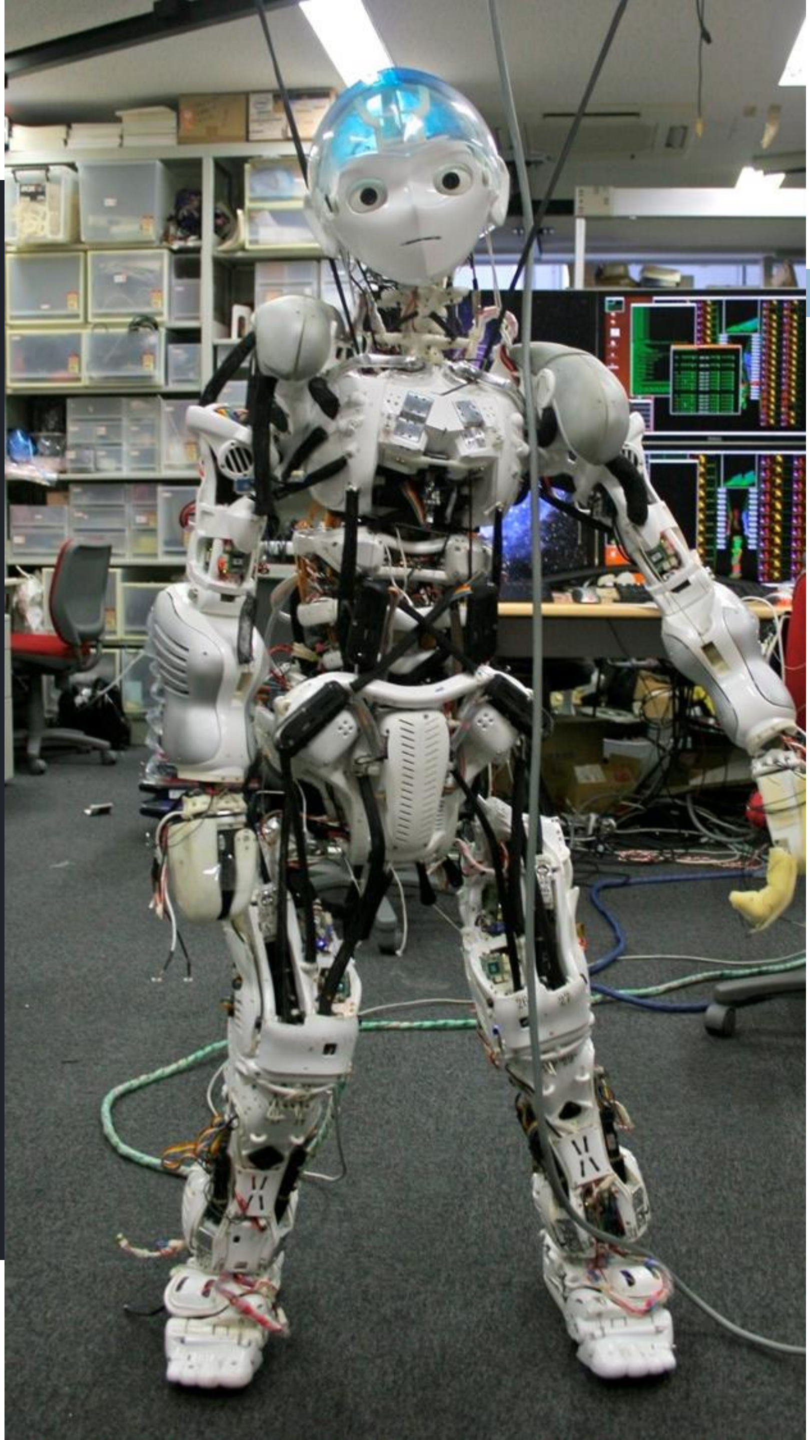
CS6482 Deep RL

E: Reinforcement Learning: the basics

J.J. Collins

Dept. of CSIS

University of Limerick



Part 1: Objectives

- ❑ Part 1: Outline the motivation for RL
  - ▪ Describe the challenge
- ❑ Part 2: Mathematical Basis underpinning Reinforcement Learning
- ❑ Part 3: Dynamic Programming
- ❑ Part 4: Monte Carlo Methods

Outline

Based on:

Chapter 1-5 in R. Sutton and A. Barto. Reinforcement Learning, 2nd Edition, MIT Press. 2018.

Available online.

Part 1

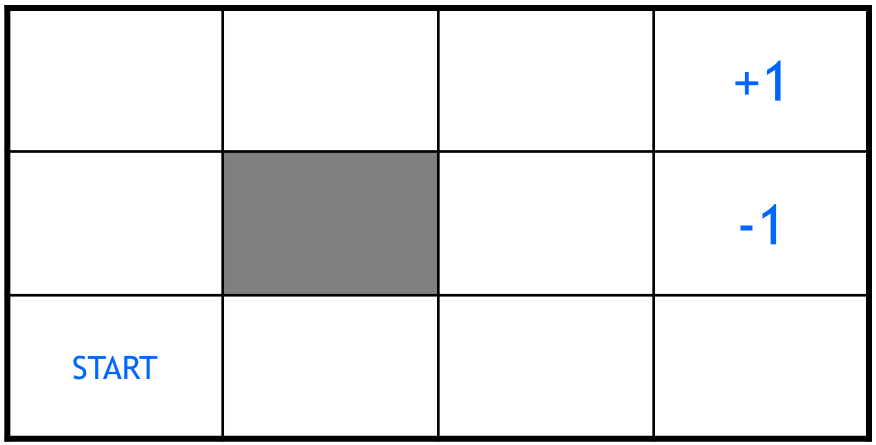
Outline the motivation for RL

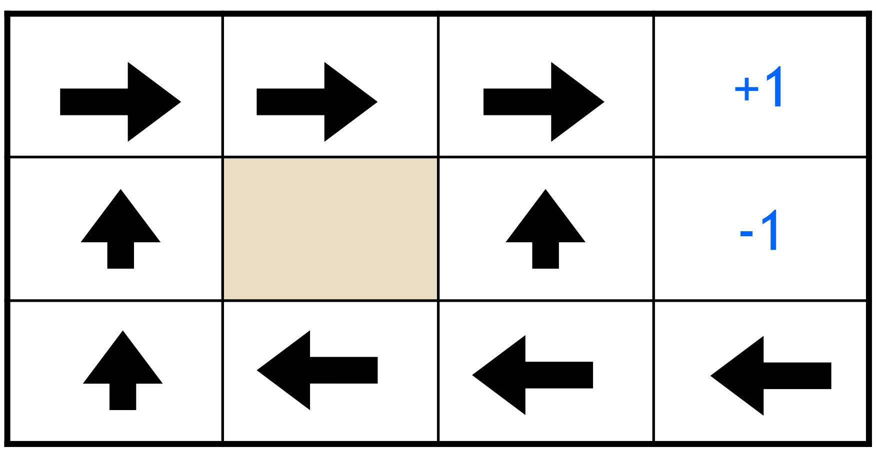
Describe the challenge(s)

Motivation: A Use Case

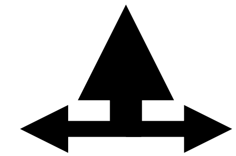
- Learning to play checkers/Connect4/chess/backgammon/go ......
- Knowing the rules is not sufficient to be competitive
- Solution: Reinforcement Learning (RL)
- RL→ learning by **Trial and Error**
- Learn → **optimal mapping from states to actions**
  - State → board configuration
    - Connect 4: 10^{12} states
  - Actions → permissible actions allowed for a particular state
- Maximising long term **REWARD**
  - Reward for a game: +1 for a win, 0 for a loss

Robot in a room



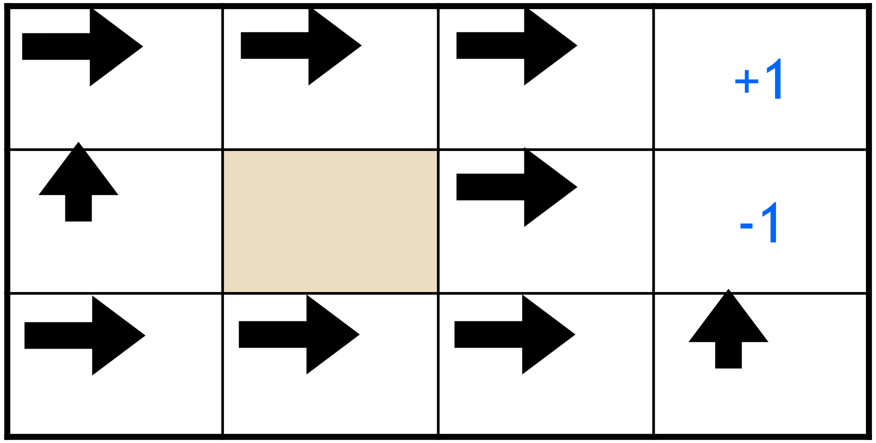


- **States**: 12 cells
- **Actions**: 3 options in each state with probabilities
  - P(up) = 0.8
  - P(left) = 0.1
  - P(right) = 0.1



- **Rewards:**
  - +1 at [4,3]
  - -1 at [4,2],
  - -0.04 for each step otherwise
- **Goal:** keeping moving until agent reaches a terminating cell
- What is the solution?
  - **Policy:** mapping from states to actions
  - Is the policy on the left optimal
  - How can an agent learn a policy?

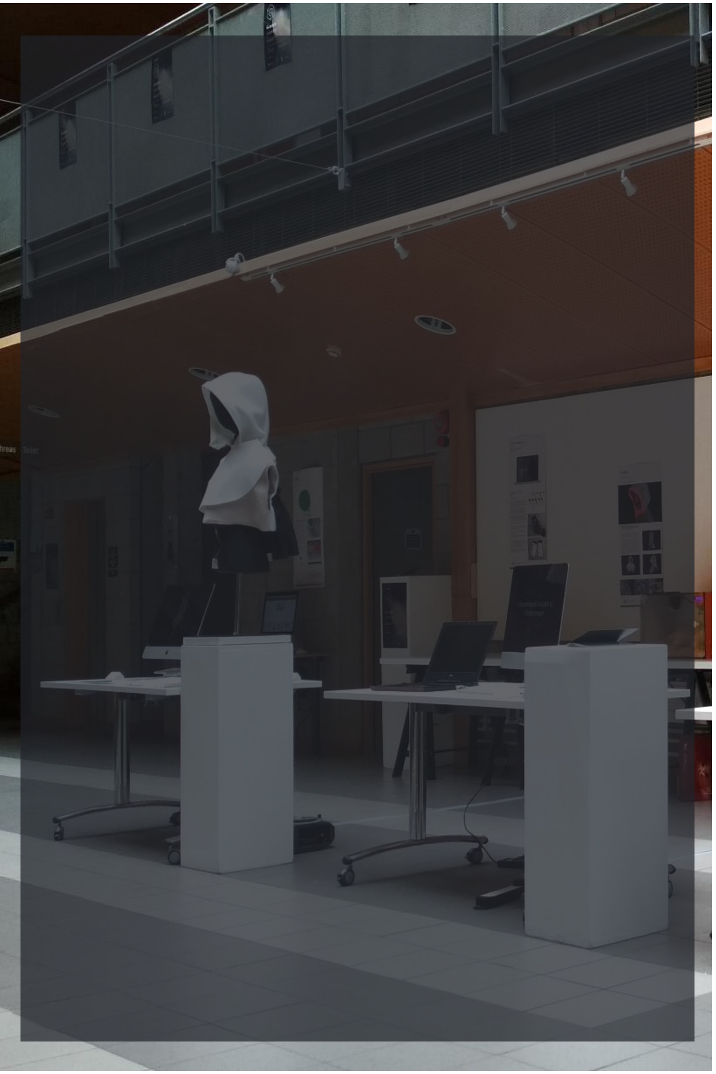
Reward for each step: -2



Some Key Challenges

- BALANCE **EXPLORATION** of new states versus **EXPLOITATION** of acquired knowledge when playing

- **Credit Assignment Problem**: in sequential decision making process, how much credit should go to each decision (move)?



Reflect

- Can an agent learn to play a game using supervised learning?
- Think up of 3 scenarios suitable for RL.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Reflection

Exercise: describe the following classical learning types:

Supervised learning is …

Unsupervised/Competitive is …

Reinforcement learning is …

POE Model

Phylogenetic Learning: genetic learning i.e. evolution

Ontogenetic Learning: development of single celled organisms

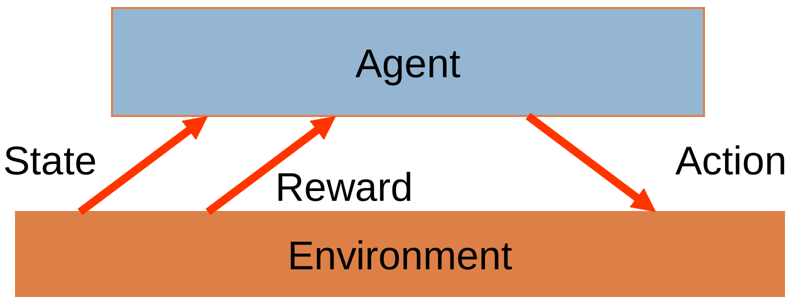
Epigenetic Learning: learning during an individual's lifetime

M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 83-97, April 1997.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

RL is a Markov Decision Process (MDP)

- MDP model <S,T,A,R>

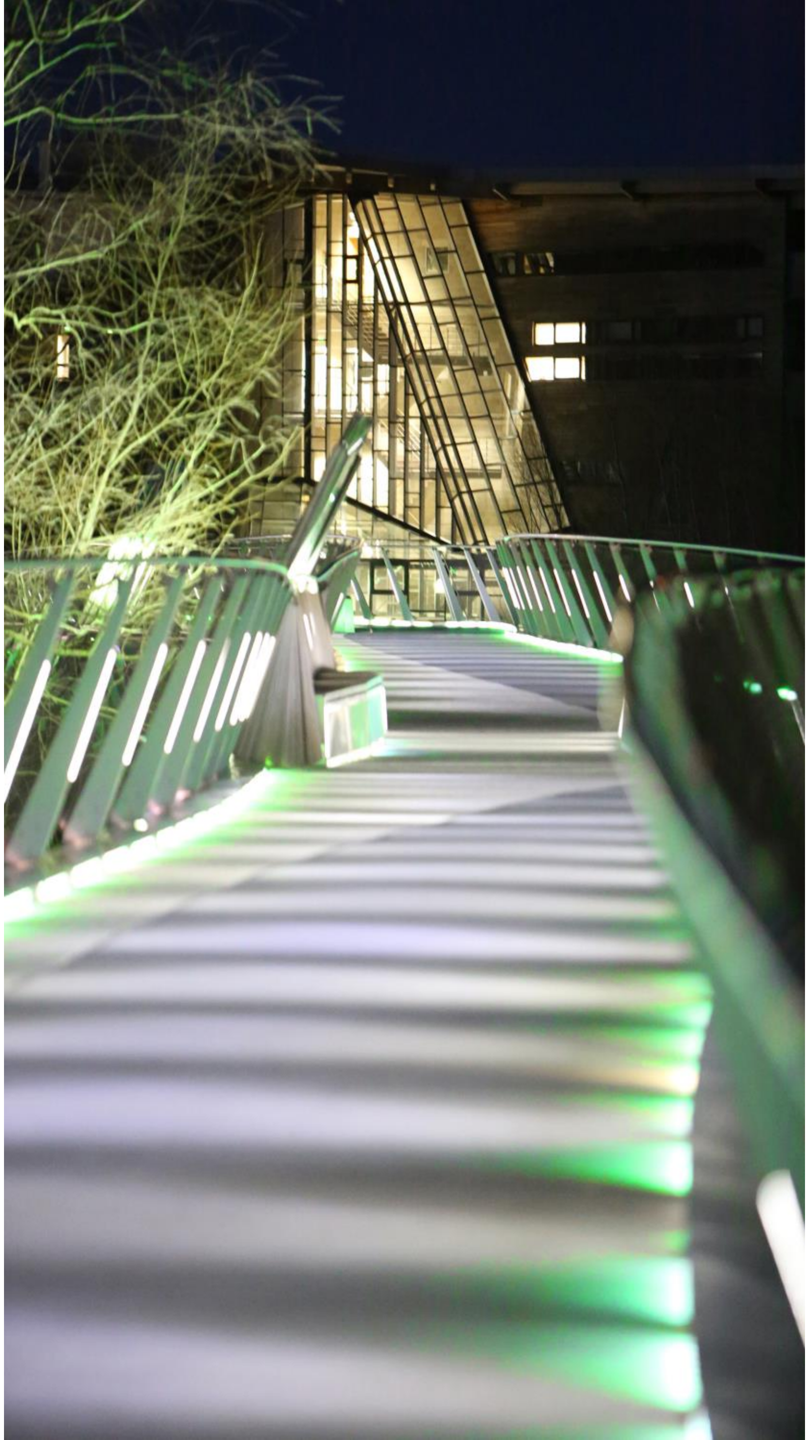


$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2} s3$$

- S– set of states
- A– set of actions
- T(s,a,s') = P(s'|s,a)– the probability of transition from *s* to *s'* given action *a*
- R(s,a)– the expected reward for taking action *a* in state *s*

$$R(s,a) = \sum_{s'} P(s'|\, s,a) r(s,a,s')$$

$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$



The Elements

A **POLICY** states what action to take in state s

A **REWARD** is a signal received by the learning agent from the environment.

> The objective of the goal directed agent is to maximise long term reward.

A **VALUE FUNCTION** specifies the likely long term reward that will accrue from a state s

> Where reward is an indicator of what is good in the short term, a value function specifies what is good in the long term

> Metaphor → rewards are like pain/pleasure, value functions are higher order judgements.

Model of the environment (optional)

Value Functions

- V(s)
  - The state value function.
  - What is the value of state s?
  - For the game Connect4, state s is a specific board configuration
  - What is the value of the board on the right?
  - Associated with PREDICTION.

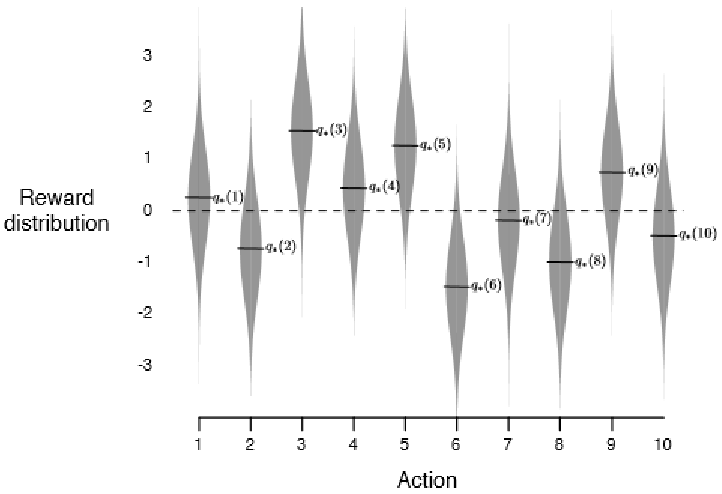
- Q(s,a)
  - The state-action value function.
  - What is the value of taking action a in state s?
  - For example, what is the value of dropping a coin into the first column for the board on the right?
  - Associated with CONTROL.

The Need for Generalization in RL

State (Inputs)							Actions			
	Current Speed	Weight of Passeng ers	Distance to vehicles in front	Depth of Brake Pad	Road Cond.	Driving style	Brake	Accelerate		Turn Steerin g
S1	120	6	25	N/A	Wet	Conserv .	?	?	?	?
S2										
S3										
S4										
….										
S_N										

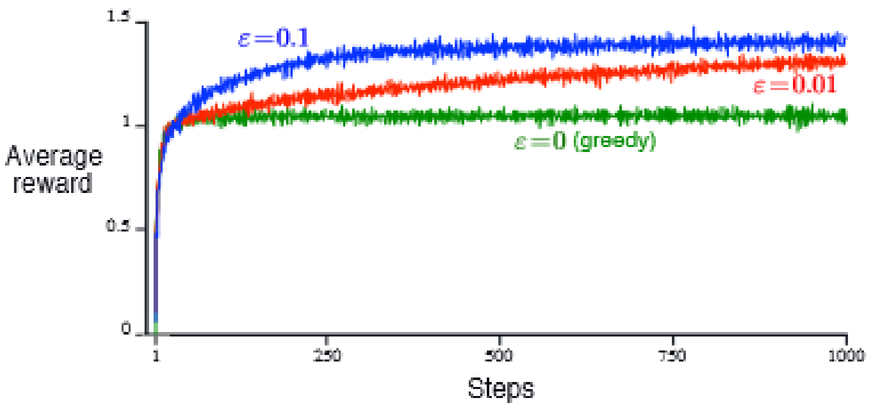
K-armed Bandit

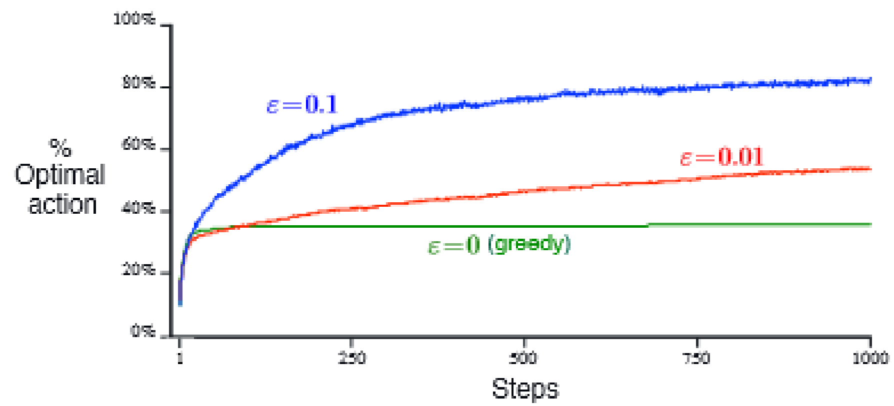
- K-armed bandit, i.e. K 1-armed bandits, let K = 10
  - Select one of the Ks, pull the lever and get a reward .
  - Play 1000 times and maximise pay out (return)
  - Each K has a normal distribution with mean 0 and variance 1.
- Non Associative task
- Stationary Environment
- Which lever gives the best pay



K-armed Bandit

- Greedy action selection: always select the lever (action) that maximises reward observed todate
- ϵ-Greedy: select the best lever with probability 1-ϵ (exploit), otherwise select a random lever (explore)





The Sample Averaging Method

- For one action a, let R_i be the reward received after the ith selection
- Its ACTION VALUE is:
- $Q_n = \frac{R_1 + R_2 + \ldots + R_{n-1}}{n-1}$
- Can be computed incrementally
- $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) =$ …. (see page 31)
- $= Q_n + \frac{1}{n}[R_n - Q_n]$
- New_Estimate = Old_Estimate + Step_Size [Target – Old_Estimate]
- Where [Target – Old_Estimate] = Error

The Sample Average Method

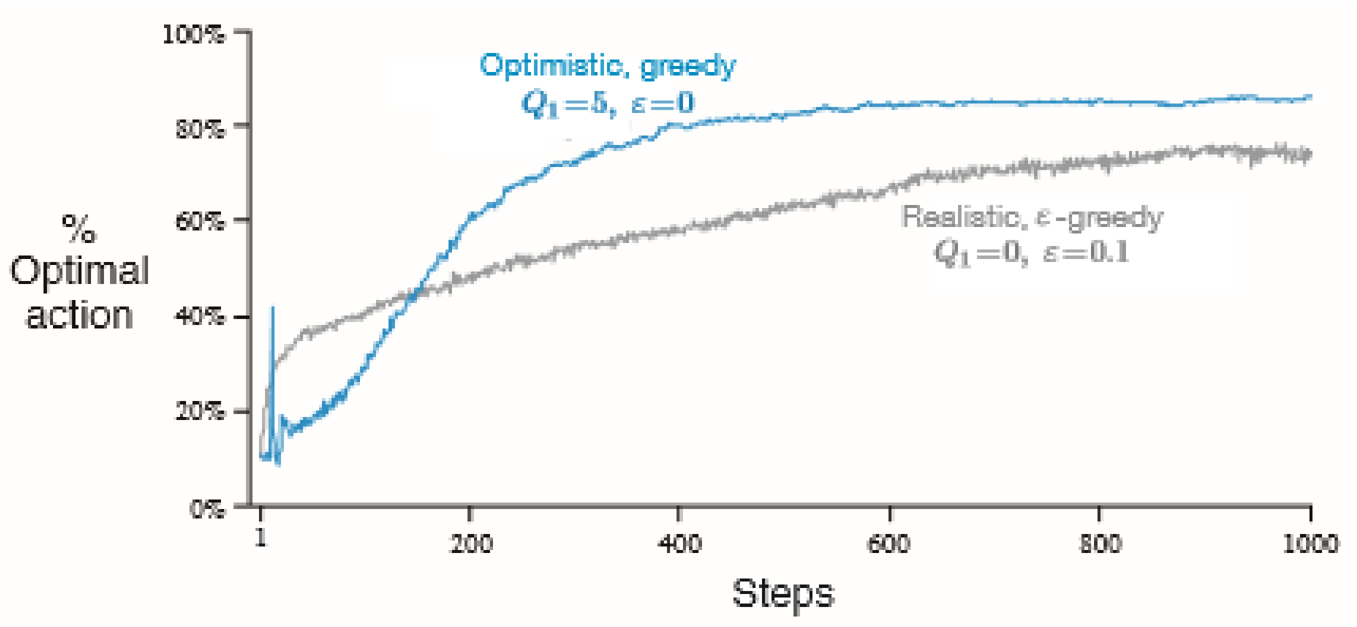
Sample Averaging Method for Stationary Armed Bandit

For a = 1 to k
 Q(a) ← 0
 N(a) ← 0

Loop for num_times
  A ← argmax_a Q(a) with probability 1-ϵ | a random action with probability ϵ
  R ← Bandit(A)
  N(A) ← N(A) + 1
  Q(A) ← Q(A) + $\frac{1}{N(A)}$[R-Q(A)]

Optimistic Initialisation

- Selecting an optimistic initial value for Q(A)

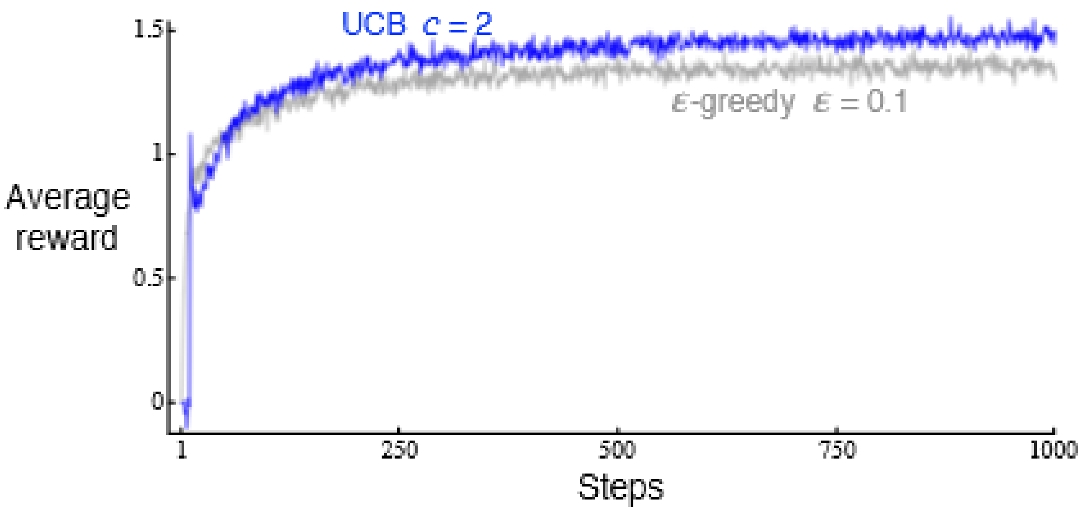


Nonstationary Problem

- Weighted Average
- $Q_n = Q_n + \alpha[R_n - Q_n]$
- $\quad = \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$
- $\quad = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_t$
- The weight given to R_i decreases as the number of intervening trials increases

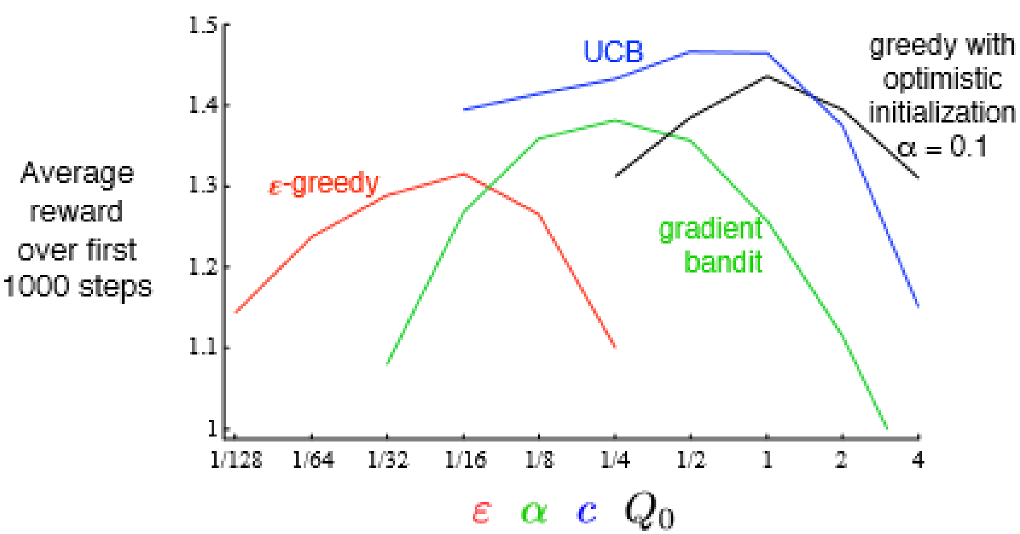
Upper Confidence Bound (UCB) Action Selection

- The action A to be selected at time t
- $A_t = \quad argmax_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$
- Where c > 0 controls exploration
- If $N_t(a) = 0$; then a is the maximising action
- Critique
  - Non-stationary
  - Generalisation



Finally

- Did not look at gradient based assent using SoftMax
- Did not look at associative tasks i.e. trial and error learning in many states → mapping from states to actions
  - Contextual K-armed Bandit
- Next
  - Dynamic Programming
  - Monte Carlo Methods



Part 2

Mathematical Basis underpinning Reinforcement Learning

Based on Chapter 3 in Sutton and Barto (2018)

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Discounted Expected Reward (chpt 3)

- **For episodic (terminating) tasks**
- Maximise expected return G at time t
- $G_t = R_{t+1} + R_{t+2} + R_{t+3} \ldots.. + R_T$
- Where T is the final step
- **For non-terminating tasks**
- $G_t = R_{t+1} + \gamma R_{+2} + \gamma^2 R_{t+3} \ldots.. = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- Where γ is the discount parameter $0 < \gamma < 1$
- $G_t = R_{t+1} + \gamma G_{t+1}$
- **Unified**
- $G_t = \sum_{k=t+1}^{T} \gamma^{k-t+1} R_k$

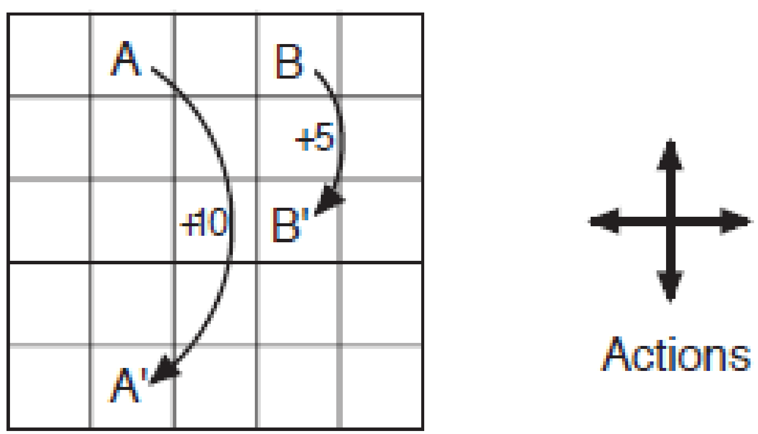
Value Functions (chpt 3)

- RL – learning value functions
  - Functions of State – v(s)
  - Functions of State-Action pairs – q(s,a)
- A policy π that specifies the action a to take in state s
- $v_\pi(s) \cong E_\pi[G_t \mid S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s], \forall s \in S$
- $q_\pi(s,a) \cong E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a],$
- v is called the value function for policy π
- q is the action value function for policy π

Value Functions (chpt 3)

Gridworld

- 4 actions: north, south, east, west.
- Reward is 0 for each step, except for
  - From State A, all actions move the robot to A’ with a reward of +10
  - From State B, all actions move the robot to B’ with a reward of +5
- Actions that move the robot off the grid leave the agent state unchanged with a reward of -1
- Policy π: all actions are equiprobable
- V π shown on the right in the diagram

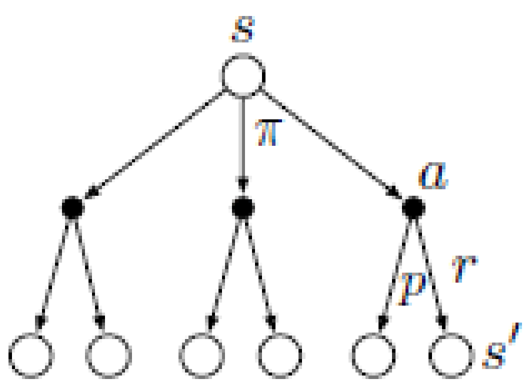


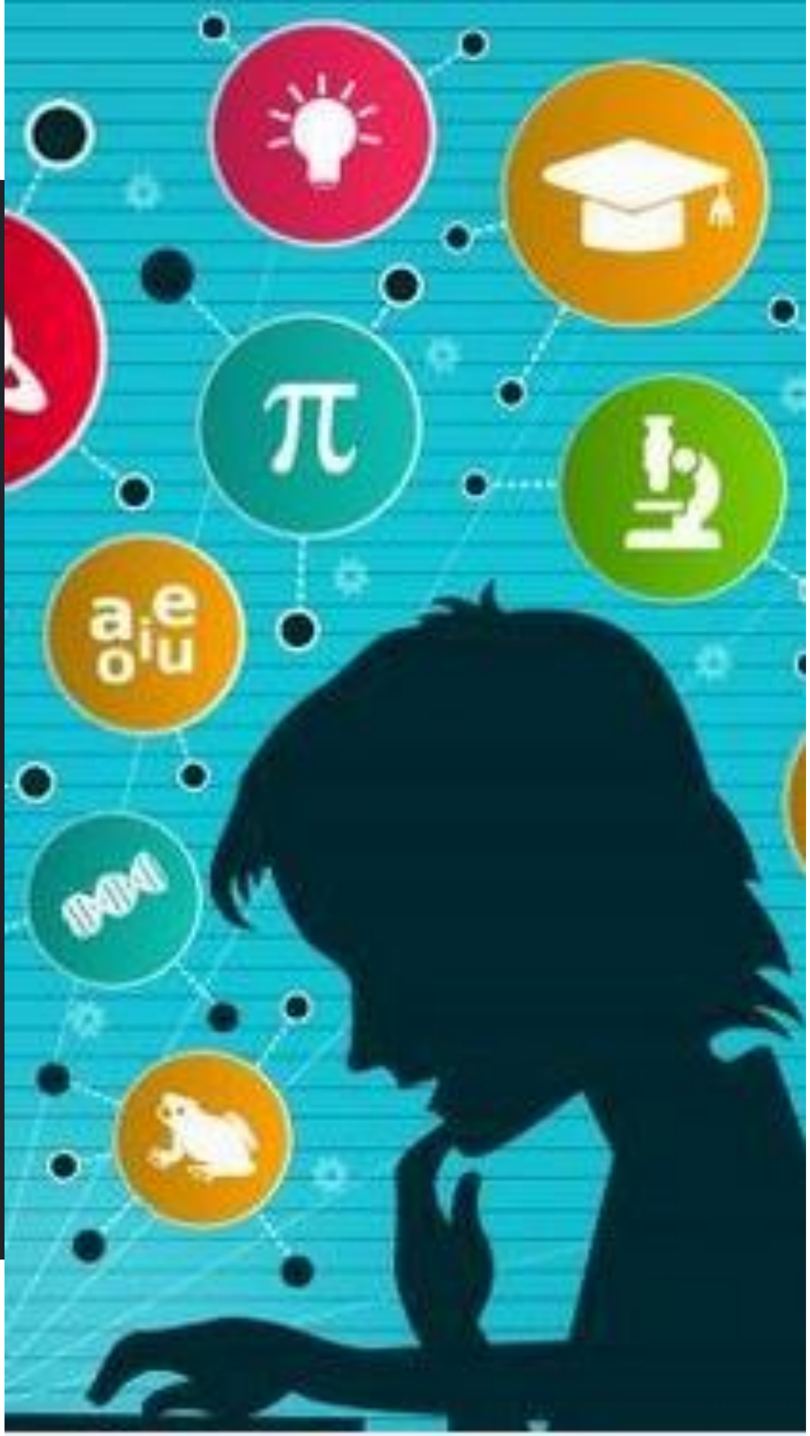
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Bellman's Equation for v_π (chpt 3)

- $v_\pi(s) \cong E_\pi[G_t \mid S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s], \forall s \in S$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']\right]$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \forall s \in S$
- **Backing up the values**
  - Diagram shows guesstimate $v_\pi(s')$being backed up
- Similar for $q_\pi(s, a)$

- **Exercise**
- Draw the backup diagram for q_π





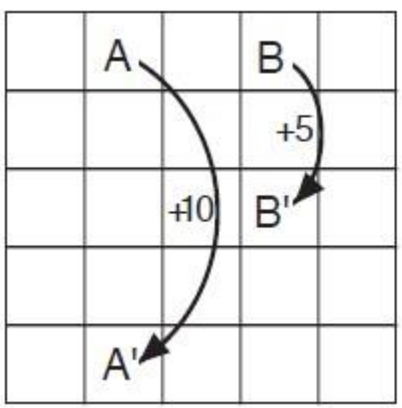
- **Bootstrapping** accelerates the process by using approximations
  - Process = Learning
  - Can lead to instability
- **Model**
  - $p(s', r|s, a)_\pi$

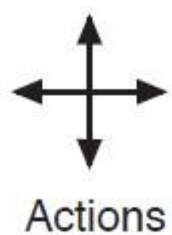
Bellman's Optimality Equation

- $v_*(s) = \ max_\pi\big(v_\pi(s)\big)$
- $v_*(s) = \ max_{a \in A(s)} q_{\pi *}(s, a)$
- $v_*(s) = \ max_a \sum_{s',r} p(s', r \mid s, a) \, [r + \ \gamma v_*(s')]$
- This is Bellman's Optimality Equation (derivation shown on page 63)
  - A system of equations, one for each state
  - N states = n equations in n unknowns
  - Solvable if dynamics of environment known i.e. a model is available
- Likewise for q
- $q_*(s, a) = \ max_\pi\big(q_\pi(s, a)\big)$

Bellman's Optimality Equation

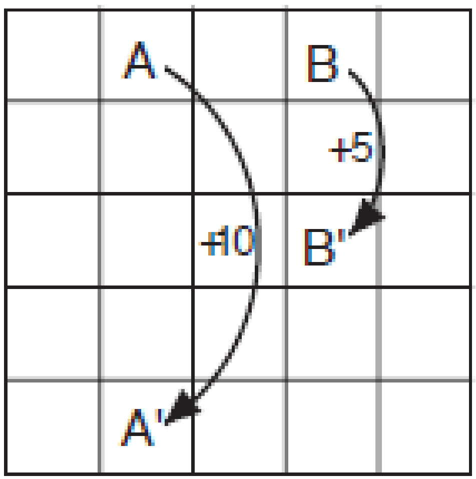
- Gridworld example
- Actions: north, south, east, and west
- Actions that would take the agent off the grid leave it in the same cell with a reward of -1
- Transition from A to A' generates a reward of +10
- Transition from B to B' generates a reward of +5
- All other transitions generate a reward of 0
- Policy: all actions equiprobable
- Shown state value function (V)





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

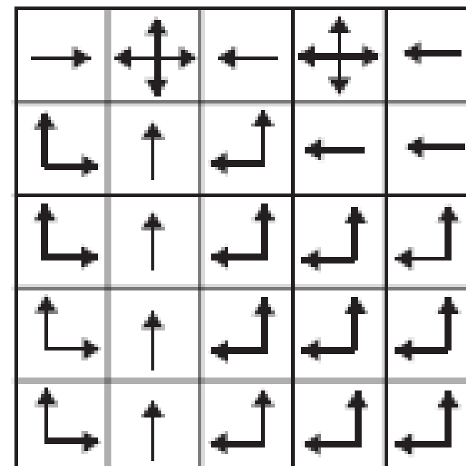
Bellman's Optimality Equation



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

v_*



π_*

Part 3

Dynamic Programming

Based on Chapter 4 in Sutton and Barto (2018)

RL Value-Based Approaches

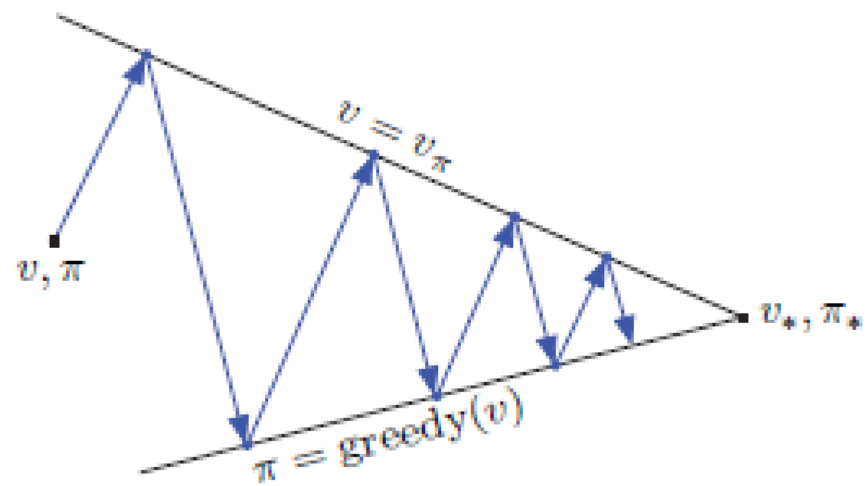
1. Dynamic Programming (DP)
   - Requires a complete model
   - All transitions known
2. Monte Carlo (MC) Methods
   - Model Free
3. Temporal Difference (TD) Methods
   - Sarsa and Q learning

- Alternative Approach based on Policy Gradients i.e. Actor Critic, PPO, etc.

Dynamic Programming (chpt 4)

- Algorithm: **Generalised Policy Iteration (GPI)** on page 82

1. Start with an arbitrary policy π
2. Loop until policy is stable
   a) Compute v_π for policy π (using iterative policy evaluation)
   b) Improve the policy
3. Output $v_\pi \approx v_*$



Dynamic Programming

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Dynamic programming

- **Value Iteration** combines policy and evaluation into one shot

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

Part 4

Monte Carlo Methods

Based on Chapter 5 in Sutton and Barto (2018)

Monte Carlo Methods (chpt 5)

- Model Free

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Summary

- Introduced DP and MC methods as solutions to RL problem
  - Bootstrapping: making a guess of $v_\pi(s)$ using a guesstimate i.e. $v_\pi(s')$
    - DP bootstraps, MC does not.
  - Model
    - DP requires a model, MC does not.

- Next **Temporal Difference (TD) methods** – bridges DP and MC
  - **Leading to Deep Q Networks**

- **Please do look at exercises in the chapters. Have a go!**

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Summary

- Introduced DP and MC methods as solutions to RL problem
- Bootstrapping: making a guess of $v_\pi(s)$ using a guesstimate i.e. $v_\pi(s')$
  - DP bootstraps, MC does not.
- Model
  - DP requires a model, MC does not.
- Next **Temporal Difference (TD) methods** – bridges DP and MC
  - **Leading to Deep Q Networks**

- **Please do look at exercises in the chapters. Have a go!**

Homework

Study these slides carefully

Light read of chapters 1 and 2 from Sutton and Barto’s 2018 text Reinforcement Learning (2nd edition), available online.

> http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

Reflect on how one might learn an optimal mapping from states to actions for a goal directed agent without a model.

Homework

Excellent RL Course by David Silver (UCL and DeepMind) on Youtube

https://www.youtube.com/watch?v=2pWv7GOvuf0

Thank you

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 T9PX,
Ireland.

Ollscoil Luimnigh,
Luimneach,
V94 T9PX, Éire.

+353 (0) 61 202020

ul.ie