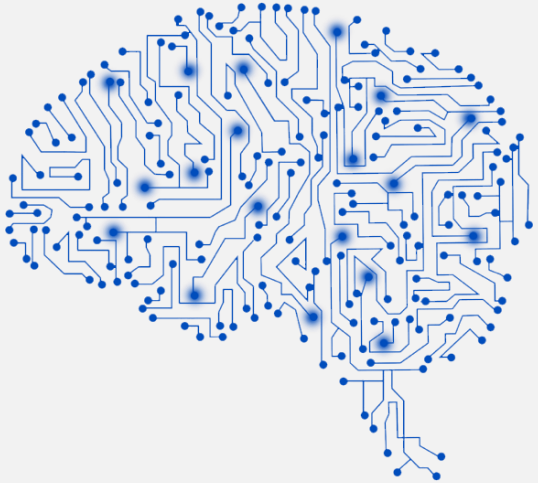


Deep Reinforcement Learning

Prof. Joongheon Kim
Korea University, Seoul, Korea
https://joongheon.github.io/
joongheon@korea.ac.kr

Lecture Roadmap

Introduction and Preliminaries

- **<u>Introduction and Applications</u>**
- Dynamic Programming
- Q-Learning and Markov Decision Process

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

Imitation Learning and Autonomous Driving

Deep Reinforcement Learning
Introduction and Preliminaries

- **Introduction and Applications**
- Dynamic Programming
- Q-Learning and Markov Decision Process

Deep Learning Revolution is Real

Geoffrey E Hinton

Yoshua Bengio

Yann LeCun

FATHERS OF THE DEEP LEARNING REVOLUTION RECEIVE ACM A.M. TURING AWARD

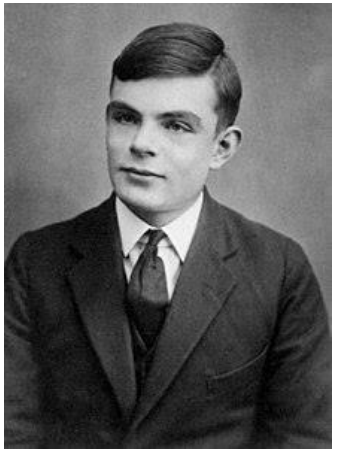
Bengio, Hinton, and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

ACM named Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. Bengio is Professor at the University of Montreal and Scientific Director at Mila, Quebec's Artificial Intelligence Institute; Hinton is VP and Engineering Fellow of Google, Chief Scientific Adviser of The Vector Institute, and University Professor Emeritus at the University of Toronto; and LeCun is Professor at New York University and VP and Chief AI Scientist at Facebook.

Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks. In recent years, deep learning methods have been responsible for astonishing breakthroughs in computer vision, speech recognition, natural language processing, and robotics—among other applications.

While the use of artificial neural networks as a tool to help computers recognize patterns and simulate human intelligence had been introduced in the 1980s, by the early 2000s, LeCun, Hinton and Bengio were among a small group who remained committed to this approach. Though their efforts to rekindle the AI community's interest in neural networks were initially met with skepticism, their ideas recently resulted in major technological advances, and their methodology is now the dominant paradigm in the field.

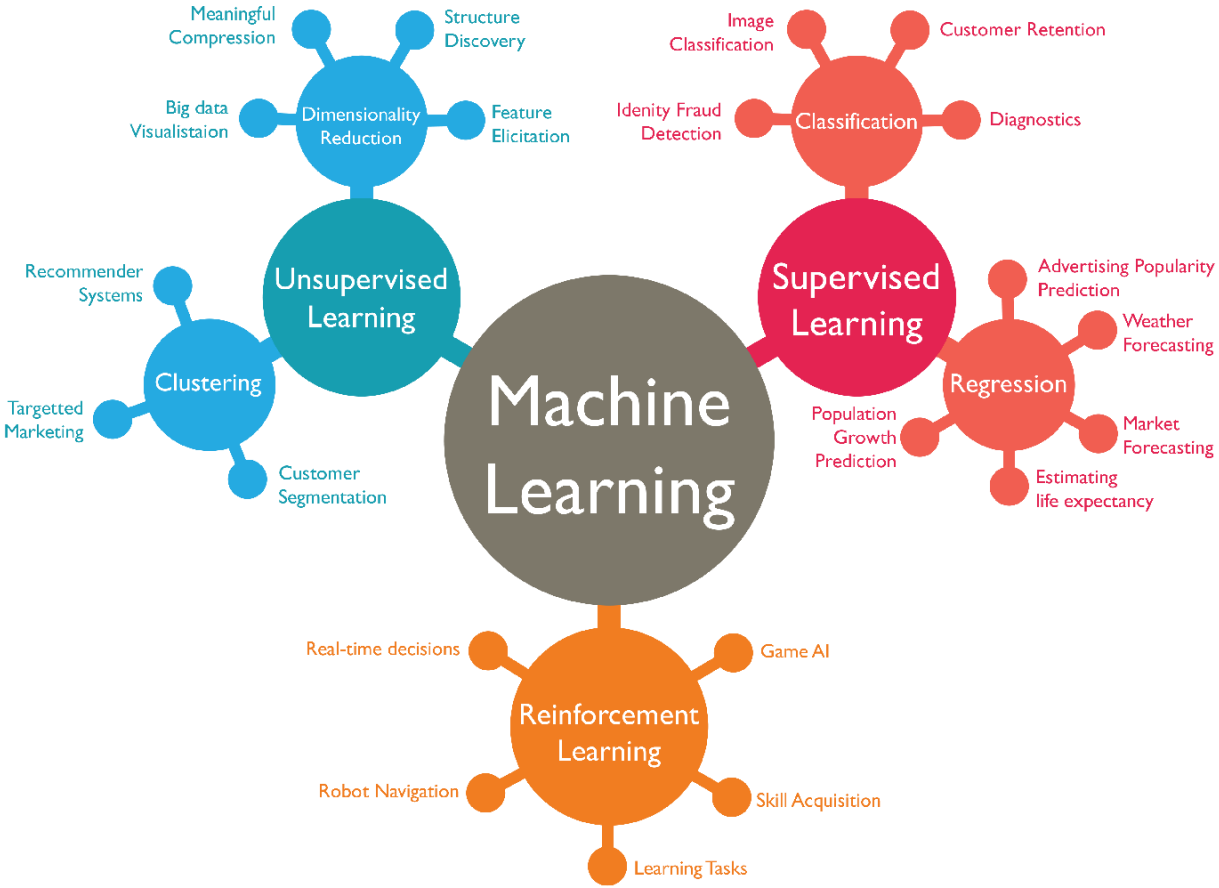
The ACM A.M. Turing Award, often referred to as the "Nobel Prize



Alan Turing (1912-1954)
Father of Computer Science

https://amturing.acm.org/

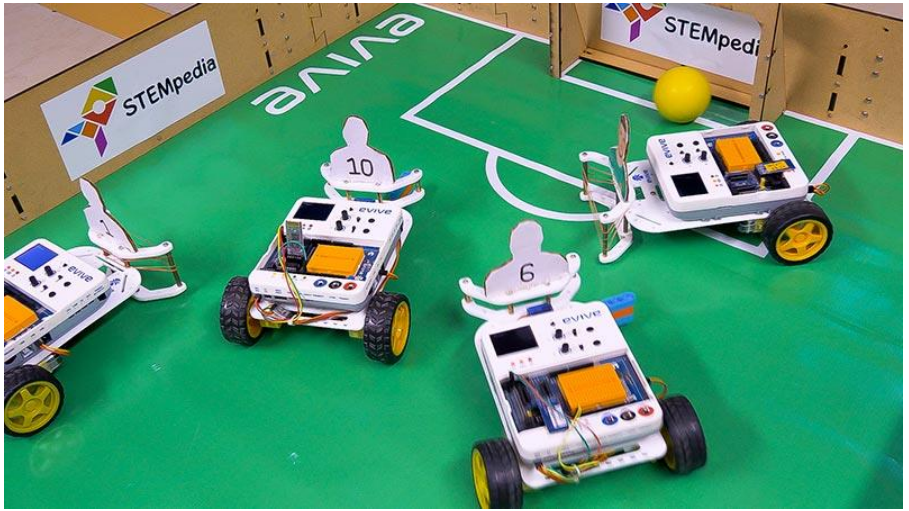
Introduction to RL



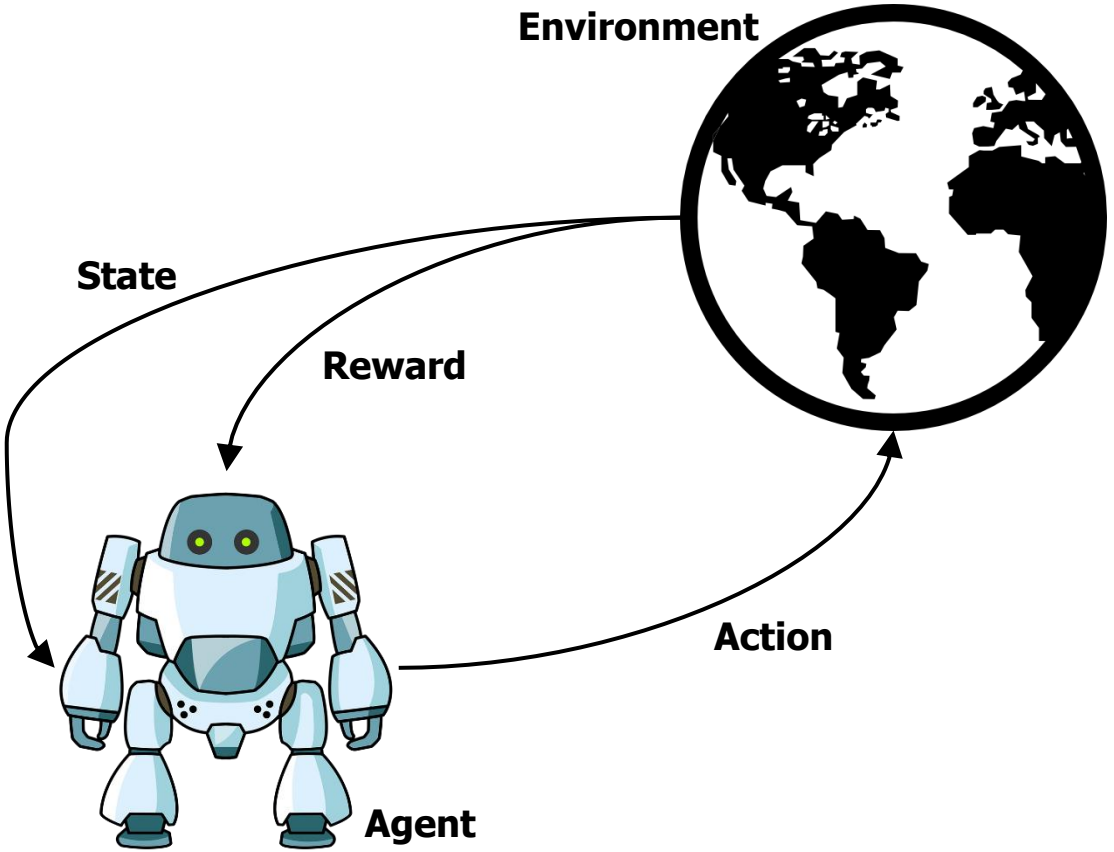
- Brief History and Successes
  - Minsky's PhD thesis (1954): Stochastic Neural-Analog **Reinforcement** Computer
  - Analogies with animal learning and psychology
  - Job-shop scheduling for NASA space missions (Zhang and Dietterich, 1997)
  - Robotic soccer (Stone and Veloso, 1998) – part of the world-champion approach

- When RL can be used?
  - Find the (approximated) **optimal action sequence** for **expected reward maximization (not for single optimal solution)**
  - Define **actions** and **rewards**. These are all we need to do.

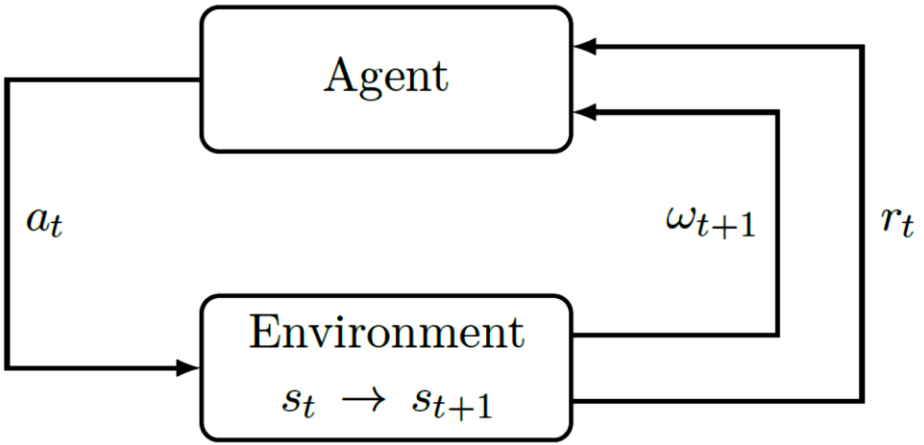
- Action Sequence (also called **Policy**, later in this presentation)!



Introduction to RL



- **RL Setting**
  - The general RL problem is formalized as a **discrete time stochastic control process** where **an agent interacts with its environment** as follows:
    1. The agent starts
       in a given state within its environment $s_0 \in S$
       by gathering an initial observation $\omega_0 \in \Omega$.
    2. At each time step t,
       The agent has to take an action $a_t \in A$.
       It follows three consequences:
       1) Obtains a reward $r_t \in R$
       2) State transitions to $s_{t+1} \in S$
       3) Obtains an observation $\omega_{t+1} \in \Omega$



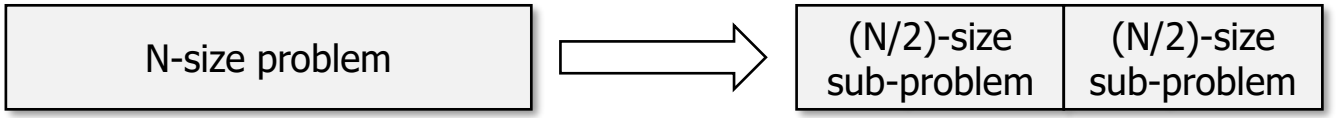
Deep Reinforcement Learning
Introduction and Preliminaries

- Introduction and Applications
- **Dynamic Programming**
- Q-Learning and Markov Decision Process

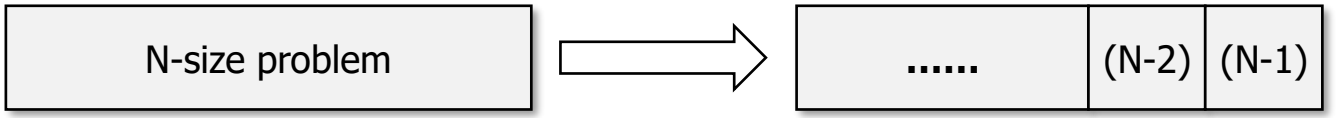
Outline

- **Introduction**
- Applications
  - Fibonacci Number
  - Pascal's Triangle
  - Knapsack Problem

- Dynamic Programming
  - The term “programming” stands for “planning”.
  - Usually used for optimization problems
  - In order to solve large-scale problems, (i) divide the problems into several sub-problems, (ii) solve the sub-problems, and (iii) obtain the solution of the original problem based on the solutions of the sub-problems, recursively
  - Difference from divide-and-conquer
    - Divide-and-conquer



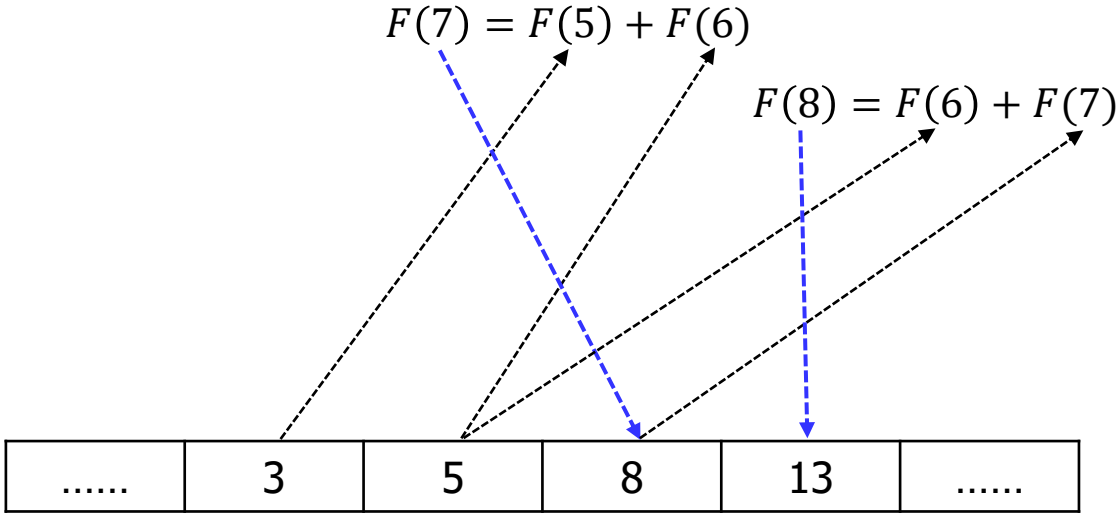
    - Dynamic programming



Outline

- Introduction
- Applications
  - **Fibonacci Number**
  - Pascal's Triangle
  - Knapsack Problem

- Fibonacci Number
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ……
  - $F(N) = F(N-2) + F(N-1)$ where $F(1) = 0$ and $F(2) = 1$ (Recursive!)



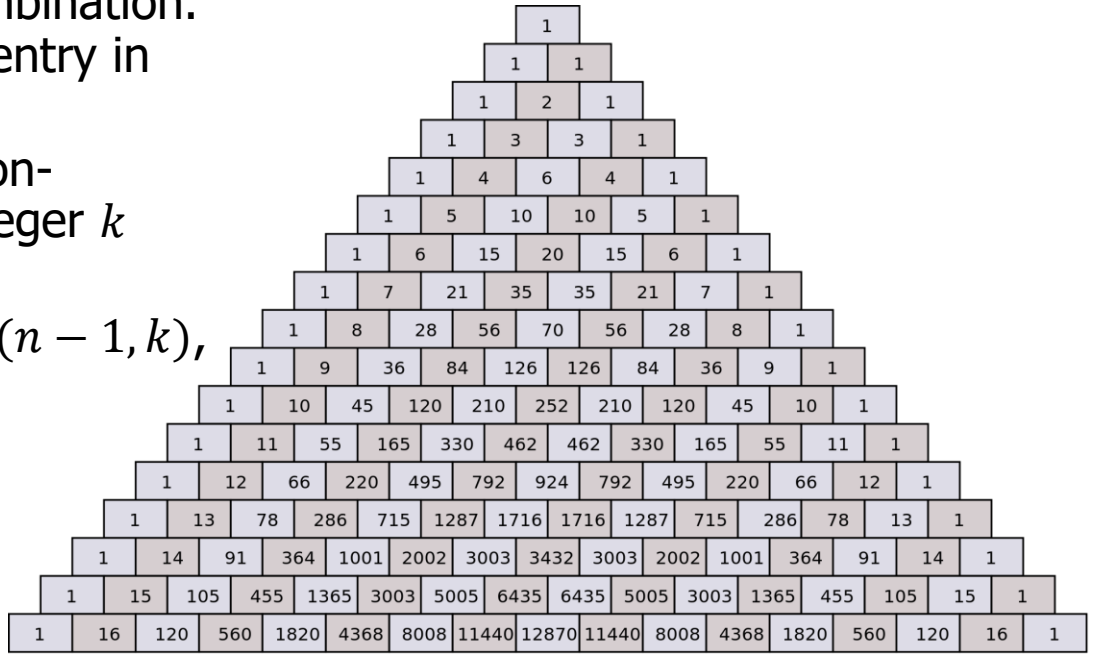
Outline

- Introduction
- Applications
  - Fibonacci Number
  - **Pascal's Triangle**
  - Knapsack Problem

- Pascal's Triangle
  - The entry in the n-th row and k-th column of Pascal's triangle is denoted $C(n,k)$ where C stands for combination. Note that the unique nonzero entry in the topmost row is $C(0,0) = 1$.
  - General Formulation for any non-negative integer n and any integer k between 0 and n:

    $$C(n,k) = C(n-1,k-1) + C(n-1,k),$$

  - Recursive!



Outline

- Introduction
- Applications
  - Fibonacci Number
  - Pascal's Triangle
  - **Knapsack Problem**

- Knapsack Problem
  - Suppose that we have N items ($I_i, i \in \{1, \cdots, N\}$) and each item has its own value (v_i for I_i where $i \in \{1, \cdots, N\}$) and weight (w_i for I_i where $i \in \{1, \cdots, N\}$). Now, we want to put items into knapsack where the capacity of knapsack is C. For the item allocation into the knapsack, we want to maximize the summation of values of allocated items.
  - Formulation:

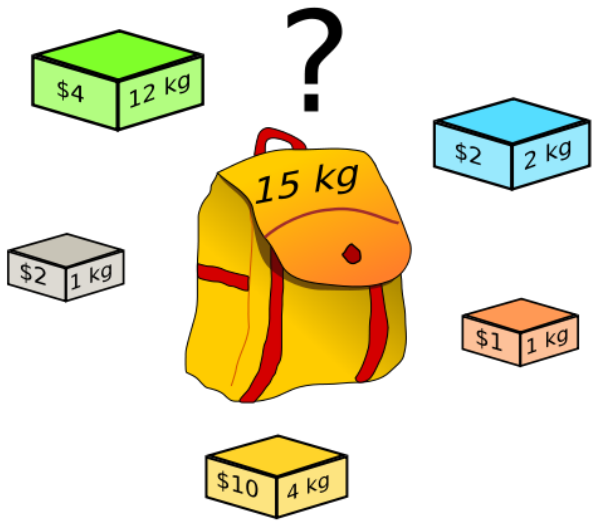
$$max: \sum_{i=1}^{N} v_i \cdot x_i$$

subject to

$$\sum_{i=1}^{N} w_i \cdot x_i \leq C$$

$$x_i \in \{0,1\}, i \in \{1, \cdots, N\}$$

and x_i is decision variable which defines the selection of I_i



- Formulation
  - **OPT(i,w):** Maximum sum value when
    - Items: $1, 2, 3, \cdots, i$
    - Residual capacity in the knapsack: w
  - Two possible cases
    - Case 1) When item i is NOT selected: **OPT(i,w) ← OPT($i-1$,w)**
    - Case 2) When item i is selected: **OPT(i,w) ← OPT($i-1$,$w-w_i$)+v_i**
      - v_i: The value of item i
      - w_i: The weight of item i
      - Note) This holds only when $w_i \leq w$.
  - Final Form

OPT(i,w) ← max{OPT($i-1$,w), OPT($i-1$,$w-w_i$)+v_i}
OPT(i,w) ← 0 // when $i = 0$

Application: Knapsack Problem

Product	A	B	C	D	E
Weight	3	4	7	8	9
Value	4	5	10	11	13

Product A Only

Capacity	3	4	5	6	7
Value	4	4	4	8	8
Product	A	A	A	AA	AA

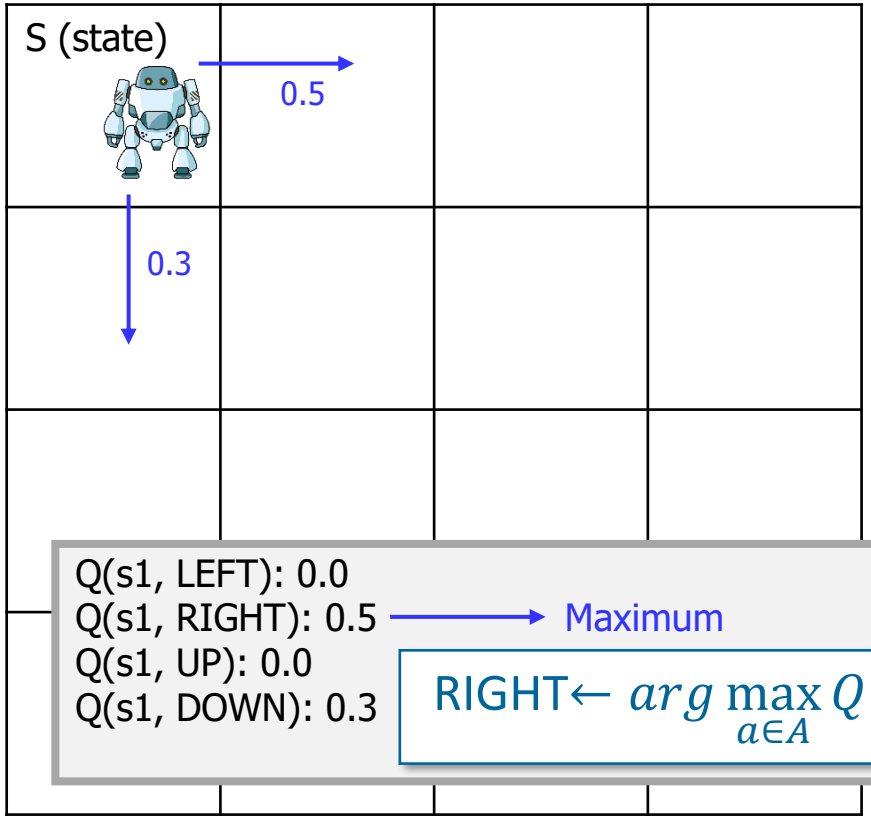
Product A and Product B

Capacity	3	4	5	6	7
Value	4	5	5	8	9
Product	A	B	B	AA	AB

Deep Reinforcement Learning
Introduction and Preliminaries

- Introduction and Applications
- Dynamic Programming
- **Q-Learning and Markov Decision Process**

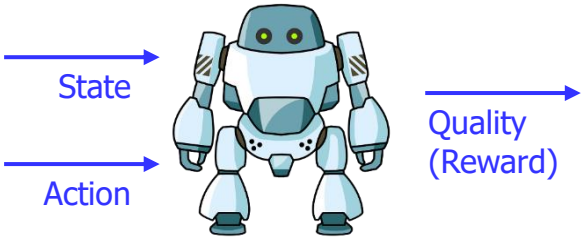
Q-Learning



Q(s1, LEFT): 0.0
Q(s1, RIGHT): 0.5 → Maximum
Q(s1, UP): 0.0
Q(s1, DOWN): 0.3

$$\text{RIGHT} \leftarrow arg\max_{a \in A} Q(s_1, a)$$

- Q-Function (State-action value)

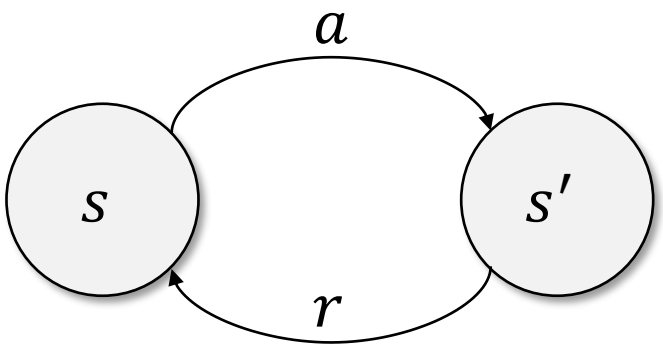


Q (state, action)

Optimal Policy π and Max Q

- Max Q = $\max_{a'} Q(s, a')$
- $\pi^*(s) = arg\max_{a} Q(s, a)$

- My condition
  - I am now in state s
  - When I do action a, I will go to s'.
  - When I do action a, I will get reward r
  - Q in s', it means $Q(s', a')$ exists.



- How can we express $Q(s, a)$ using $Q(s', a')$?

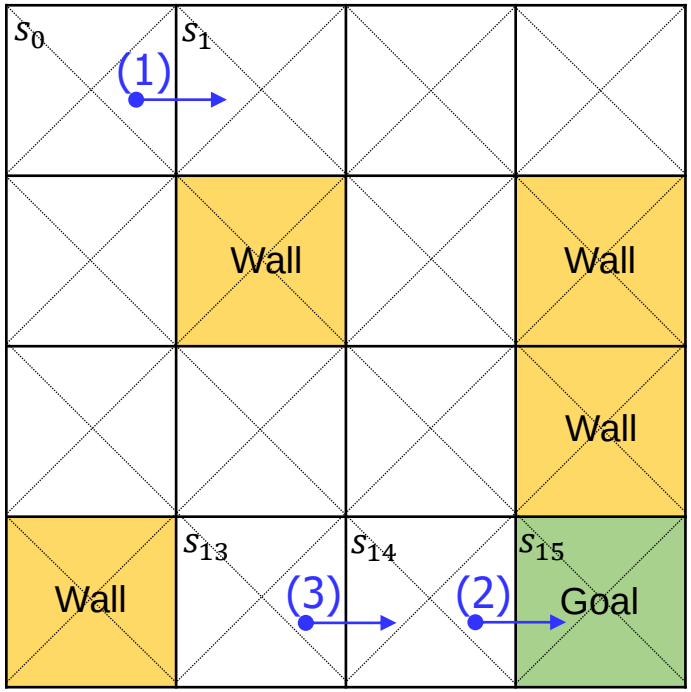
$$Q(s, a) = r + \max_{a'} Q(s', a')$$

Recurrence (e.g., factorial)

```
F(x){
        if (x != 1){ x * F(x-1) }
        if (x == 1){ F(x) = 1 }
        }
}
```

3! = F(3) = 3 * F(2)
= 3 * 2 * F(1)
= 3 * 2* 1 = 6

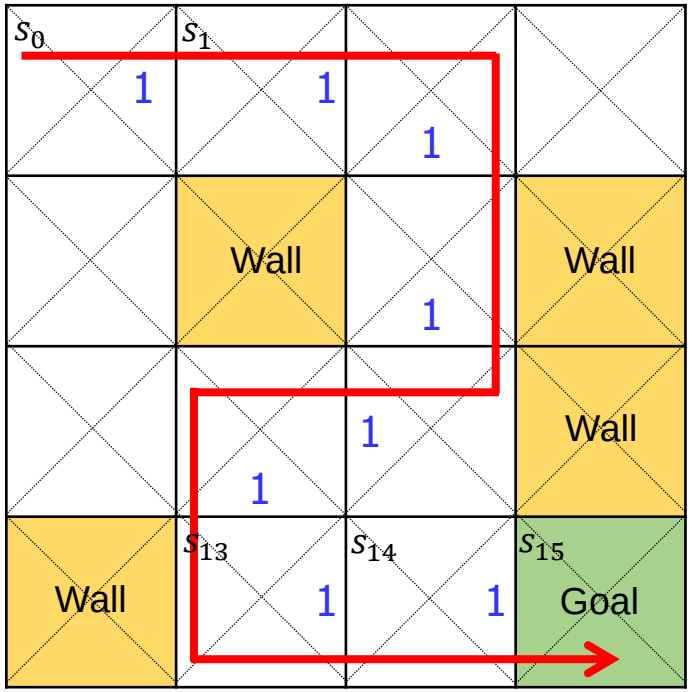
- 16 states and 4 actions (U, D, L, R)



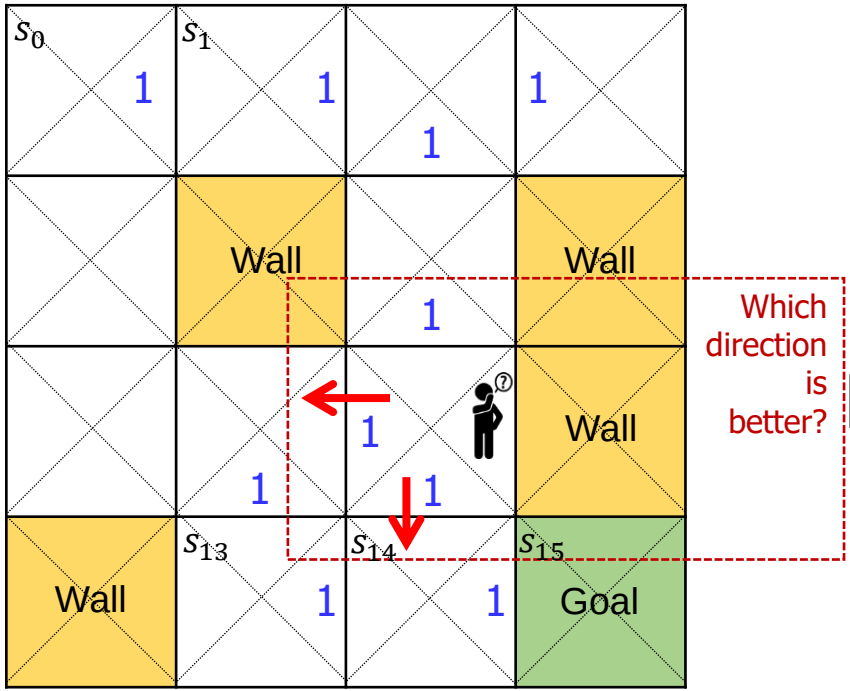
- Initial Status
  - All 64 Q values are 0,
  - Reward are all zero except $r_{s_{15},L} = 1$
- For (1), from s_0 to s_1
  - $Q(s_0, a_R) = r + \max_a Q(s_1, a) = 0 + \max\{0,0,0,0\} = 0$
- For (2), from s_{14} to s_{15} (goal)
  - $Q(s_{14}, a_R) = r + \max_a Q(s_{15}, a) = 1 + \max\{0,0,0,0\} = 1$
- For (3), from s_{13} to s_{14}
  - $Q(s_{13}, a_R) = r + \max_a Q(s_{14}, a) = 0 + \max\{0,0,1,0\} = 1$

Q-Learning

- 16 states and 4 actions (U, D, L, R)



- 16 states and 4 actions (U, D, L, R)



Which direction is better?

Learning $Q(s,a)$ with Discounted Reward

$$Q(s,a) = r + \gamma \cdot arg\max_a Q(s',a')$$

$$0 < \gamma \leq 1$$

Q-Learning

- For each s, a, initialize table entry $Q(s, a) \leftarrow 0$
- Observe current state s
- Do forever
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $Q(s, a)$ as follows:

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

  - $s \leftarrow s'$

Q-Learning with Exploit and Exploration: ε-Greedy

Finding the Best Restaurant

- Try the best one during weekdays.
- Try new ones during weekends.

ε-Greedy

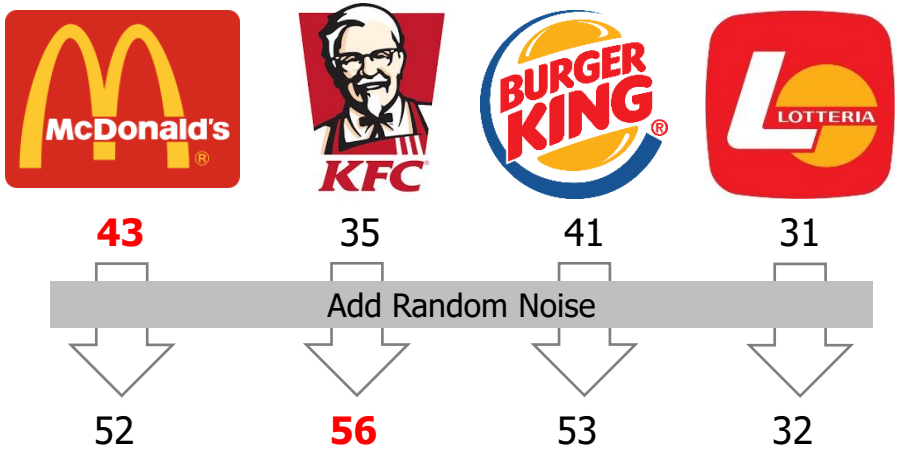
```
e=0.1
IF (random < e)
        a = random;
ELSE
        a = argmax(Q(s,a));
```

Decaying ε-Greedy

```
for i in range (1000); e=0.1 / (i+1);
IF (random < e)
        a = random;
ELSE
        a = argmax(Q(s,a));
```

Q-Learning with Exploit and Exploration: Add Random Noise

Finding the Best Restaurant



Add Random Noise

a = argmax(Q(s,a) + **random_values**);

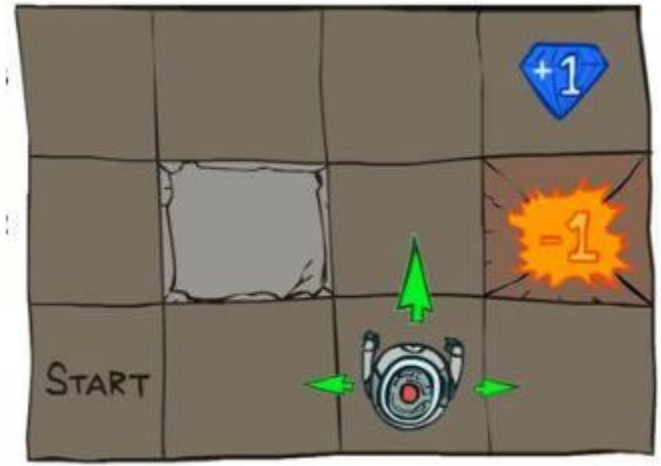
Add Decaying Random Noise

for i in range (1000);
a = argmax(Q(s,a) + random**/(i+1)**);

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - γ: Discount factor

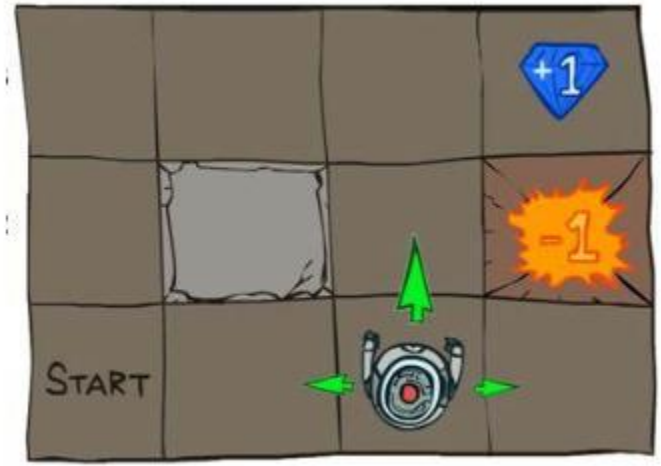
How can we use MDP to model agent in a maze?

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - **S: Set of states**
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - γ: Discount factor



S: location (x, y) if the maze is a 2D grid
- s_0: starting state
- s: current state
- s': next state
- s_t: state at time t

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - **A: Set of actions**
  - R: Reward function
  - T: Transition function
  - γ: Discount factor

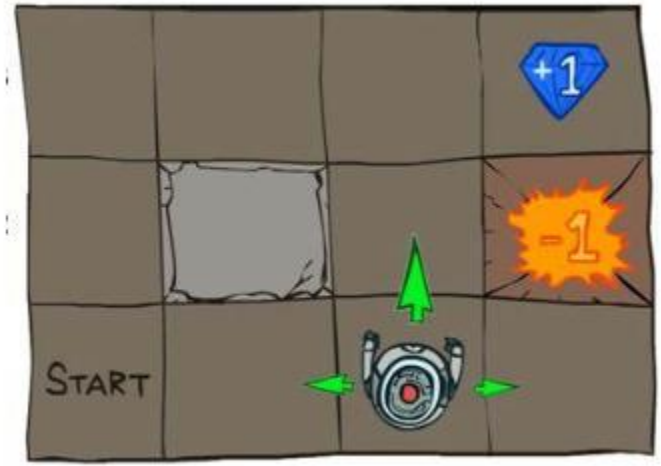


S: location (x, y) if the maze is a 2D grid

A: move up, down, left, or right

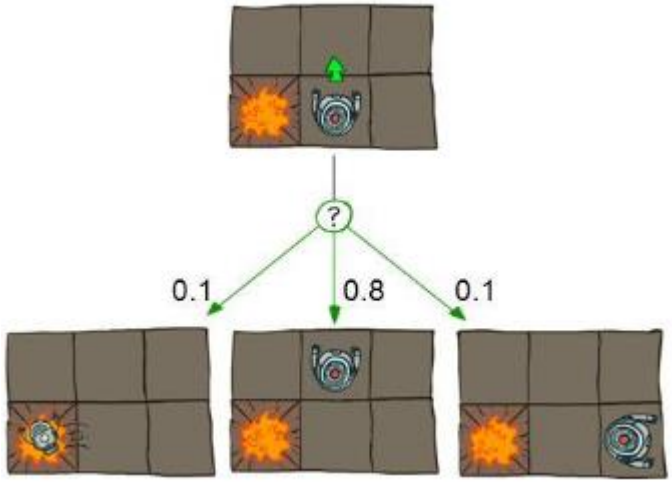
- $s \rightarrow s'$

- Markov Decision Process (MDP) Components: <S, A, R, T, γ>
  - S: Set of states
  - A: Set of actions
  - **R: Reward function**
  - T: Transition function
  - γ: Discount factor



S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
- $r = R(s, a, s')$
- -1 for moving (battery used)
- +1 for jewel? +100 for exit?

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - **T: Transition function**
  - γ: Discount factor



Stochastic Transition

S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
T: where is the robot's new location?
- $T = P(s'|s, a)$

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - **γ: Discount factor**

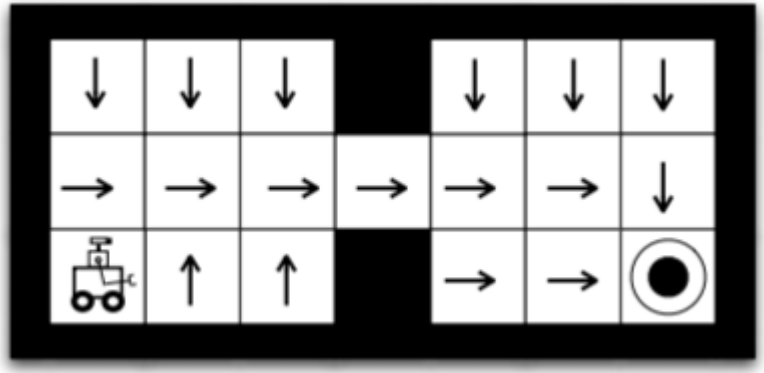
S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
T: where is the robot's new location?
γ: how much does future reward worth?

- $0 \leq \gamma \leq 1$, [$\gamma \approx 0$: future reward is near 0 (immediate action is preferred)]

Markov Decision Process (MDP)

- Policy
  - $\pi: S \rightarrow A$
  - Maps states to actions
  - Gives an action for every state
- Return
  -

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$



Our goal:
Find π that maximizes expected return!

- State Value Function (V)

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s\right)$$

  - Expected return of starting at state s and following policy π
  - How much return do I expect starting from state s?

- Action Value Function (Q)

$$Q^{\pi}(s,a) = E_{\pi}(R_t | s_t = s, a_t = a) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a\right)$$

  - Expected return of starting at state s, taking action a, and then following policy π
  - How much return do I expect starting from state s and taking action a?

- Our goal is to find the optimal policy

$$\pi^*(s) = \max_{\pi} R^{\pi}(s)$$

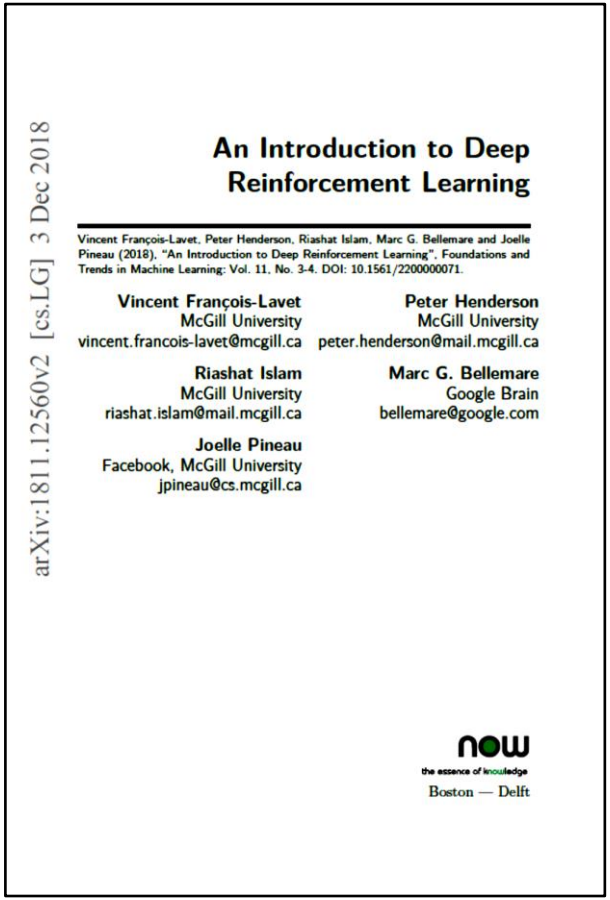
  - If $T(s'|s,a)$ and $R(s,a,s')$ are known, this is a **planning** problem.
  - We can use **dynamic programming** to find the optimal policy.

- Notes
  - Bellman Equation (Value Iteration)

$$\forall s \in S\colon V^*(s) = \max_a \sum_{s'} \{R(s,a,s') \cdot T(s,a,s') + \gamma V^*(s')\}$$

Outline

- **Intro to Reinforcement Learning**
  - **Prologue**
  - Formal Framework



An Introduction to Deep Reinforcement Learning

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), "An Introduction to Deep Reinforcement Learning", Foundations and Trends in Machine Learning: Vol. 11, No. 3-4. DOI: 10.1561/2200000071.

Vincent François-Lavet
McGill University
vincent.francois-lavet@mcgill.ca

Peter Henderson
McGill University
peter.henderson@mail.mcgill.ca

Riashat Islam
McGill University
riashat.islam@mail.mcgill.ca

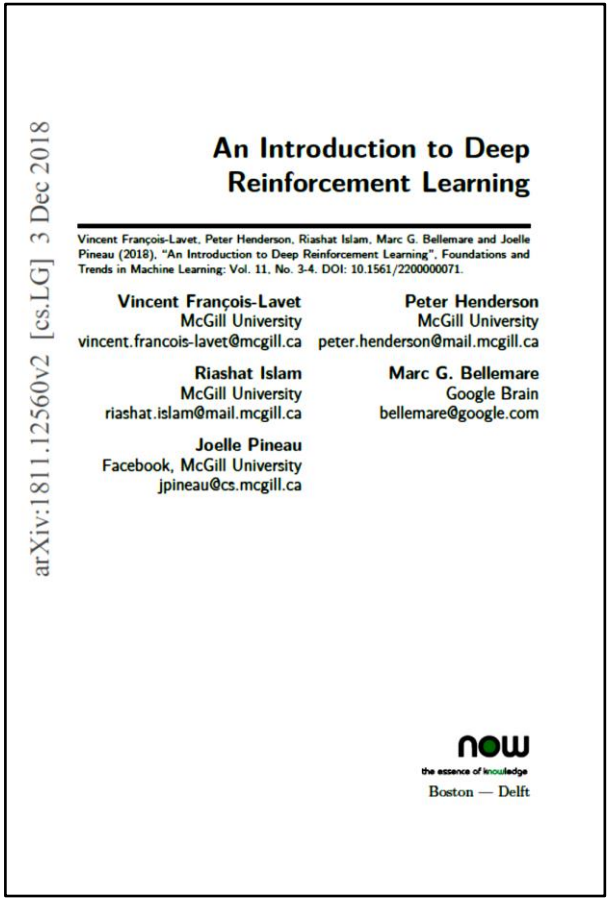
Marc G. Bellemare
Google Brain
bellemare@google.com

Joelle Pineau
Facebook, McGill University
jpineau@cs.mcgill.ca

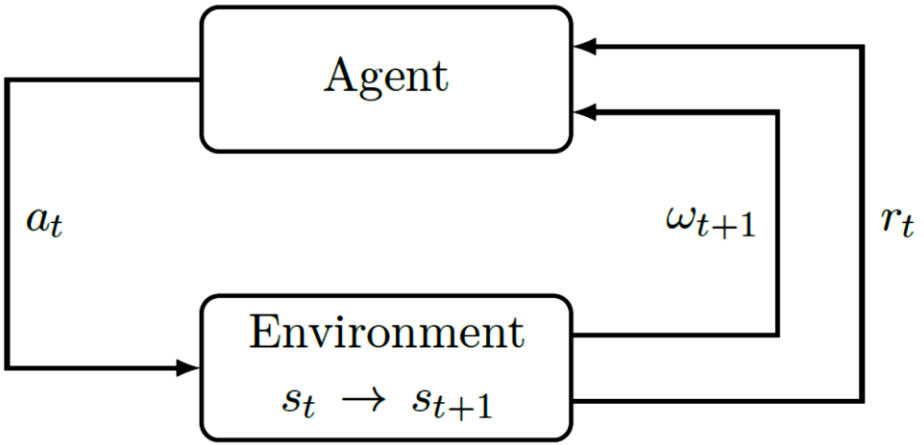
now
the essence of knowledge
Boston — Delft

- Reinforcement Learning (RL)
  - RL is the area of machine learning that deals with **sequential decision-making**.
  - RL problem can be formalized as an **agent** that has to make decisions in an environment to **optimize a given notion of cumulative rewards**.
  - Key aspects of RL
    - An agent **learns** a good behavior.
      - It modifies or acquires new behaviors and skills incrementally.
    - It uses trial-and-error **experience**.
      - An RL agent does not require complete knowledge or control of the environment.
      - It only needs to be able to interact with the environment and collect information.

- **Intro to Reinforcement Learning**
  - Prologue
  - **Formal Framework**



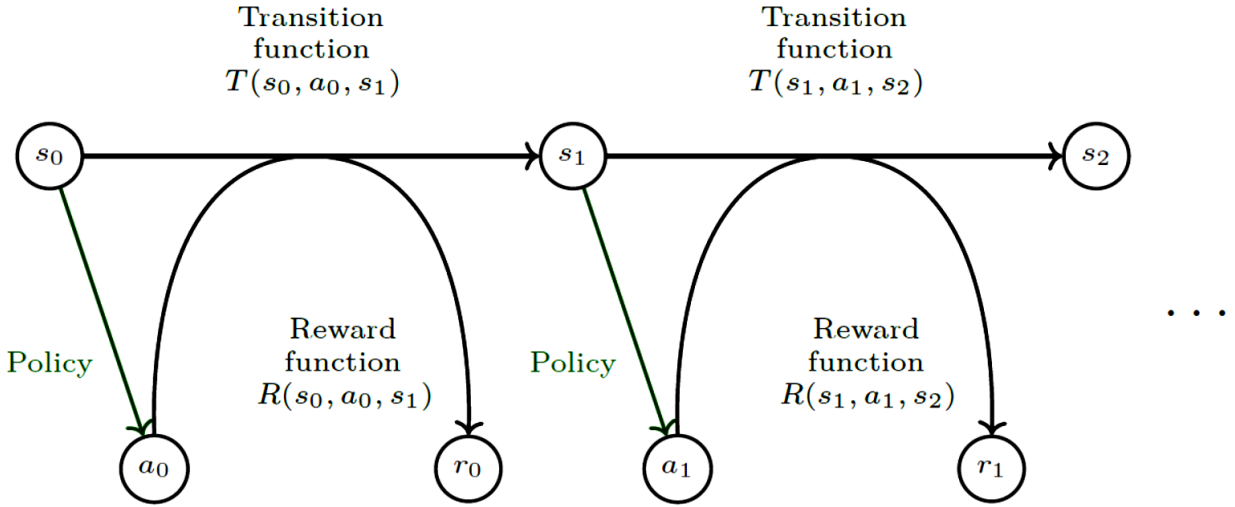
- **RL Setting**
  - The general RL problem is formalized as a **discrete time stochastic control process** where **an agent interacts with its environment** as follows:
    1. The agent starts
       in a given state within its environment $s_0 \in S$
       by gathering an initial observation $\omega_0 \in \Omega$.
    2. At each time step t,
       The agent has to take an action $a_t \in A$.
       It follows three consequences:
       1) Obtains a reward $r_t \in R$
       2) State transitions to $s_{t+1} \in S$
       3) Obtains an observation $\omega_{t+1} \in \Omega$



- **Markov Property**
  - [Definition (Markovian)] A discrete time stochastic control process is Markovian (i.e., it has the Markov property) if
    - $P(\omega_{t+1}|\omega_t, a_t) = P(\omega_{t+1}|\omega_t, a_t, \cdots, \omega_0, a_0)$, and
    - $P(r_t|\omega_t, a_t) = P(r_t|\omega_t, a_t, \cdots, \omega_0, a_0)$
  - The Markov property means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

- **Markov Property**
  - [Definition (MDP)] A Markov Decision Process (MDP) is a discrete time stochastic control process defined as follows. An MDP is a 5-tuple (S, A, T, R, γ) where:
    - S is the state space,
    - A is the action space,
    - $T: S \times A \times S \rightarrow [0,1]$ is the transition function (set of conditional transition probabilities between states),
    - $R: S \times A \times S \rightarrow R$ is the reward function, where R is a continuous set of possible rewards in a range $R_{\max} \in R^{+}$ (e.g., $[0, R_{\max}]$),
    - $\gamma \in [0,1)$ is the discount factor.

- **Markov Property**
  - The system in [Definition (MDP)] is fully observable in an MDP, which means that the observation is the same as the state of the environment: $\omega_t = s_t$.
  - At each time step t,
    - The probability of moving to s_{t+1} is given by the state transition function $T(s_t, a_t, s_{t+1})$ and the reward is given by a bounded reward function $R(s_t, a_t, s_{t+1}) \in R$.



- **Different Categories of Policies**
  - A policy defines how an agent selects actions.
  - Policies can also be categorized under a second criterion of being either deterministic or stochastic:
    - In the deterministic case, the policy is described by $\pi(s): S \rightarrow A$.
    - In the stochastic case, the policy is described by $\pi(s, a): S \times A \rightarrow [0,1]$ where $\pi(s, a)$ denotes the probability that action a may be chosen in state s.

- **Expected Return**
  - Basic Assumption: Consider the case of an RL agent whose goal is to find a policy $\pi(s,a) \in \Pi$, so as to optimize an **expected return** $V^{\pi}(s): S \rightarrow R$ (also called V-value function) such that

$$V^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \,|s_t = s, \pi\right]$$

    where
    - $r_t = E_{a\sim\pi(s_t,\cdot)}\{R(s_t, a, s_{t+1})\}$
    - $P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1})$ with $a\sim\pi(s_t,\cdot)$

  - From the definition of the expected return, the optimal expected return is as:

$$V^{*}(s) = \max_{\pi\in\Pi} V^{\pi}(s)$$

- **Expected Return**
  - In addition, the Q-value function $Q^{\pi}(s,a): S \times A \rightarrow R$ is defined as follows:

$$Q^{\pi}(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \,|s_t = s, a_t = a, \pi\right]$$

  - This can be rewritten recursively in the case of an MDP using Bellman's equation:

$$Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma Q^{\pi}(s', a = \pi(s'))\}$$

  - Similar to the V-value function, the optimal Q-value function $Q^{*}(s,a)$ is as:

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

- **Expected Return**
  - The **optimal policy** can be obtained directly from $Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a)$:

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