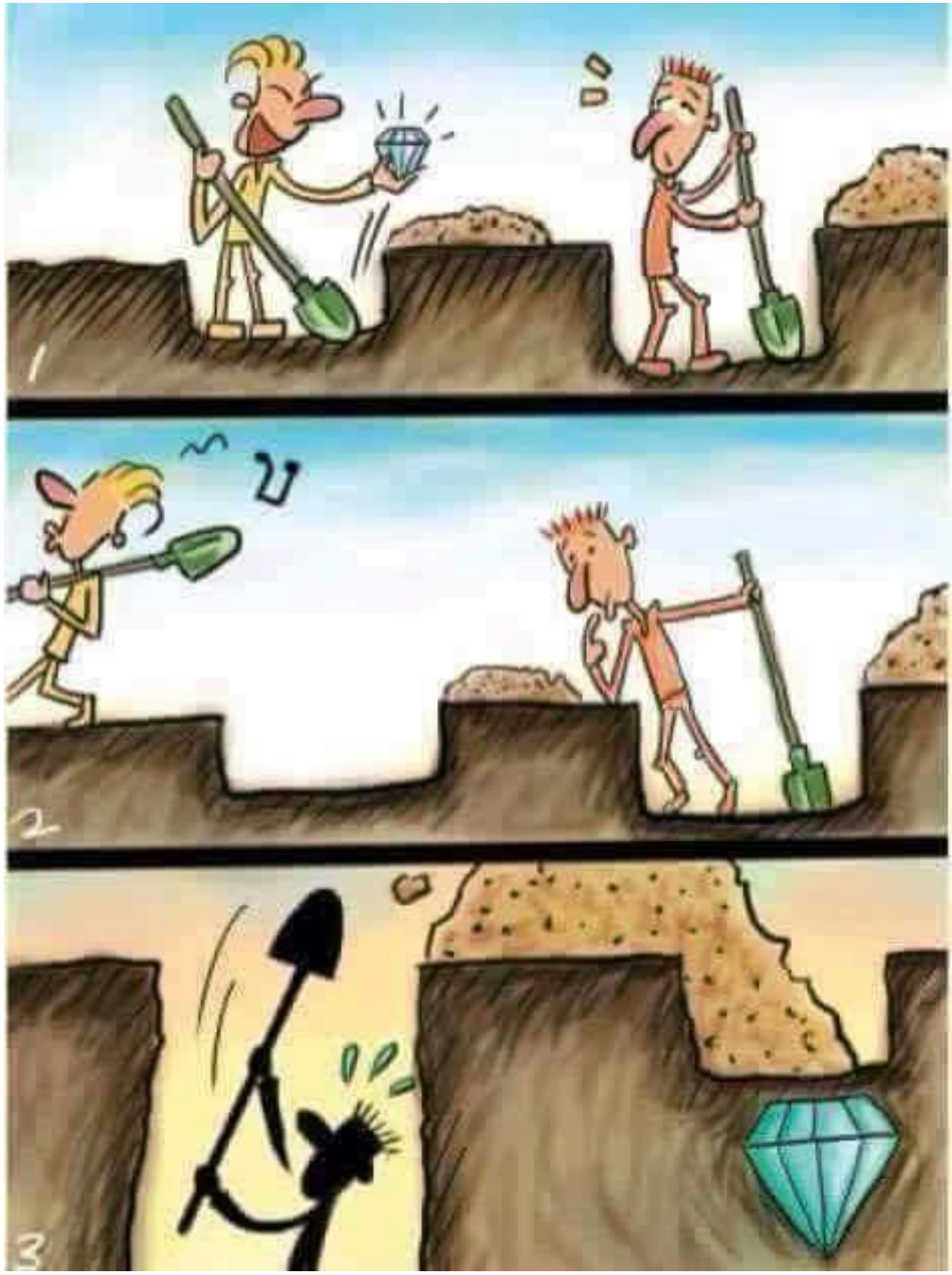
- Show ads on a webpage to maximize clicks
- Product recommendation to maximize sales

How to balance exploration and exploitation in reinforcement learning

Exploration: – try out each action/option to find the best one, gather more information for long term benefit • Exploitation: – take the best action/option believed to give the best reward/payoff, get the maximum immediate reward given current information.

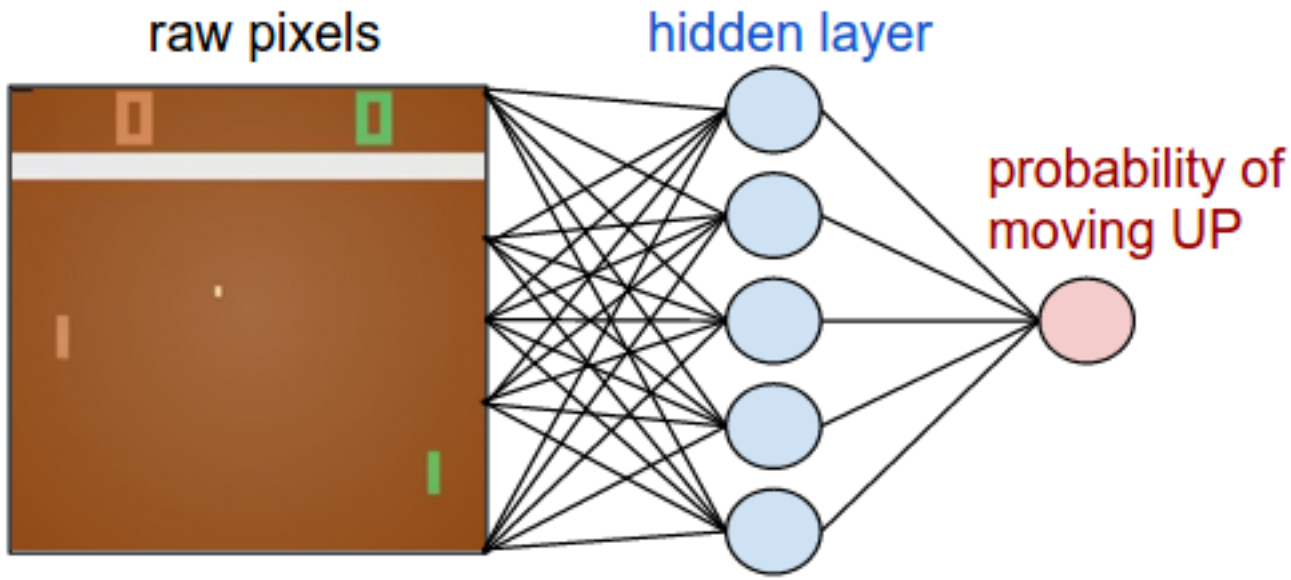
Questions: Exploration-exploitation problems in real world?

Select a restaurant for dinner
– Medical treatment in clinical trials
– Online advertisement
– Oil drilling



Evolutionary Strategies for Reinforcement Learning

Deep Reinforcement Learning: Pong from Pixels



https://github.com/paraschopra/deepneuroevolution/blob/master/openai-gym-cartpole-neuroevolution.ipynb

Evolutionary Neural Networks

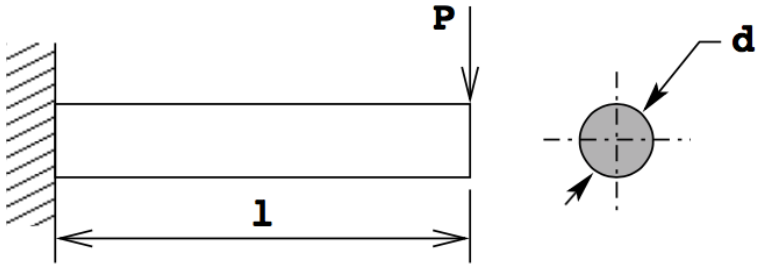
Human Following Robot with
Active Vision Based on Kinect

Created by: James Koo, Jimmy Tanner, Yalun Wu,
Atul Thakur, Petr Svec

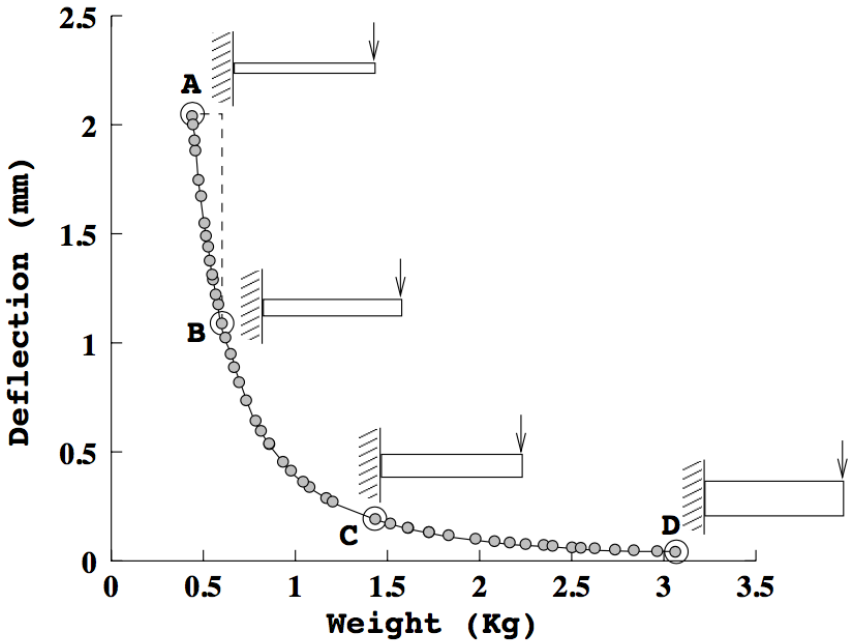
Advisors: Satyandra K. Gupta

Few Examples

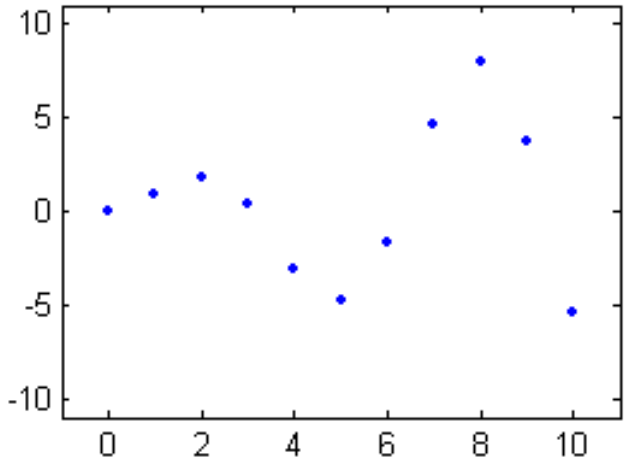
How do you make an agent who design a cantilever beam ?

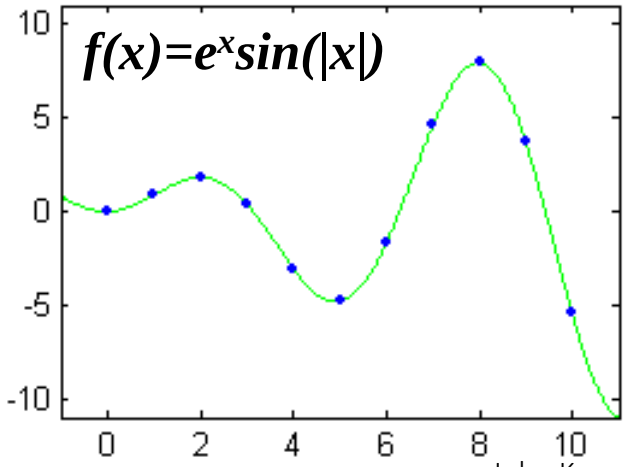


$$\left.\begin{aligned} \text{Minimize} \quad & f_1(d,l) = \rho\frac{\pi d^2}{4}l, \\ \text{Minimize} \quad & f_2(d,l) = \frac{64Pl^3}{3E\pi d^4}, \\ \text{subject to} \quad & \sigma_{\max} \le S_y, \\ & d^{(L)} \le d \le d^{(U)}, \\ & l^{(L)} \le l \le l^{(U)}, \end{aligned}\right\}$$

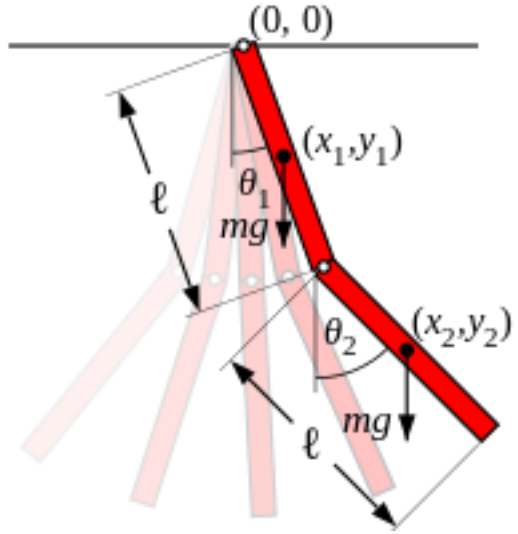


What function describes this data?



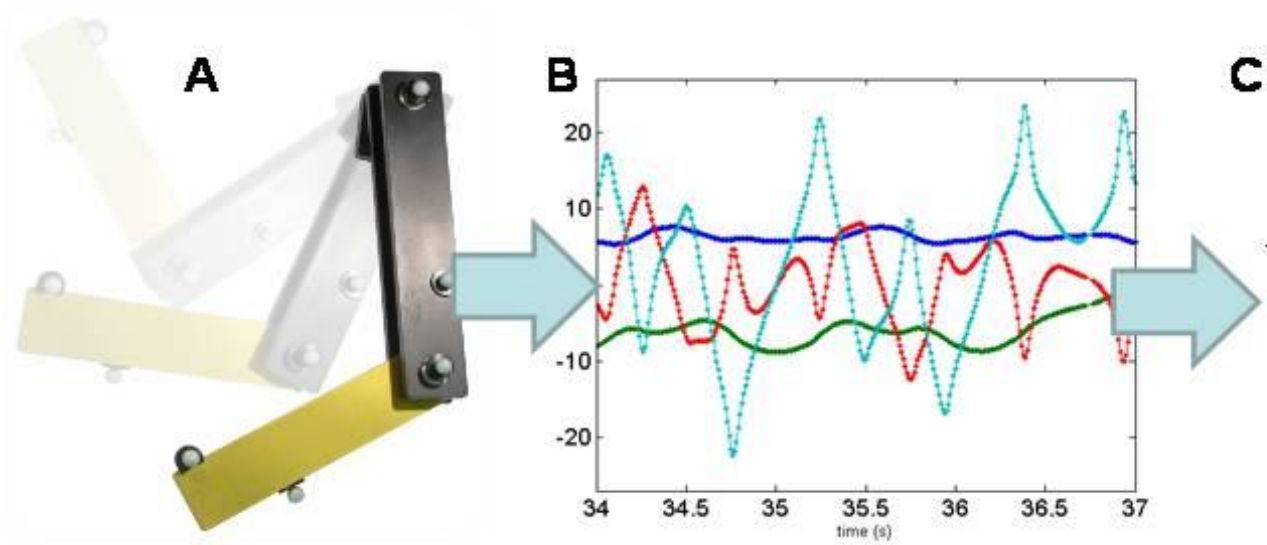


John Koza, 1992



$$\ddot{\theta}_1 = \frac{\begin{array}{c} -m_2\cos(\theta_1-\theta_2)l_1\dot{\theta}_1^{\,2}\sin(\theta_1-\theta_2)+m_2\cos(\theta_1-\theta_2)g\sin(\theta_2) \\ -m_2 l_2\dot{\theta}_2^{\,2}\sin(\theta_1-\theta_2)-(m_1+m_2)g\sin(\theta_1) \end{array}}{l_1(m_1+m_2-m_2\cos^2(\theta_1-\theta_2))}$$

$$\ddot{\theta}_2 = \frac{(m_1+m_2)\left(\begin{array}{l} l_1\dot{\theta}_1^{\,2}\sin(\theta_1-\theta_2)+\dfrac{\dot{\theta}_2^{\,2}\sin(\theta_1-\theta_2)\cos(\theta_1-\theta_2)m_2 l_2}{m_1+m_2} \\ +\cos(\theta_1-\theta_2)g\sin(\theta_1)-g\sin(\theta_2) \end{array}\right)}{l_2\left(m_1+m_2\sin^2(\theta_1-\theta_2)\right)}$$



From Data:

x	y	…
0.1	2.3	
0.2	4.5	
0.3	9.7	
0.4	5.1	
0.5	3.3	
0.6	1.0	
…	…	…

Calculate partial derivatives Numerically:

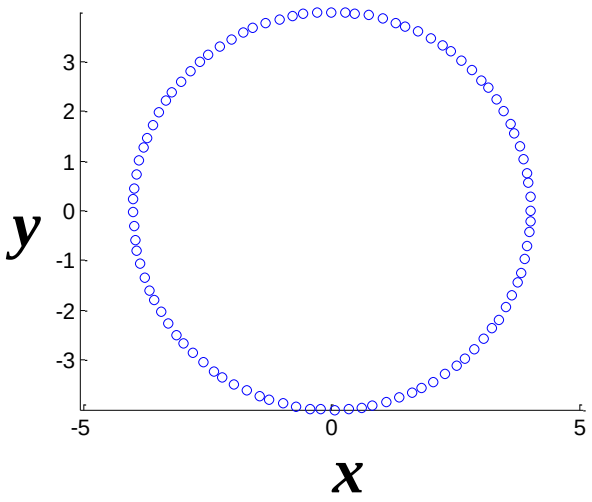
$$\frac{\delta x}{\delta y}, \quad \frac{\delta y}{\delta x}, \quad \dots$$

From Equation:

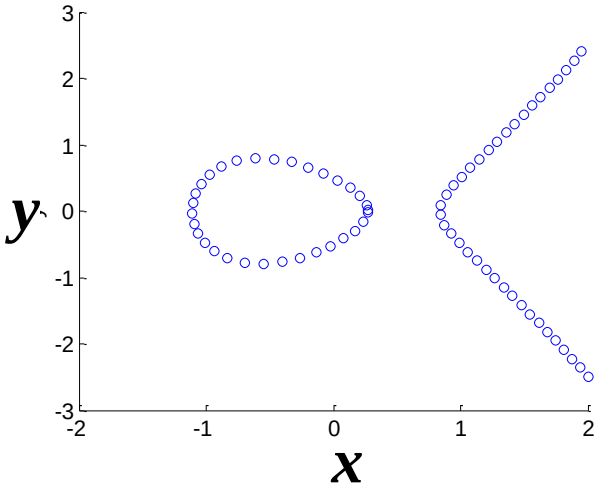
Calculate predicted partial derivatives Symbolically:

$$\frac{\delta f}{\delta x} \Big/ \frac{\delta f}{\delta y} \rightarrow \frac{\delta x'}{\delta y'}, \quad \frac{\delta y'}{\delta x'}, \quad \dots$$

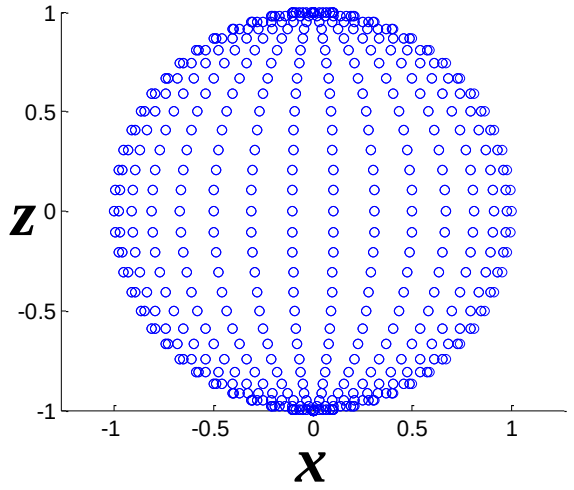
How do we learn ?



$x^2 + y^2 - 16 = 0$



$x^3 + x - y^2 - 1.5 = 0$



$x^2 + y^2 + z^2 - 1 = 0$