W06: An Introduction to Classical Reinforcement Learning

[Complexity-in-Action Research Lab]
Macquarie University
[Fred Amouzgar]
2021S1

ACKNOWLEDGEMENT

Awesome people who supported me and without them this presentation was not possible.

- **A/Prof Mark Dras (Primary Supervisor)**

- **Dr Rolf Shwitter**

- **Dr Malcolm Ryan (Co-supervisor)**

- **Prof Michael Richardson (Associate supervisor)**

AGENDA

- **Introduction**
  1. Machine Learning: The big picture
  2. Mind, Brain, and Intelligence
- **Introduction to Reinforcement Learning**
  1. A History of Reinforcement Learning
  2. Introduction to Reinforcement Learning Theory
  3. The Muti-Armed Bandit Problem (MABP)
  4. Trajectory (τ) and Return
- **Reinforcement Learning Theory**
  1. Markov Decision Making Problem (MDP): A mathematical model for RL
  2. Some Important Concepts
- **Two Model-Free and Classical Reinforcement Learning Algorithms**
  1. A Monte-Carlo Algorithm: On-policy First-visit Monte-Carlo
  2. Temporal-Difference Algorithms: SARSA, Q-Learning

Machine Learning:
The Big Picture

MACHINE LEARNING AND ANALYTICAL ALGORITHMS DICHOTOMY [11]

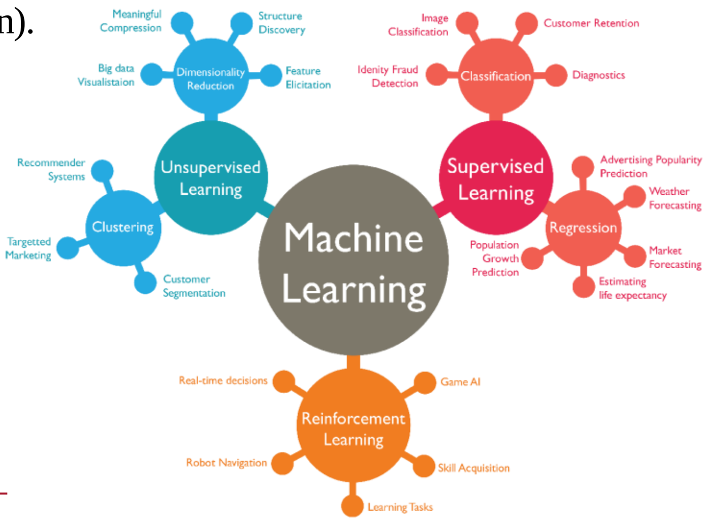
What are the essential differences between ML algorithms and the ones we studied in the Algorithm courses?

- **Analytical Algorithms**:
  1. **Analytical algorithms work like filters or catalyzer for the data**. They're generally built to process and **change the data**.
  2. Data does not change the structure or behavior of the algorithm.
  3. For instance, sorting a lot of arrays doesn't make bubble sort quick sort. Its behavior never changes.
- **Machine Learning Algorithms**:
  1. **ML algorithms work like a digestive system.** The goal is to **change/improve the algorithm using the data**. Once the digestion (training) is over, ML algorithms can act like an analytical one (unless the training is online and continuous).
  2. The data changes one or many data structures (e.g., matrices or vectors) in the algorithm. Those changes lead to behavior changes as well. e.g., a classifier categorizes better and more accurate as the algorithm receives more data.
  3. In classical and Tabular RL, the changing data structure is a table of values, and in approximate and deep RL, it's a linear model or a neural network.

MACHINE LEARNING THE BIG PICTURE

What are the differences?

- Supervised Learning (SL):
  - Learning from labelled data collected and verified by humans.
  - The goal is to **Generalize** (Classification and Regression).
- Unsupervised Learning (UL):
  - Learning from unlabelled data.
  - The goal is to **Compress** (Clustering, Dim. Reduction).
- Reinforcement Learning (RL):
  - Learning from trial and error and feedbacks.
  - No data (labelled or unlabelled) is available.
  - The goal is to **Act** (Topic of this lecture).

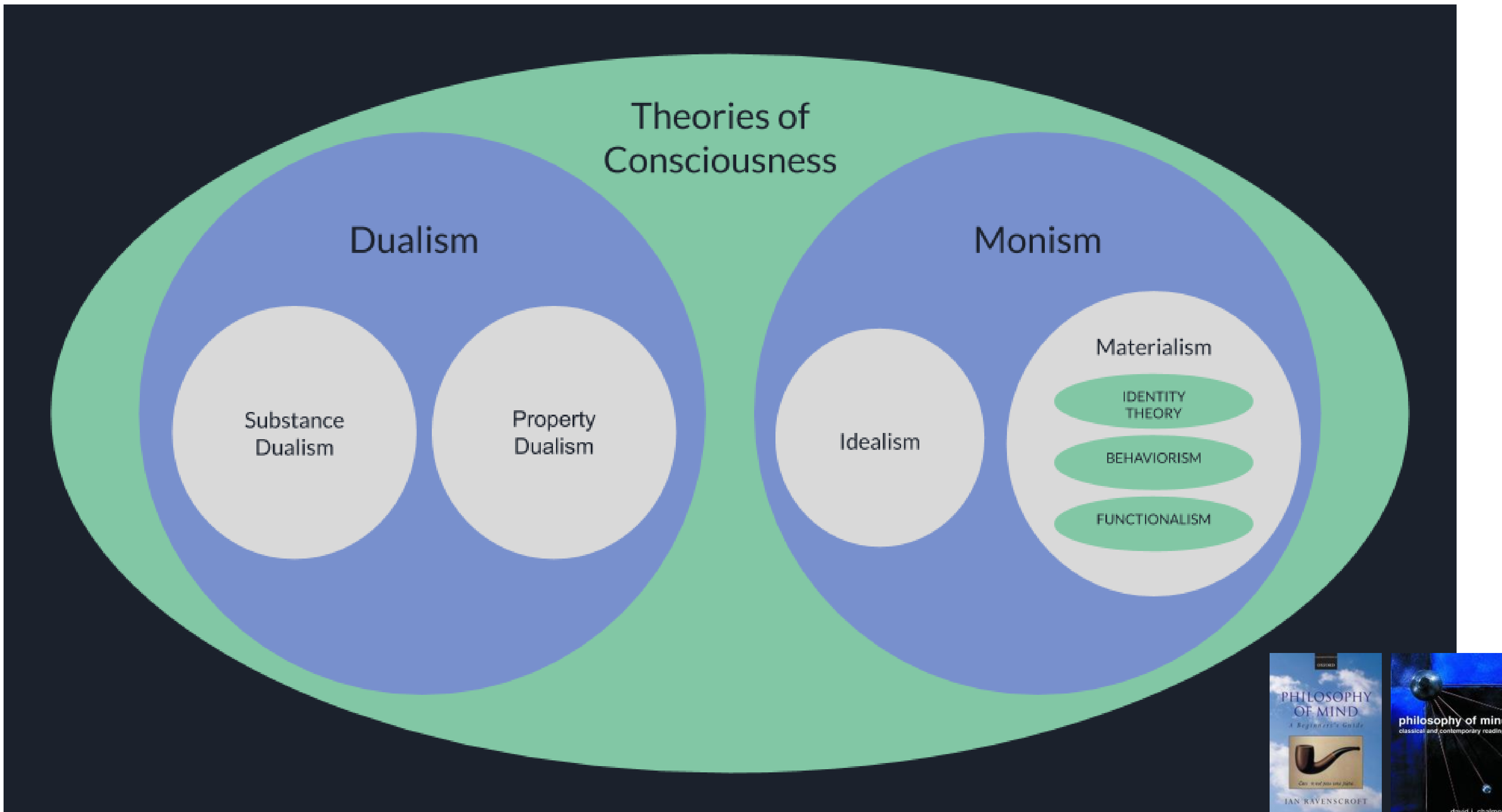


- Deep Reinforcement Learning:
  - Combining the generalization power of SL and acting powers of RL (week 12).
  - Deep learning = stronger methods for SL algorithms
- Imitation Learning:
  - Using datasets of expert's behavior and a SL algorithm to learn how to act (week 12).

Mind, Brain and Intelligence

PHILOSOPHY AND SCIENCE OF MIND

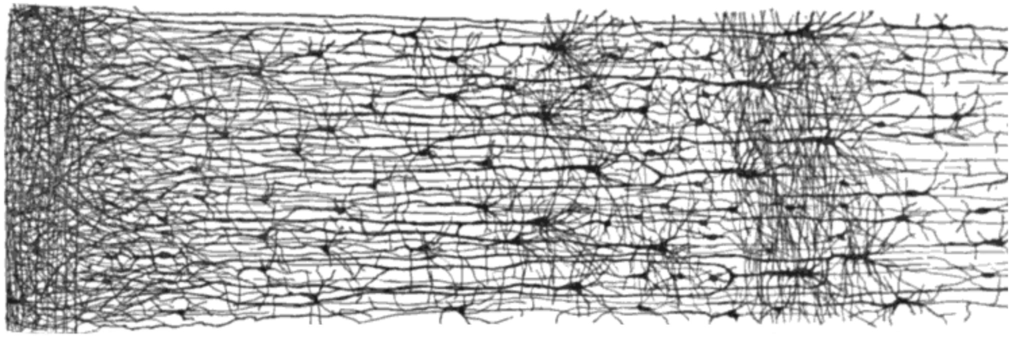
Different Philosophical positions on the nature of mind [1, 2, 3]:

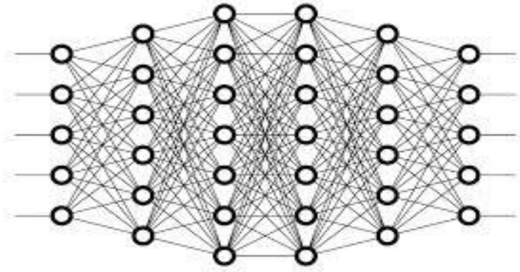


BRAIN AND INTELLIGENCE - 1

What is intelligence then and why we need a brain for it?

- Intelligence has been defined in many ways: the capacity for logic, understanding, self-awareness, learning, emotional knowledge, reasoning, planning, creativity, critical thinking, and problem-solving
- Intelligence is like sugar!
- But simply, Intelligence can be defined as the cognitive part of action [7].
- The common denominator: Efficient Information Processing + Storage ==> Decision Making
- Neural networks are great for this. Parallel, Modular, Multipurpose, etc.





BRAIN AND INTELLIGENCE - 2

A hypothesis:

- Hypothesis: brains, decision-making, and motor functions co-evolved.
- Decision making is usually regulated by an expensive organ: **The Brain (or other Central Nervous Systems such as Ganglia)**

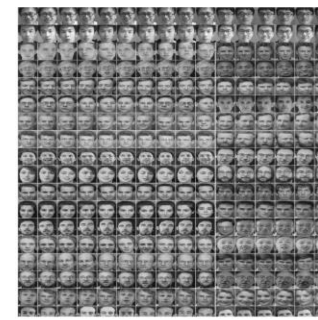
A support for our hypothesis [4]:

- This evolutionary step is still observable in primitive life forms. Meet the humble Sea Squirt.
    - It's born with a small brain, one eye, and a small tail.
    - It looks around and finds a solid rock to settle down.
    - Once it finds its rock, it attaches to it permanently, digests its tail, eye and brain!
    - Evolution hates redundancy. Use it or lose it!
    - This supports the hypothesis of close relation between the brain, decision making and motor functions.

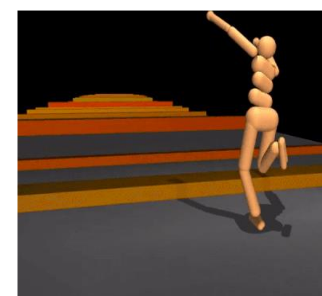
DIFFERENT METHODS FOR CREATING INTELLIGENCE [2]

- You can **Program it**; or "give a man a fish and he will eat for a day."
    - Biological processes: e.g., how to digest food, how to grow hair, etc.
    - Analytical Algorithms: e.g., how to send information to a website

- You can **Teach it**; or "teach a man to fish and he will eat for a lifetime."
    - Learning by observing: e.g., When an animal learns how to find food by observing others. Or learning in a classroom
    - Supervised Learning Algorithms: e.g., When a classifier is exposed to a dataset of faces and learns to identify all faces in images

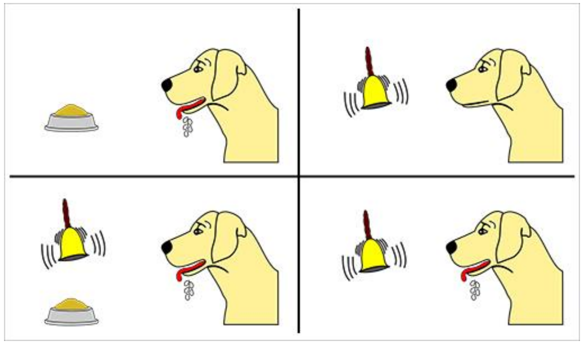


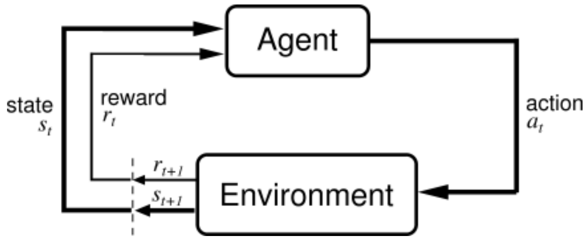
- You can **Provide the Motivation**; or "give a man a taste for fish and he'll figure out how to get fish, even if the details change!"
    - Adaptive Behavior among Animals and Humans: e.g., Learning to walk, speak, perform complex motor skills
    - Reinforcement Learning (RL): e.g. An agent learns how to play pong by itself.

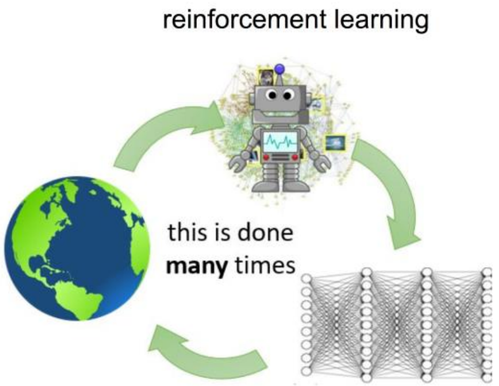


WHAT IS REINFORCEMENT LEARNING?

- **A computational approach to learning from interaction.**
- <u>What makes reinforcement learning different?</u>
    - There are no supervisors, only a reward/feedback signal (pain and pleasure)
    - Delayed feedback
    - The sequence of decisions matters
    - Current decisions and actions influence the later states and data we receive.
- Thus, Reinforcement Learning provides the formalism for intelligent and adaptive behavior.







A History of Reinforcement Learning and its Achievements

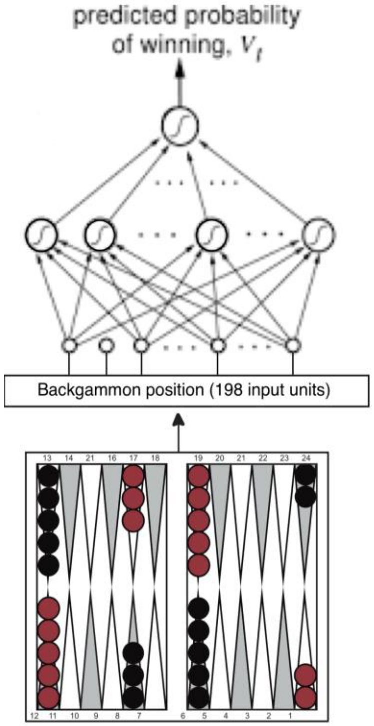
A BRIEF HISTORY OF RL ACHIEVEMENTS: GAMES – 1959: CHECKERS

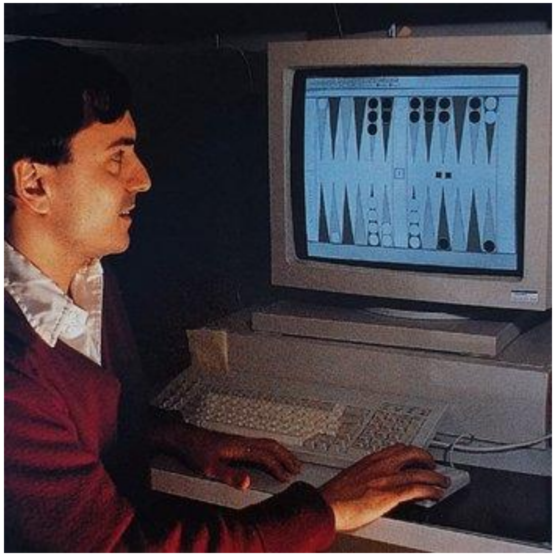
1959: Arthur Samuel's Checkers Player

- Samuel method was an assisted alpha-beta search which was using a scoring function based on the position of the board at any given time.
- The function tried to measure the chance of winning for each side.
- The scoring function was like the idea of a value function in RL.
- His discoveries led him to his unique definition of AI and Machine Learning. "ML as a field of study that gives computers the ability to learn without being explicitly programmed"

1994: Gerald Tesauro's Backgammon Player (TD-Gammon)

- One of the first successful examples of the combination of RL and multi-layer neural networks. The notion of self-play also applied here which was later used in other approaches.
- After six weeks of training, it was the best player of backgammon in the world!

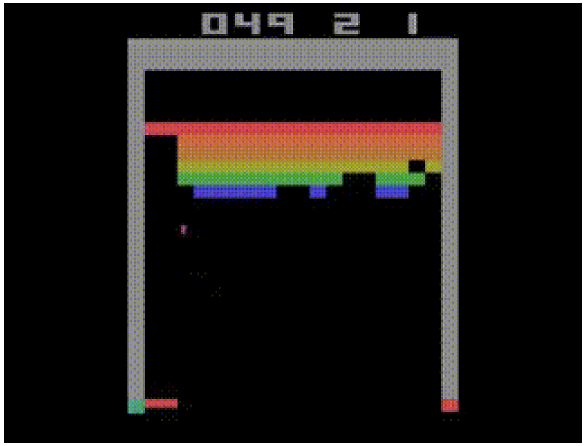


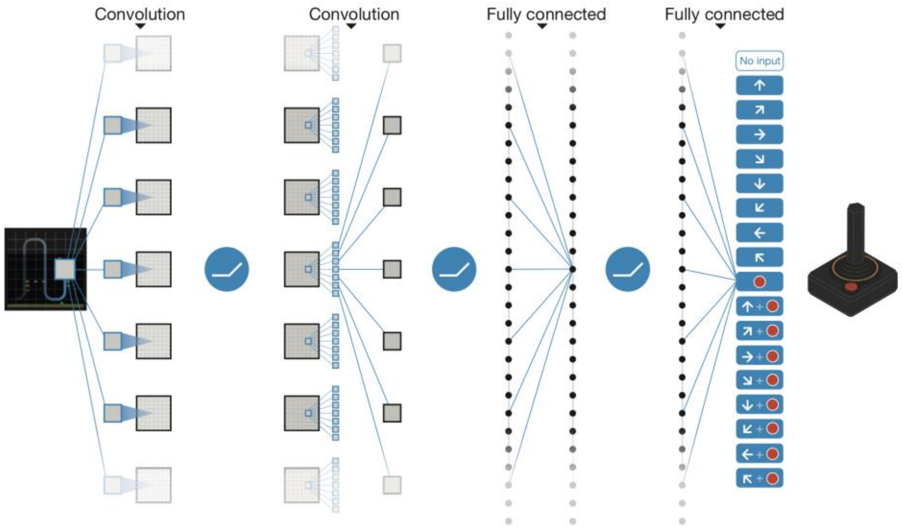


A BRIEF HISTORY OF RL ACHIEVEMENTS: GAMES – 2013-2015: ATARI

2013-2015: Atari Games (Deep Q-Network)

- Learning from raw pixels instead of handcrafted features
- Use of deep neural networks (convolutional neural networks, and fully-connected networks)
- Recurrent neural networks later used for capturing temporal patterns

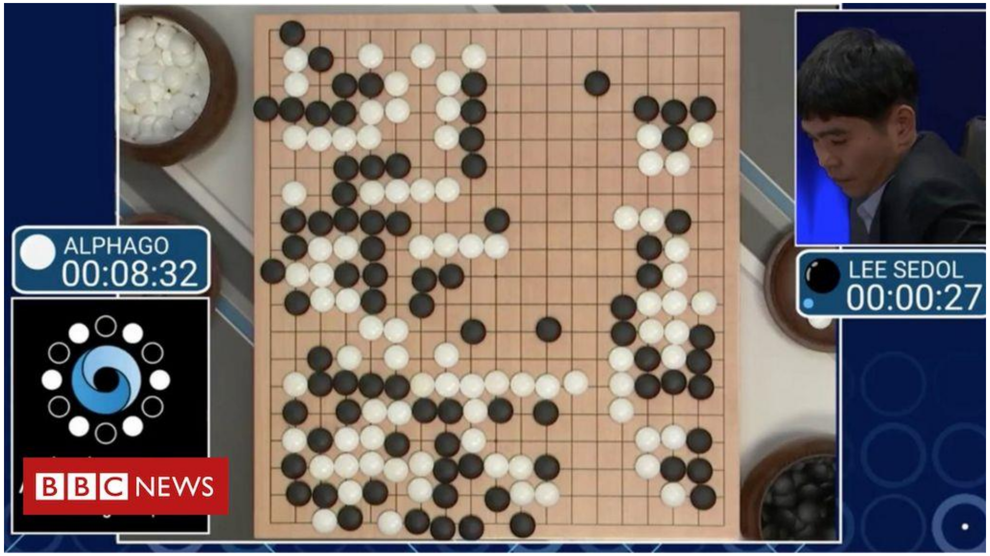




MACQUARIE University

2015-2016: The game of Go (AlphaGo, and AlphaGo Zero))

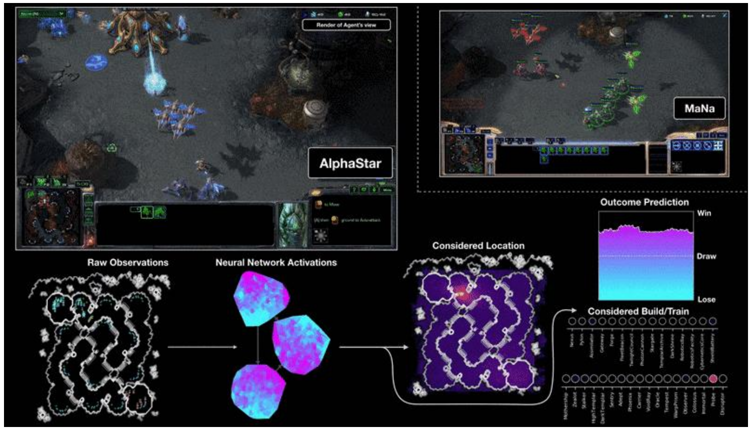
- Self-play agent.
- The use of Convolutional Neural Networks for feature extraction.
- Monte-Carlo Tree Search (MCTS) was used for searching the game tree and rollout.



A BRIEF HISTORY OF RL ACHIEVEMENTS: GAMES – 2017-2018: MULTIPLAYER VIDEO GAMES

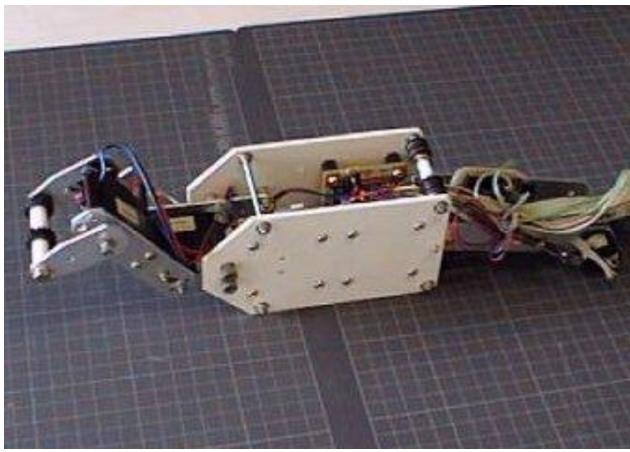
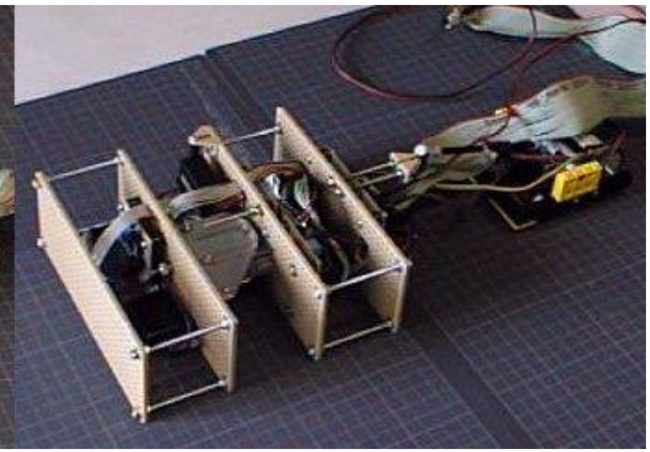
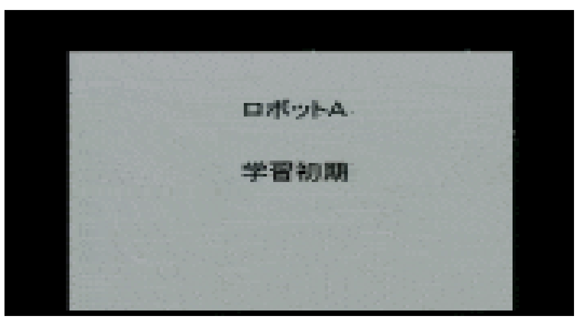
- **2017: Dota 2 (OpenAI Five)**

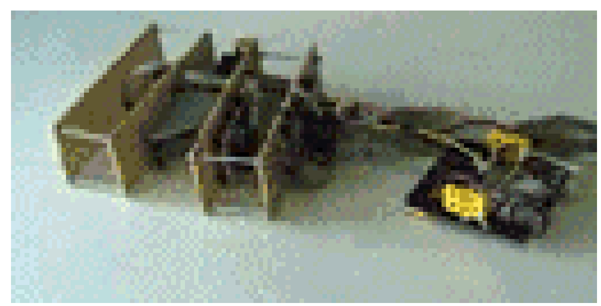
- **2018: StarCraft (DeepMind's AlphaStar)**



A BRIEF HISTORY OF RL ACHIEVEMENTS: ROBOTICS EARLY WORKS

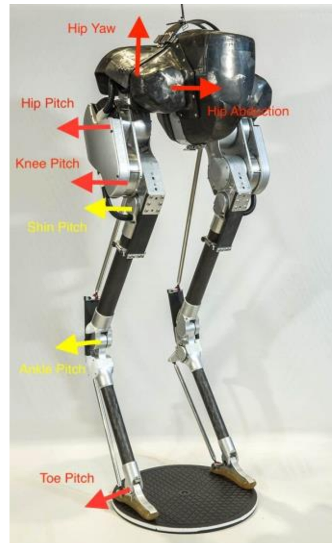
- **Robotics:** Hajime Kimura's RL Robots (1990s)



A BRIEF HISTORY OF RL ACHIEVEMENTS: MODERN ROBOTICS AND OTHER FIELDS

- **Robotics:**
  - Training a bipedal robot
  - Self-driving car (partly RL)

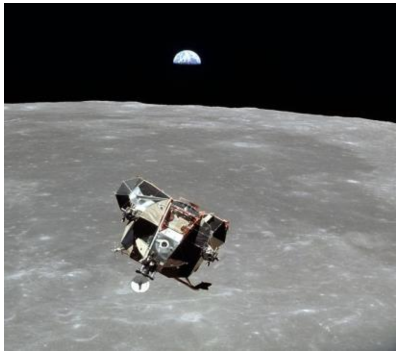


- **Controllers:**
  - Smart thermostats
  - Adaptive Air-conditioning
  - Auto-pilot systems
  - Advance/Cognitive Controlling Systems
  - ...

A BRIEF HISTORY OF REINFORCEMENT LEARNING (RL): COMMUNITIES INVOLVED

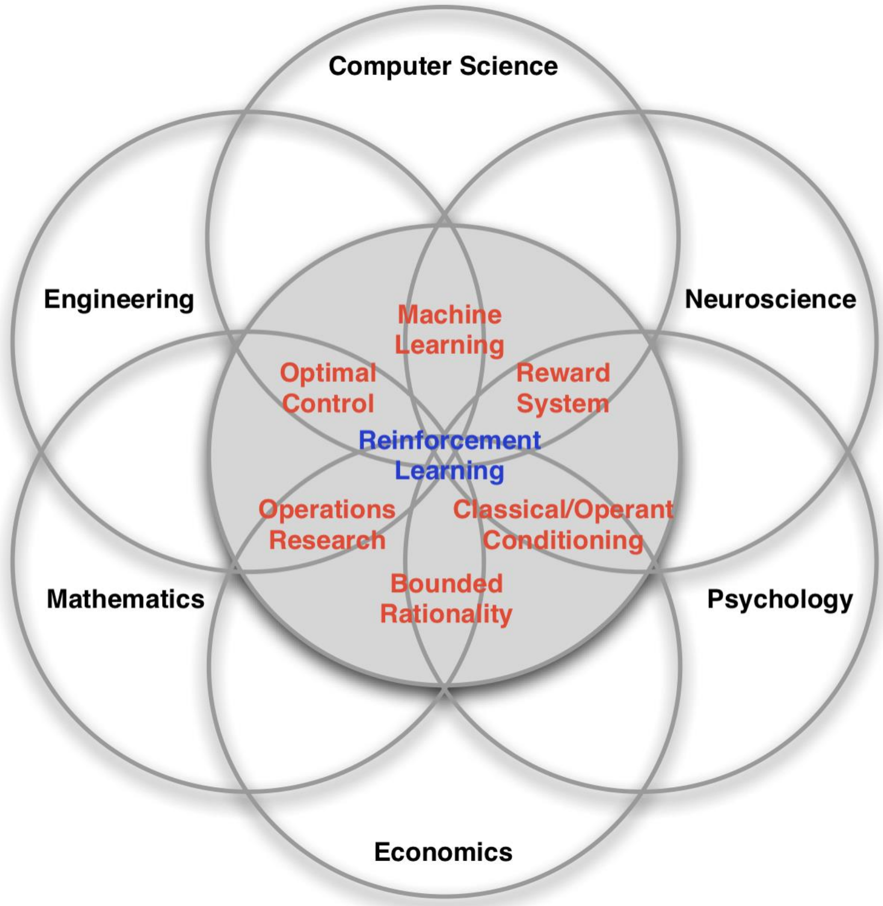
- Historically two major communities studied RL:
  - Psychology: For instance, in the psychology of animal learning as a form of learning by trial and error

  - Optimal Control: which tries to address the problem of designing a controller to minimize a measure of a dynamical system's behavior over time.



A BRIEF HISTORY OF REINFORCEMENT LEARNING (RL): COMMUNITIES INVOLVED

- RL; a multi-disciplinary topic [10]

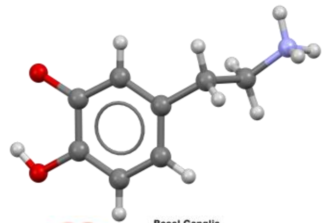


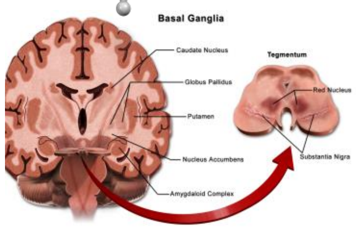
MACQUARIE University

Introduction to Reinforcement Learning Theory

ELEMENTS OF RL

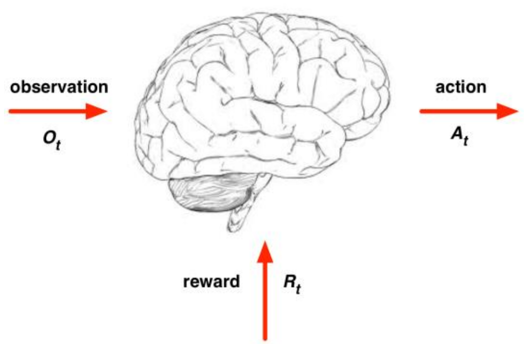
- Reward: A reward is a scalar feedback signal like +1/-1 (reward/punishment). It describes **how well the agent doing at a step**. Humans experience it as pleasure or pain regulated by **dopamine neurotransmitter** and brain structures such as **Basal ganglia**.

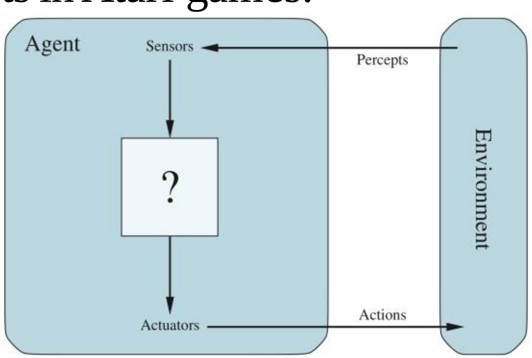




- State: A state (observation/percept) is a vector that represents **the state of the world the agent inhabits**. It can be a numerical vector, image (a matrix), multiple vectors, etc. It's tightly related to sensors in robots and living organisms.

- Action: An outward signal from agent to the world. It helps the agent to **modify its world**. For instance, the left or right movements in Atari games. Can be high or low level.





THE REWARD HYPOTHESIS

- In RL, agent's goal is to maximize the cumulative reward (like collecting a lot of "Good Jobs!").
- RL is based on **reward hypothesis** which considers rewards as a reliable source of guidance and assumes that **all goals can be formulated by the maximization of expected cumulative reward**.
- Unfortunately, poorly designed reward systems can mislead the agent, or they can be exploited.

SOME EXAMPLES OF REWARDS IN RL

Play with world's champion in Go:

- Winning the game +1 (at the end of the game)
- Losing the game –1!

Autopilot:

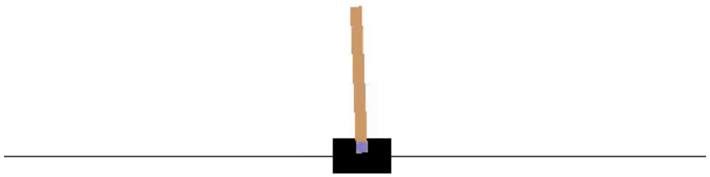
- Following the predefined path: +1 (in any time step)
- Causing or not preventing turbulence: -0.2
- Any detour: -1
- Crashing: -10

Control a building's air-conditioning system:

- keeping the desirable temperature: +1
- Going below or above the threshold: -1

Balancing a Cart Pole:

- Keeping the balance: +1
- Falling: 0 (receiving rewards stops)



SEQUENTIAL DECISION-MAKING PROBLEMS

- Goal: select actions that maximize total future reward.
- Main Difficulties:
  1. Actions have long-term consequences.
  2. Delayed Rewards (the credit assignment problem).
  3. It might be better to sacrifice immediate rewards to gain more long-term rewards.
  4. Exploration vs. Exploitation Dilemma

- Example:
  1. Education (may take years to finish, find a job, and earn money)
  2. A financial investment (may take months to mature)
  3. Facing the enemy in a video game (we may hit, get injured, or receive negative reward)

  - Exercise: Think of two more situations like the above and identify those difficulties.

EXPLORATION VS EXPLOITATION DILEMMA

- Online decision-making involves a fundamental choice:
  - ***<u>Exploitation</u>***: Make the best decision given current information
  - ***<u>Exploration</u>***: Gather more information (may help us to make better decisions later)
- Our goal is to find the best long-term strategy which involves short-term sacrifices which requires gathering "enough information."
- Examples:
  - Game Playing:
    - Exploitation: Play the move you believe is best
    - Exploration: Play an experimental move
  - Choosing a partner:
    - Exploitation: Continue dating your partner
    - Exploration: Date a new person
  - Restaurant selection:
    - Exploitation: Go to a favourite restaurant
    - Exploration: Try a new restaurant

EXPLORATION VS EXPLOITATION DILEMMA SOLUTIONS

MACQUARIE University

- Random exploration methods (Naïve exploration):
  - **ε-greedy**: Add noise to a greedy policy (act randomly with ε probability and greedy otherwise (1 - ε))
  - **Softmax (Boltzmann Exploration)**: forming a probability distribution over action space and sample from it.
  - **Gaussian noise**: simply adding noise from the Normal distribution to each action.

- Optimism in the face of uncertainty (prefer actions with uncertain values):
  - Optimistic initialization
  - UCB
  - Thompson sampling

- Information state space (consider the information gathered from a lookahead search to help the reward):
  - Gittins indices
  - Bayes-adaptive MDPs

MACQUARIE University

The Multi-Armed Bandit Problem (MABP):

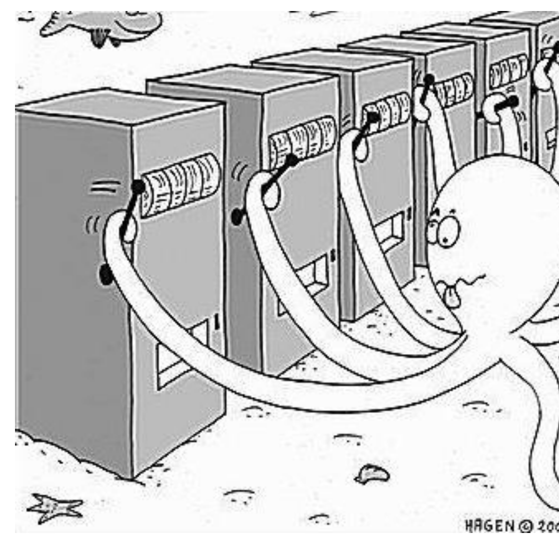
A platform for exploration

THE MULTI-ARMED BANDIT (MAB) [6]

MACQUARIE University

- An MAB is formally a tuple (A, R)
- A is a set of actions (levers)
- R is an unknown probability distributions over rewards $R^a(r) = P[r|a]$
- Process:
  1. At each step t the agent selects an action (pull a lever)
  2. The environment (multi-armed bandit) generates a reward $r_t \sim R^{a_t}$
- Goal: The goal is to maximize cumulative reward

$$maximize \sum_{\tau=1}^{t} r_\tau$$



ACTION-VALUE METHODS (MAB SOLUTION) [9]

- An action-value Q is defined as:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

- To simplify notation, we concentrate on a single action. Let Q_n denote the estimate of its action value after it had been selected n-1 times:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

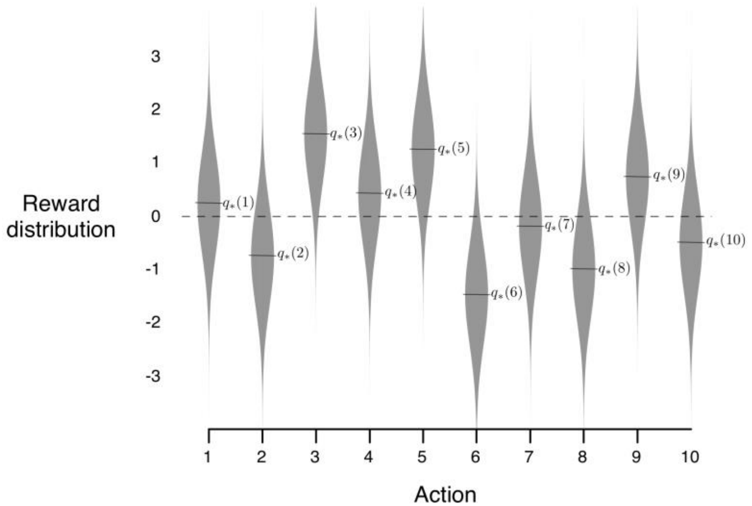
- Let's do some math and make it incremental:

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]
\end{aligned}$$

$$NewEstimate \leftarrow OldEstimate + StepSize\Big[Target - OldEstimate\Big]$$

A SIMPLE EPSILON-GREEDY BANDIT ALGORITHM [9]

- Consider the the 10-armed testbed:



- This ε-greedy algorithm can iteratively find the optimal action (lever):

A simple bandit algorithm

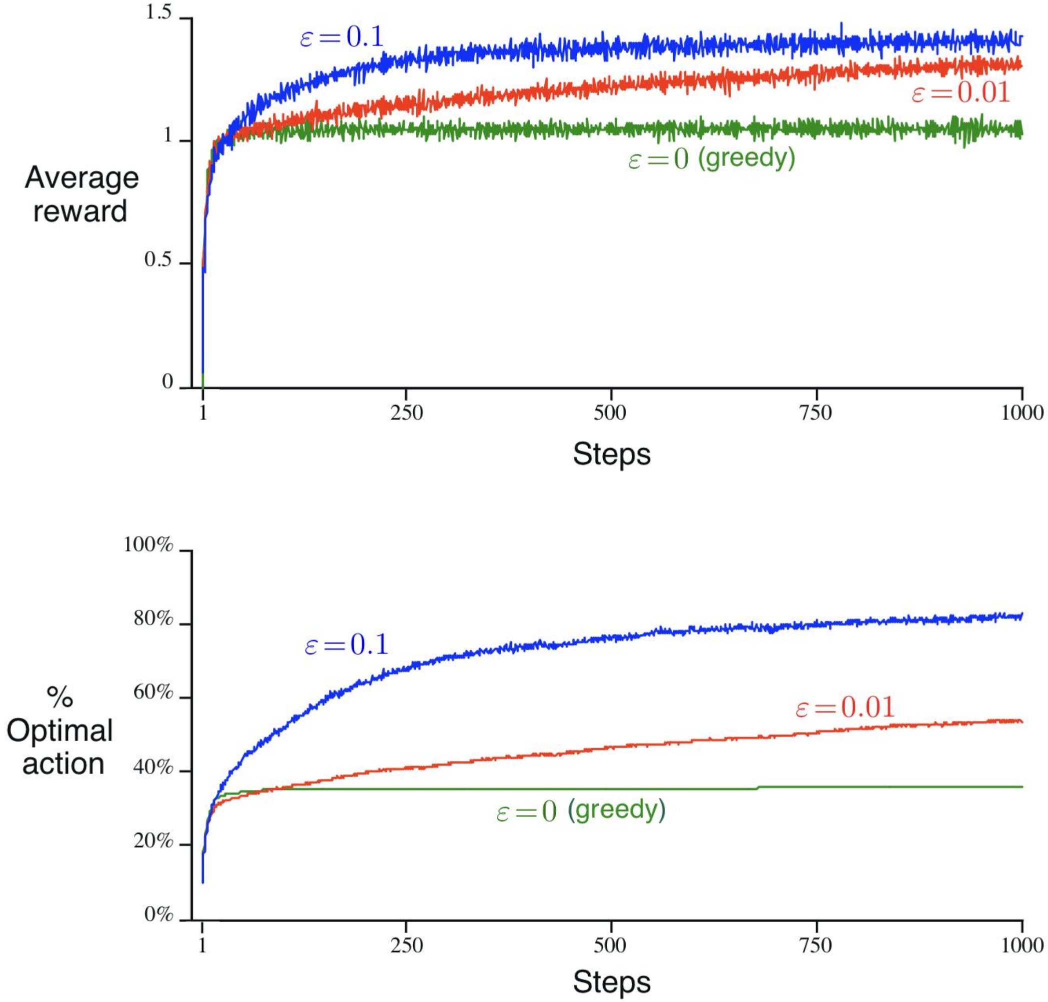
Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

A SIMPLE EPSILON-GREEDY BANDIT ALGORITHM RESULTS (DEMO THE BANDIT) [9]



Trajectory(τ) and Return

TRAJECTORY AND RETURN

- Trajectory: The life of an agent can be formally captured by a sequence of states, actions and rewards, $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ (terminal state).

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

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

  - The discount factor, γ (gamma), is a real number in [0, 1] and expresses the importance of future rewards.
  - This values immediate reward above delayed reward. In other words, the value of receiving reward R after k+1 time-step is $\gamma^k R$.
  - The extreme cases should be avoided. In most RL problems the gamma is set to 0.9 or 0.99.
    - $\gamma = 0$: Myopic evaluation (absolute hedonism)
    - $\gamma = 1$: Extreme far-sighted evaluation (the sage state)

RETURN AND A RECURSIVE REPRESENTATION

- We saw that a discounted return is $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

- The discounted return has interesting properties, and the most obvious one is its finitude.

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

- We can also write it in a recursive manner which is helpful in RL algorithms:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Markov Decision Process (MDP):

A mathematical model for reinforcement learning

MARKOV DECISION PROCESS (MDP)

- It introduces assumptions to deal with RL at the mathematical level. Assumptions which we suspend in practice.
  - Markov Property: the future is independent of the past given the present (all you need to know is here, and once the current state is known, the history can be discarded). Thus, a Markov state is defined as:

$$\mathbf{P}[S_{t+1}|S_t] = \mathbf{P}[S_{t+1}|S_1, S_2, ..., S_t]$$

  - Environment is fully observable

- Almost all RL problems can be formalized as MDP.
- A Markov Decision Process is a tuple <S, A, P, R, γ>
  - S is a finite set of states.
  - A is a finite set of actions.
  - P is a state transition probability matrix (the dynamic of the MDP, usually unknown)

$$P[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

  - R is a reward function $r(s, a, s') = E[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$
  - γ is a discount factor $\gamma \in [0, 1]$

MDP LOOP (DEMO RANDOM AGENT)

- MDP is also the way to program a training loop for the agent. The training loop is like a time-loop movie! Here's its general pattern [8]:

```
Given an env (environment) and an agent:
for episode = 0, ..., MAX_EPISODE do
    state = env.reset()
    agent.reset()
    for t = 0, ..., T do
        action = agent.act(state)
        state, reward = env.step(action)
        agent.update(action, state, reward)
        if env.done() then
            break
        end if
    end for
end for
```

POLICY AND VALUE FUNCTIONS

- **Policy**: A policy (π) is a function that maps the current state onto a set of probabilities for taking each action (also a distribution over a given s).

$$\pi : s \rightarrow p(a) \text{ also written as } \pi(a|s)$$

$$\sum_{a_t \in A(s_t)} \pi(a_t|s_t) = 1 \ , \ \pi(a|s) \geq 0$$

- The solution to an MDP is a policy that associates a decision with every state that the agent might reach.
- It's evident that the deterministic policy is a special case of a stochastic policy when the probability of one action is 1 and the rest is 0.

- **Value Functions**: If we provide a policy, value functions can predict reward in the future following that policy. Thus, the policy can be derived from them or they can guide the policy. There're two types of value functions:
  1. State-value function
  2. Action-value function

VALUE FUNCTIONS DEFINITIONS

- The state-value function V of an MDP is the expected return starting from state s, and then following policy π.

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

- The action-value function Q is the expected return starting from state s, taking action a, and then following policy π.

$$Q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

BELLMAN EXPECTATION EQUATION

- The state-value function can be decomposed into immediate reward plus discounted value of successor state (sometimes called **bootstrapping**),

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S} \end{aligned}$$

- The action-value function can similarly be decomposed,

$$Q_\pi(s, a) = E_\pi[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

- The main reason is a similar decomposition pattern in the return definition:

$$G_t = R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+1} + \ldots = R_{t+1} + \gamma G_{t+1}$$

BELLMAN OPTIMALITY EQUATION

- By maximizing these value functions, we get the Bellman optimality equations that are used in RL algorithms.

- Optimal V:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r \mid s,a) \big[r + \gamma v_*(s') \big]. \end{aligned}$$

- Optimal Q:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \Bigm| S_t = s, A_t = a \Big] \\ &= \sum_{s',r} p(s',r \mid s,a) \Big[r + \gamma \max_{a'} q_*(s',a') \Big]. \end{aligned}$$

POLICY AND VALUE FUNCTION RELATION

- We can mathematically prove:

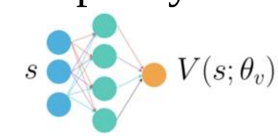
$$if\ \ V_{\pi_1}(s) \geq V_{\pi_2}(s)\ \ then\ \ \pi_1 \geq \pi_2\ \ \forall s \in S$$

- This means that finding a better value function will lead to a better policy.
- The optimal value V function is unique, the optimal policy isn't necessarily unique.

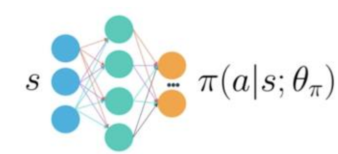
Some Important Concepts

DIFFERENT TYPES OF RL ALGORITHMS: POLICY, VALUE OR MODEL-BASED

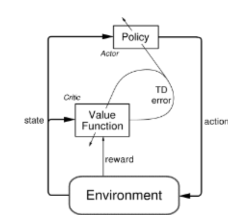
1. Model-based: The algorithm is required to have or will generate a model of the world for successful execution
    1. Dynamic Programming (Sutton's book [9] chapter 4)
    2. Dyna (Sutton's book [9] chapter 9)
2. Value-based: these methods are usually model-free. They first update the value of the states-action iteratively, then use those values to approximate the optimal policy.
    1. Q-learning and Deep Q-Network (DQN) (will be discussed in week 12)
    2. Sarsa



3. Policy-based (policy-search): they by-pass the value function and directly connect states to actions. Usually, we give the state to the neural network, and it tells us the desirable action (week 12).
    1. REINFORCE
    2. PPO



- Actor-Critic: Two neural networks work together in learning the task. One is generating the action (policy-based), the other evaluates the quality of the generated action (value-based) which leads to a faster convergence. It's postulated that a similar method used by the brain (week 12).
    1. DDPG
    2. A3C



DIFFERENT TYPES OF RL ALGORITHMS: ON-POLICY & OFF-POLICY

- **On-Policy Algorithms**:
  - The distinguishing feature of on-policy methods is that they estimate the value of a policy while using it for control.
  - It's like on-the-job training.
  - Some algorithms: On-Policy Monte-Carlo, SARSA, REINFORCE (week 12).

- **Off-Policy Algorithms**:
  - They follow a behavior policy but update a target policy.
  - The behavior and target policy might be related or completely decoupled.
  - The decoupling factor facilitates an asynchronous architecture, meaning we can collect data from multiple concurrent behavior policies, and update the target with all of them.
  - Notice that if we set behavior=target, it simply collapses to an on-policy algorithm.
  - Off-policy methods are more data efficient, but harder to implement.
  - Imitation learning is a kind of off-policy method. Human=Behavior π, Agent=Target π
  - Some algorithms: Q-learning, DQN (week 12)

A Monte-Carlo Algorithm

MODEL-FREE RL: MONTE-CARLO (MC) RL [6]: SOME FACTS

- MC usually refers to any method with a random component/sampling.
- In RL the randomness is the returns and their corresponding trajectories.
- MC methods learn directly from the episodes of experience.
- MC is model-free: no knowledge of MDP transition/rewards needed.
- MC learns from complete episodes (the entire trajectory): No bootstrapping
- MC uses the simplest possible idea: value = mean return
- Limitation: MC can only solve episodic problems (all episodes must terminate)

MODEL-FREE RL: MONTE-CARLO (MC) RL: THE METHOD

- The agent plays a bunch of episodes with an ε-based policy (ε-greedy or ε-soft), save the states, actions, and rewards.

$$s_0, a_0, r_1, s_1, a_1, \ldots, s_T$$

- Then, after each episode, we set the return to 0 (G=0), go backwards and calculate the return for each state, and action.

  G(t) = r(t+1) + gamma * G(t+1)

- Very helpful if this is done in the reverse order.

- Once we have (s, a, G), we update the Q(s, a).

MODEL-FREE RL: MONTE-CARLO (MC) RL: THE ALGORITHM

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\quad \pi \leftarrow$ an arbitrary ε-soft policy

$\quad Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

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

Repeat forever (for each episode):

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

$\quad G \leftarrow 0$

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

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

$\quad\quad$ Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

$\quad\quad\quad Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\,[G - Q(S_t, A_t)]$

$\quad\quad\quad A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

$\quad\quad\quad$ For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

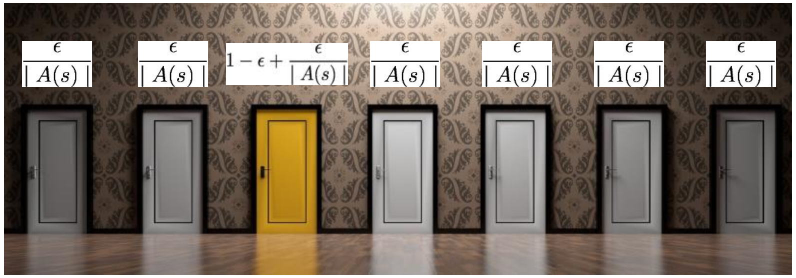
MODEL-FREE RL: MONTE-CARLO (MC) RL: THE EPSILON-SOFT AND EPSILON-GREEDY

- The exploration used in the on-policy MC algorithm is called ε-soft.
- Like ε-greedy, we want each action has at least a minimum chance of being chosen. The lower bound for every action to be selected:

$$\pi(a|s) \geq \frac{\epsilon}{|\ A(s)\ |}, \forall a \in A(s)$$

- We use ε to decide how much we want to explore. According to this formula, when ε is small the optimal action has a higher chance of being selected:

$$\begin{aligned} a_* &= argmax_a Q(s,a) \\ \pi(a|s) &= 1 - \epsilon + \frac{\epsilon}{|\ A(s)\ |} \quad if\ a = a_* \\ &= \frac{\epsilon}{|\ A(s)\ |} \quad if\ a \neq a_* \end{aligned}$$



- In practice, the result is identical to using ε-greedy. **Notice that the optimal action can be chosen as the optimal action or as a random action**. Hence, the $+\frac{\epsilon}{|\ A(s)\ |}$.

Temporal-Difference Algorithms: SARSA, Q-Learning

MODEL-FREE RL: TEMPORAL-DIFFERENCE (TD) METHODS - BOOTSTRAPPING

- The key idea underlying both **Dynamic Programming** (DP) and **Temporal-difference (TD)** learning (but NOT Monte-Carlo method).
- Updating an estimate from an estimate, a guess from a guess.
- Based on the **Bellman expectation equation**:

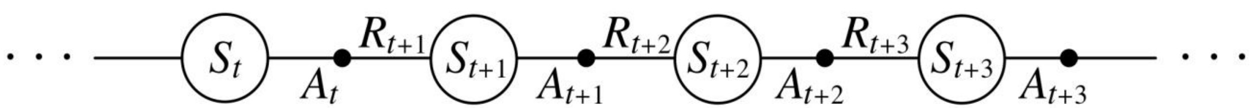
$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

- Or the **Bellman optimallity equation** (the target in Q-learning):

$$Q_*(s,a) = E_\pi[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a]$$

MODEL-FREE RL: TEMPORAL-DIFFERENCE (TD) METHODS - SARSA

- MC has a low performance due its high variability, so we add bias to it by bootstrapping.
- SARSA is easier than MC and more efficient.
- The agent takes one step (in time t), observes the next state (s in t+1), and reward (r in t+1) and chooses the next action (a in t+1) then updates Q value in time t.



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

MODEL-FREE RL: TEMPORAL-DIFFERENCE (TD) METHODS – SARSA ALGORITHM

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

MODEL-FREE RL: TEMPORAL-DIFFERENCE (TD) METHODS – Q-LEARNING ALGORITHM

MACQUARIE University

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

MODEL-FREE RL: TEMPORAL-DIFFERENCE (TD) METHODS – SARSA VS. Q-LEARNING [7]

SARSA:

- SARSA is on-policy.
- It learns Q-values that answer the question "What would this action be worth in this state, assuming I stick with my policy?"

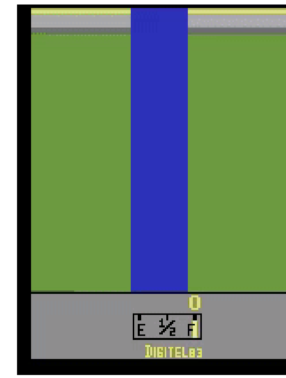
$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-Learning:

- Q-learning is off-policy.
- It learns Q-values that answer the question "What would this action be worth in this state, assuming that I stop using whatever policy I am using now, and start acting according to a policy that chooses the best action (according to my estimate)?"

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

SIMULATED ENVIRONMENTS FOR RL

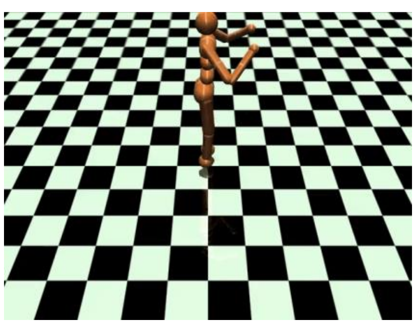


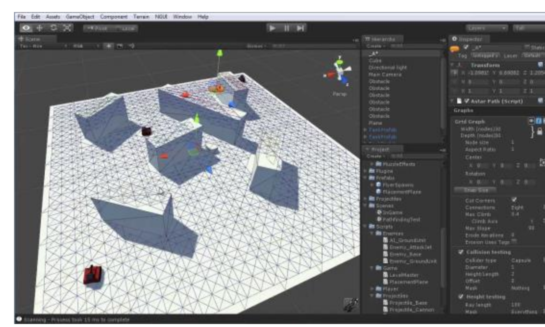
Why video games are good platforms for RL?

- Cheap (compare it to robotics)
- Accessible (you don't need to be in Boston Dynamics to use it!)
- Parallelizable (we can easily spawn multiple parallel environments)
- Easy to modify
- Beneficial for the video game industry:
  - e.g., Good for designing NPCs (non-player characters)

Popular Benchmarks and Environments for RL:

Openai Gym, Mujoco, VizDoom, Deepmind Lab, MindMaker Deep RL package (for the Unreal game engine), Unity ML-Agents (for the Unity game engine), ...





CONCLUSION

- RL is a big and inter-disciplinary topic, with a long history, an elegant mathematical core, novel algorithms, a lot of uncharted territories, and endless possibilities.
- RL can be seen as a unifying thread that pulls the entire AI together. You can find room for vision, perception, language, acting, planning, learning, even knowledge representation in it.
- RL has the capacity to reduce learning problems to small steps and deal with them in a systematic way.
- Recent developments showed the tremendous capacity of RL in robotics and simulated environments.
- The best is yet to come! Deep RL in week 12.

REFERENCES

1. Zachary Fruhling Website.
2. Ian Ravenscroft, "Philosophy of Mind", Oxford University Press, 2005.
3. David J. Chalmers, "Philosophy of Mind, Classical and Contemporary Readings", 2002.
4. Emma Brunskill's RL lectures, 2018.
5. Sutton's example: Reinforcement Learning Specialization, Coursera.
6. David Silver's Slides, DeepMind.
7. Russel, Norvig, "Artificial Intelligence: A Modern Approach", 2020.
8. Graesser, Loon Keng, "Foundations of Deep Reinforcement Learning", 2019.
9. R. Sutton, "Reinforcement Learning: An introduction (2e)", 2018.
10. R. Sutton, "Introduction to RL Lecture", NeurIPS 2015.
11. Y. Abu-Mostafa, "Learning From Data", 2012.