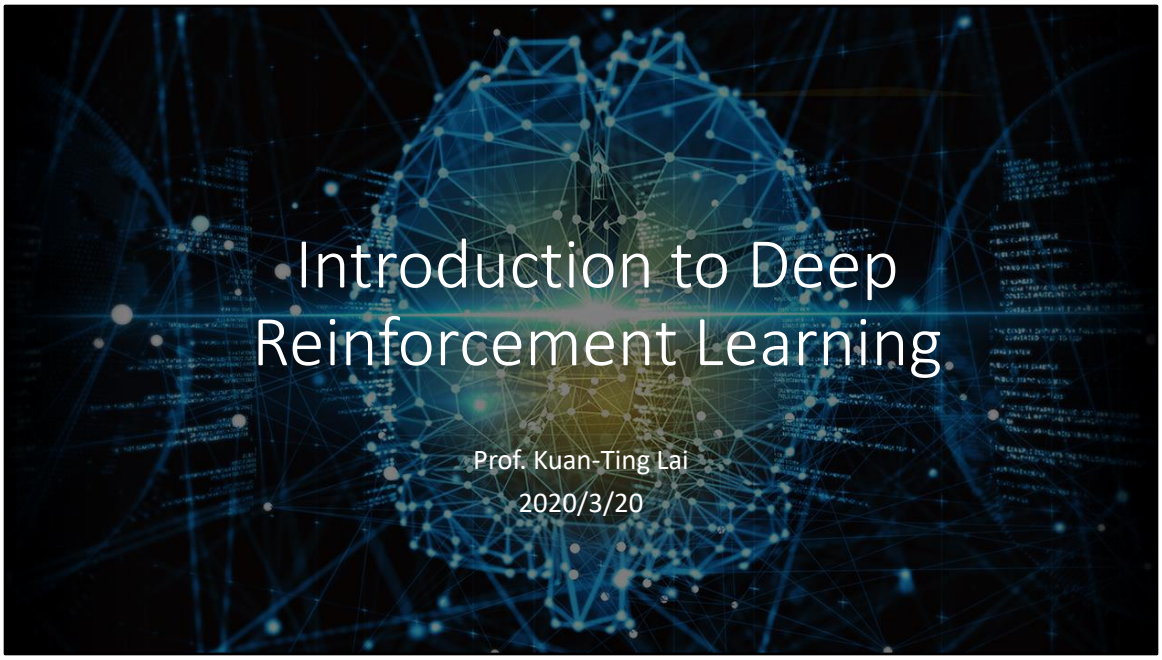


Introduction to Deep Reinforcement Learning

Prof. Kuan-Ting Lai

2020/3/20

What is Reinforcement Learning?

What is reinforcement learning? In its simplest form, reinforcement learning is “trial and error” plus “delayed reward”. “trial and error” means “learn from your own mistakes”, while “delayed reward” means our goal of learning is to maximize the cumulative reward in the end.

什麼是強化學習？ 在最簡單的形式中，強化學習是“嘗試錯誤”加上“延遲獎勵”。 “嘗試錯誤”意味著“從自己的錯誤中學習”，而 “延遲獎勵”代表我們的學習目標是最終使累積獎勵最大化。

Reinforcement Learning in Humans

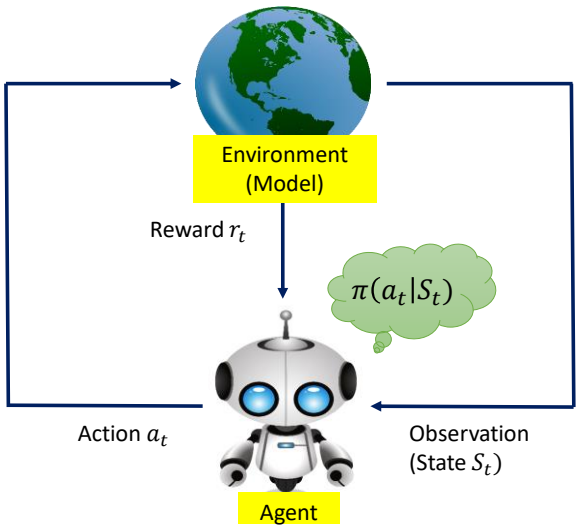
- Human can learn from “very few examples” + “trial-and-error”
- How? We don’t know yet…
- Possible answers
  - Hardware: 230 million years of bipedal movement data
  - Imitation learning: observing other human
  - Algorithms: Better than backpropagation & SGD

Lex Fridman, MIT Deep Learning, https://deeplearning.mit.edu/

In fact, there are a lot of reinforcement learning examples of animals. For example, let’s think about a baby that tries to learn how to walk by themselves. They’ll keep falling and standing up, falling and standing up many many times until they learn how to walk. However there's a big difference between today’s RL algorithms and the animals’ learning. That is, animal and human can learn from “very few examples” . This is an unsolved problem and we still don’t know why. The possible answers may be that animals have some pre-programmed algorithms in their brain after the 230 million evolution, or human can do imitation learning by observing other human, or that we need better algorithms than current backpropagation and gradient descent.

實際上，有很多關於動物的強化學習實例。 例如，讓我們考慮一個試圖學習如何獨自走路的嬰兒。 他們會不斷摔倒並站起來，摔倒並站起來很多次，直到他們學會走路。 但是，當今的RL算法與動物的學習方法之間存在很大差異。 那就是動物和人類可以從“很少的例子”中學習。 這是一個尚未解決的問題，我們仍然不知道為什麼。 可能的答案可能是，在經歷了2.3億次進化之後，動物的大腦中已經有了一些預編程的算法，或者人類可以通過觀察其他人類來進行模仿學習，或者我們需要比當前的反向傳播和梯度下降更好的算法。

Define Reinforcement Learning

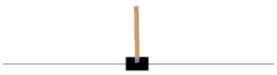


Let's define deep reinforcement learning mathematically. First, there is a software agent and an environment. The environment provides observations and rewards to the agent. Observation is also called state St. The goal of the agent is learning how to perform different actions a_t under various states S_t, to achieve maximum cumulative future reward R_t. In other words, the software agents learn an optimized action model in an environment by doing trial-and-errors million times. The model is called policy $\pi(a_t|S_t)$.

讓我們用數學定義深度強化學習。 首先，有一個軟體代理程式(agent)和一個環境。 環境向agent提供了觀察和獎勵。 觀察又稱為狀態St。Agent的目標是學習如何在各種狀態S_t下選擇不同的動作a_t，以實現最大的累積未來獎勵R_t。 換句話說，軟件代理通過進行百萬次反複試驗來學習環境中的優化操作模型。 該模型稱為Policy(策略) $\pi(a_t|S_t)$。

Environments and Actions

- Fully observable (Chess) vs Partially observable (StarCraft)
- Single Agent (Atari) vs Multi-agent (DeepTraffic)
- Deterministic (Chess) vs Stochastic (DeepTraffic)
- Discrete (Chess) vs Continuous (Cart Pole)



Cart Pole

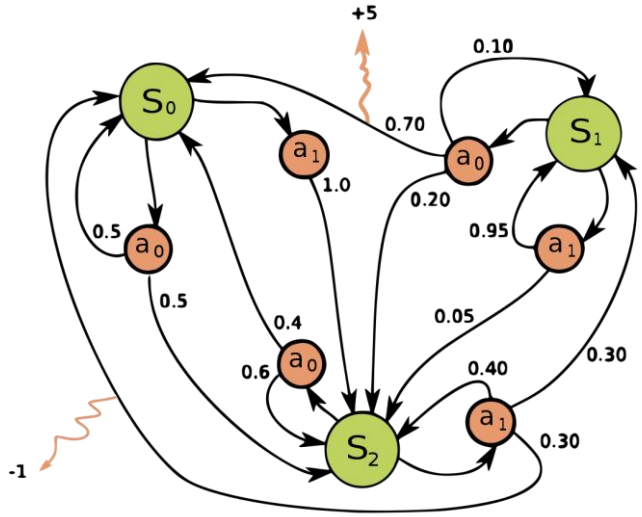
DeepTraffic

Lex Fridman, MIT Deep Learning, https://deeplearning.mit.edu/

There are many kinds of environments and actions, as listed in this slide. Chess is fully observable, which means we can observe all the actions of our opponents. On the contrary, the video game StarCraft is only partially observable. The other characteristics is if the learning is performed by single agent, or cooperative learning of multi-agents, such as the simulation environment DeepTraffic developed by MIT. Another characteristics is whether the environment is deterministic, i.e., the environment rules don't change during learning, or stochastics. Finally, the environment can be either discrete or continuous.

如本頁中所列，有許多種環境和操作。 棋類遊戲是fully observable，這意味著我們可以觀察對手的所有動作。 相反，電子遊戲《星際爭霸》只能部分觀察到對手動作。 其他特徵是學習是由單個代理執行的，還是由多代理協作學習的，例如由MIT開發的仿真環境DeepTraffic。 另一個特徵是環境是否具有確定性，即在學習或隨機過程中環境規則不會改變。 最後，環境可以是離散的，也可以是連續的。

Markov Decision Process (MDP)



https://en.wikipedia.org/wiki/Markov_decision_process

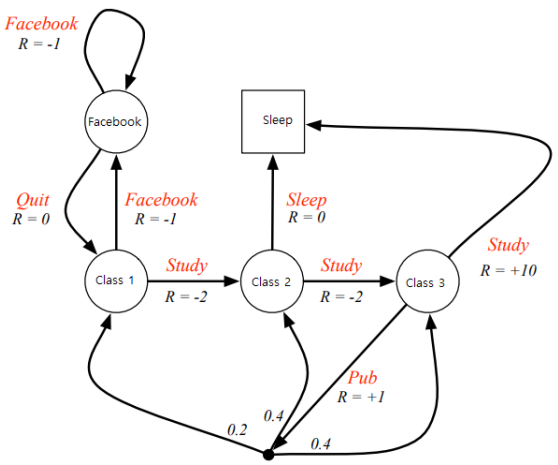
The previous model can be formulated as a **Markov decision process** (**MDP**), which is a discrete time stochastic control process. MDP has been widely used in many disciplines, including robotics, automatic control, economics and manufacturing. MDPs were proposed as early as the 1950s and provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

At each time step, the process is in some state S, and the decision maker may choose any action a that is available in state s. The process responds at the next time step by randomly moving into a new state s' and giving the decision maker a corresponding reward Ra(s,s'). For more details, please refer to Wikipedia (https://en.wikipedia.org/wiki/Markov_decision_process).

先前的模型可以表述為馬爾可夫決策過程（MDP），這是一個離散時間隨機控製過程。 MDP已廣泛應用於許多學科，包括機器人技術，自動控制，經濟學和工業製造。 MDP最早是在1950年代提出的，它提供了一種數學模型，用於在結果部分隨機且部分受決策者控制的情況下對決策制定進行建模。 MDP可用於研究通過dynamic programming和強化學習解決的優化問題。

在每個時間步驟中，流程都處於某種狀態S，決策者可以選擇狀態中可用的任何操作a。 該過程在下一時間步響應，隨機進入新狀態s’，並給予決策者相應的獎勵Ra（ s, s’）。 詳細信息請參閱Wikipedia。

Example: Student MDP



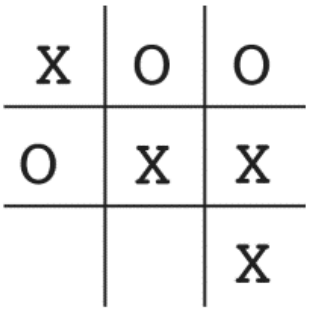
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf by David Silver

Here is a student Markov Decision Process (MDP) from David Silver's RL course. A student first go to a class "Class 1". After the first class, a student may choose to do Facebook or continue to join the next class "Class 2". Doing Facebook will hurt your learning, so you get the reward – 1. However, continuing go to class will hurt your feeling, so you get reward -2. if you can insist to go to class 3 and study, then you will pass and get reward +10, and go to sleep finally. Sleep is the terminal state of the MDP. On the other hand, you may choose to go to the pub for relax, and then go back to class 1, class 2 or class 3.

這是David Silver的RL課程的學生馬爾可夫決策過程（MDP）範例。 一個學生首先去 " Class 1"上課。 上完第一堂課後，學生可以選擇做Facebook或繼續參加下一堂課 " Class 2"。 做Facebook會損害您的學習，因此您會獲得獎勵-1。但是，繼續上課會損害您的感覺，因此您會獲得獎勵-2。 如果您可以堅持上完第三節課Class 3，那麼您將通過並獲得+10的獎勵，並最終入睡。 睡眠是MDP結束狀態。另一方面，您可以選擇去酒吧放鬆一下，然後回到Class 1，Class 2或Class 3繼續學習 .

Example: Tic-Tac-Toe

- One player plays Xs and the other Os until one player wins by placing three marks in a row horizontally, vertically, or diagonally.



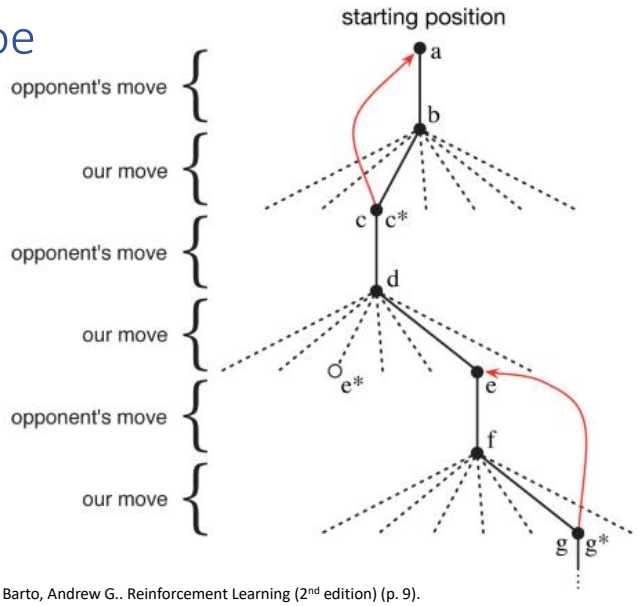
Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (2nd edition) (p. 8).

Let's look at another example. Consider the familiar child's game of tic-tac-toe. Two players take turns playing on a three-by-three board. One player plays Xs and the other Os until one player wins by placing three marks in a row, horizontally, vertically, or diagonally, as the X player has in the game shown to the right. If the board fills up with neither player getting three in a row, then the game is a draw.
Because a skilled player can never lose, let us assume that we are playing against an imperfect player, one whose play is sometimes incorrect and allows us to win.

我們再來看另外一個例子: 孩子的井字遊戲。 兩名玩家輪流玩三乘三局。 一位玩家玩Xs，另一位玩家玩Os，直到一位玩家通過在水平，垂直或對角線上連續放置三個標記（如X玩家在右圖中所示）獲勝為止。 如果棋盤上的棋子都滿了，而沒有一個玩家連續獲得3個棋子，那麼這場比賽就是平局。
因為熟練的玩家永遠不會輸，所以讓我們假設我們正在與一個不完美的玩家進行比賽，這個玩家有時的比賽是不正確的，並且可以讓我們獲勝。

Playing Tic-Tac-Toe

X	O	O
O	X	X
		X



Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (2nd edition) (p. 9).

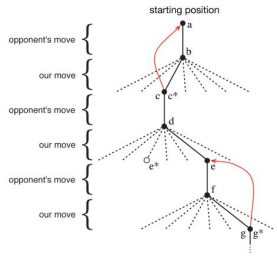
When playing a game, our move will affect opponents' move, and opponents' move will affect our move, vice versa. This process can be expanded as a tree structure. If the number of states is small, we can exhaustively calculate all winning possibilities. First, we would set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state. We treat this estimate as the state's value, and the whole table is the learned value function.
We then play many games against the opponent. We move greedily, selecting the move that leads to the state with greatest value, that is, with the highest estimated probability of winning. Occasionally, however, we select randomly from among the other moves instead. These are called exploratory moves because they cause us to experience states that we might otherwise never see, and let us learn other playing strategies.

在玩遊戲時，我們的舉動會影響對手的舉動，而對手的舉動會影響我們的舉動，反之亦然。 此過程可以擴展為樹結構。 如果狀態state的數目很小，我們可以計算出詳盡所有獲獎的可能性。
首先，我們將建立一個數字表，每個可能的遊戲狀態對應一個數字。 每個數字都是對我們從該state獲勝的可能性的最新估計。 我們將此估算值視為state的價

值(value)，整個表格就是學習後的價值函數。
然後，我們與對手進行了許多比賽。 我們貪婪地移動，選擇導致該狀態具有最大價值的下法，就是獲勝概率最高方法。 但是，有時我們會從其他動作中隨機選擇。 這些之所以稱為探索性動作，是因為它們使我們經歷了本來無法看到的狀態，讓我們有機會學習不同的策略。

Updating the Value Function

- S_t: State at time t
- $V(\quad)$: Value (expected return)
- α: Step-size parameter (learning rate)

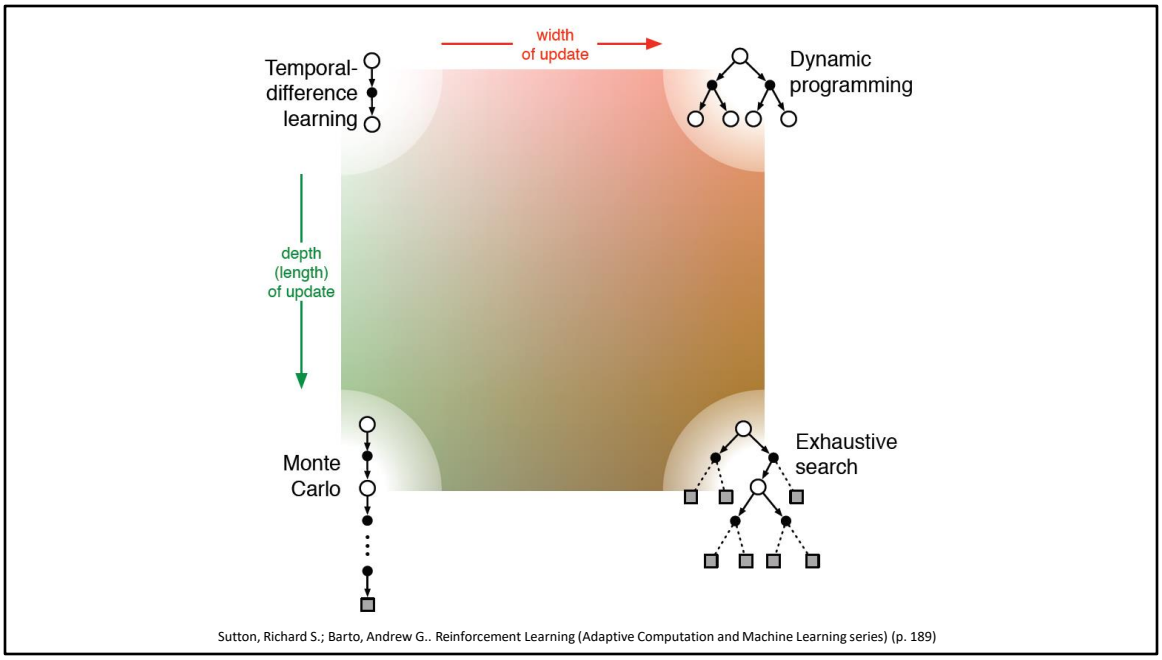


$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

Temporal Difference

While playing tic-tac-toe, we keep updating the values of the states $V(S_t)$. We attempt to make them more accurate estimates of the probabilities of winning. To do this, we "back up" the value of the state after each greedy move to the state before the move, as suggested by the red arrows. More precisely, the current value of the earlier state $V(S_t)$ is updated to be closer to the value of the later state $V(S_{t+1})$.

玩井字遊戲時，我們會不斷更新遊戲過程中所處狀態的值$V(S_t)$，試圖使他們更準確地估計獲勝的概率。 如紅色箭頭所示。我們將每次貪婪移動後的狀態值“回溯”到移動前的狀態。 更精確地，更新前狀態的value $V(S_t)$為更接近較晚狀態的value $V(S_{t+1})$。



Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (Adaptive Computation and Machine Learning series) (p. 189)

All kinds of reinforcement learning algorithms can be viewed as searching along width or depth of the MDP expanded tree structure. The record of each depth searching is called a trajectory.

All RL methods have three key ideas in common: first, they all seek to estimate value functions; second, they all operate by backing up values along actual or possible state trajectories; and third, they all follow the general strategy of generalized policy iteration (GPI), meaning that they maintain an approximate value function and an approximate policy, and they continually try to improve each on the basis of the other. These three ideas are central to the subjects covered in Sutton's book.

As shown in the figure, these dimensions have to do with the kind of update used to improve the value function. The horizontal dimension is whether they are sample updates (based on a sample trajectory) or expected updates (based on a distribution of possible trajectories). Expected updates require a distribution model, whereas sample updates need only a sample model, or can be done from actual experience with no model at all (another dimension of variation).

The vertical dimension corresponds to the depth of updates, that is, to the degree of bootstrapping. At three of the four corners of the space are the three primary methods for estimating values: dynamic programming, TD, and Monte Carlo. Dynamic programming methods are shown in the extreme upper-right corner of the space

because they involve one-step expected updates of all states. The lower-right corner is the extreme case of expected updates so deep that they run all the way to terminal states.

各種強化學習算法都可以看作是沿著MDP擴展樹結構的寬度或深度進行搜索。每次深度搜索的記錄稱為軌跡。
所有RL方法都具有三個共同的關鍵思想：第一，它們都試圖估計價值函數；第二，它們都試圖估計價值函數。其次，它們都通過沿實際或可能的狀態軌跡回溯和更新價值函數的值。第三，它們都遵循廣義策略迭代（GPI）的通用策略，這意味著它們維護一個近似值函數和一個近似策略，並且不斷地在彼此的基礎上不斷進行改進。這三個想法對於Sutton的書所涵蓋的主題至關重要。
如圖所示，這些維度與用於改進值函數的更新類型有關。水平維度是它們是樣本更新（基於樣本軌跡）還是預期更新（基於可能軌蹟的分佈）。預期的更新需要一個分佈模型，而樣本更新則只需要一個樣本模型，或者可以完全不使用模型（另一維度的變化）根據實際經驗來完成。
垂直尺寸對應於更新的深度，即自我學習的程度。在空間的四個角中的三個角處，存在三種用於估算值的主要方法：Dynamic Programming，TD和蒙特卡洛。Dynamic programming顯示在空間的右上角，因為它們涉及所有狀態的一步式預期更新。右下角是預期更新的極端情況，其深度如此之深以至於它們一直運行到終端狀態。

Exploration vs. Exploitation

- ε-greedy policy
  - With ε performs random action, (1- ε) performs greedy action
  - Slowly move to greedy policy ε -> 0

Reinforcement learning faces the exploration and exploitation dilemma. Exploration means the agent try to search for new ways by taking random actions. Exploitation means the agent follows current policy and selects actions greedily, that is, always selects the current best action. Exploration tries to find better strategies while exploitation focuses on improving current policy. The most common method is ε-greedy policy, which performs random action ε of the time and performs greedy action (1- ε) of the time.

強化學習面臨探索與開發的困境。 探索Exploration意味著代理商嘗試通過採取隨機行動來尋找新的方式。 開發Exploitation是指代理遵循當前策略並貪婪地選擇操作，即是永遠選擇當前最佳操作。 探索試圖找到更好的策略，而開發則著眼於改善當前政策。 最常見的方法是ε貪心策略，該策略是在ε時間內的執行隨機動作，然後在其他的（1-ε）時間內執行的貪婪(目前最佳的)動作。

On-policy vs. Off-policy

- On-policy
  - Uses one policy, exploit (1- ε) and explore ε of the time
  - Simpler and fast to converge to local minima
- Off-policy
  - Uses two policies: target policy and behavior policy
  - Target policy is the optimal policy
  - Behavior policy is more exploratory
  - Harder to train but more powerful

Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (Adaptive Computation and Machine Learning series) (p. 103).

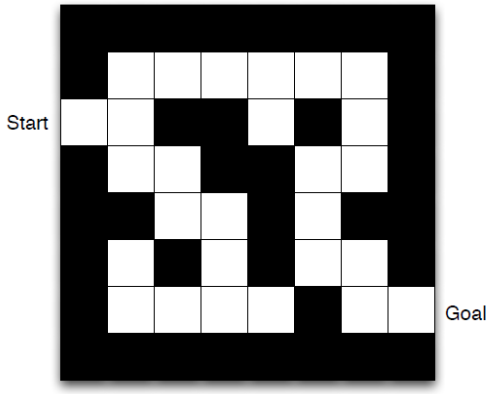
All learning control methods face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions (and then find the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy? The on-policy approach such as ε-greedy is actually a compromise— it learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the target policy, and the policy used to generate behavior is called the behavior policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning.

所有學習控制方法都面臨一個難題：它們試圖以後續的最佳行為為條件來學習動作值，但是它們需要表現為非最佳狀態才能探索所有動作（然後找到最佳動作）。 他們如何根據探索性策略了解最佳策略？ On-policy方法（例如ε-greedy）實際上是一種折衷方案，它不是針對最優策略而是針對仍在探索的近乎最優的策略來學習動作值。 一種更直接的方法是使用兩種策略，一種是已了解並成為最佳策略，另一種是更探索性的用於產生行為的策略。 正在學習的策略稱為目

標策略，而用於生成行為的策略稱為行為策略。 在這種情況下，我們說學習是從目標策略“脫離”數據中進行的，整個過程稱為Off-policy。

Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (Adaptive Computation and Machine Learning series) (p. 103). The MIT Press.

Maze Example



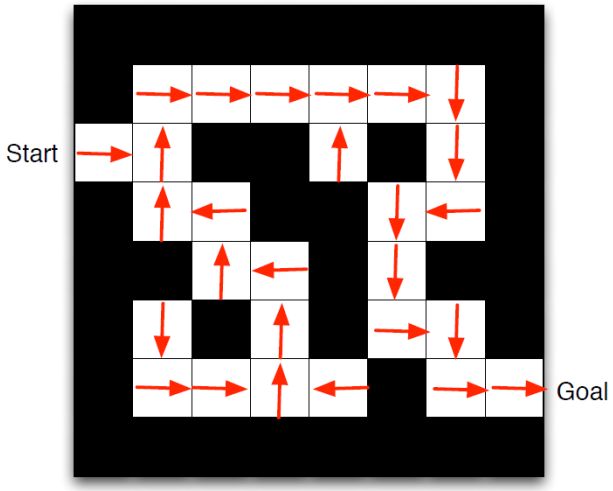
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf by David Silver

Let's look at a maze learning example. This example is from the first class of David Silver's Reinforcement Learning. The objective of the agent is to find the shortest path between "Start" and "Goal". We give the agent -1 reward per time-step to encourage it to arrive at the Goal in shortest time. There are four actions: moving north (N), east (E), south (S) or west (W). The states are the agent's location.

讓我們看一個迷宮學習的例子。 該示例來自David Silver的強化學習第一堂課。 Agent的目標是找到“Start”和“Goal”之間的最短路徑。 我們會按時間向agent提供-1獎勵，以鼓勵其在最短時間內到達目標。 Agent有四個動作：向北（N），向東（E），向南（S）或向西（W）移動。 環境狀態(States)是Agent目前的位置。

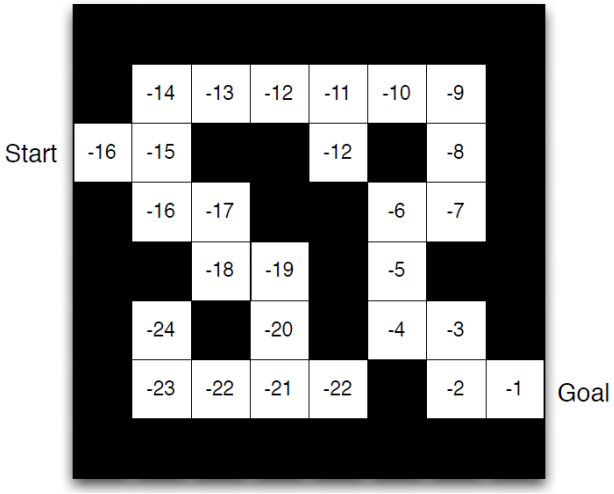
Maze Example: Policy $\pi(a_t|S_t)$



Here is the visualization of Policy $\pi(a_t|S_t)$. In this example, the red arrows are the actions the agent choose to do under different state. It is a deterministic policy, which means we always select one specific action at specific state.

這是策略Policy $\pi(a_t|S_t)$的視覺化圖。在這個範例中，紅色箭頭是agent在不同狀態下選擇執行的操作。 這是一種確定性策略，這意味著我們總是在特定狀態下選擇一個特定操作。

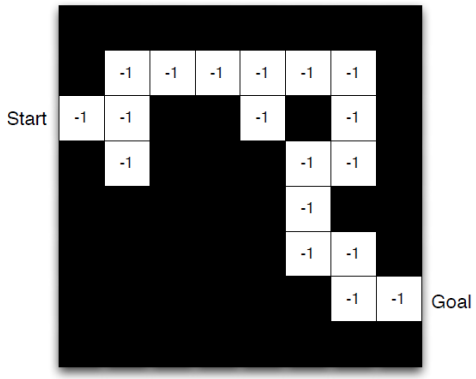
Maze Example: Value Function



Let's look at the value function of the map. Starting from the location nearest to "Goal", we have -1 reward, because it will take only one step to reach the goal. As we can see, by moving toward the location with higher value (reward), we can easily find the shortest path from "Start" to "Goal".

讓我們看看地圖的價值函數(value function)。 從最接近“目標”的位置開始，我們將獲得-1獎勵，因為它只需一步就可以達到目標。 如我們所見，通過向價值(Reward)更高的位置移動，我們可以輕鬆找到從“Start”到“Goal”的最短路徑。

Maze Example: Model

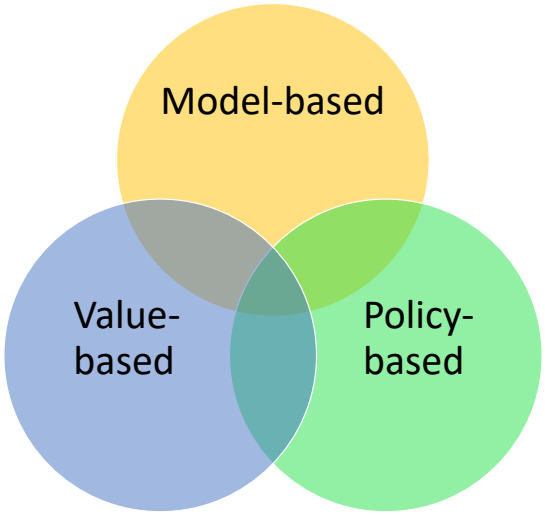


- Agent may have an internal model of the environment
- The model may be imperfect
- Numbers (-1) represent immediate reward R from each state s (same for all actions)

In addition, the agent can build an internal model of the environment from its own exploration experiences. For example, the agent may have found the way to the "Goal", and create an internal model based on its trajectories, as shown at the left.

另外，agent可以根據自己的探索經驗來構建環境的內部模型。 例如，agent可能已經找到通向“目標”的道路，並根據其軌跡創建內部模型，如左圖所示。

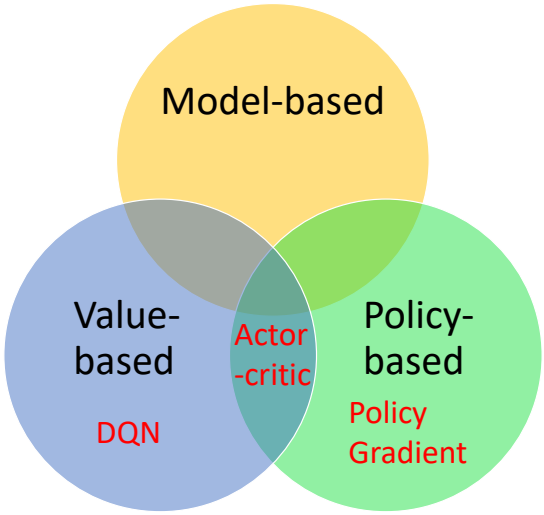
3 Types of Reinforcement Learning



So now we know that reinforcement learning algorithms can be classified into three types: model-based, value-based and policy-based.

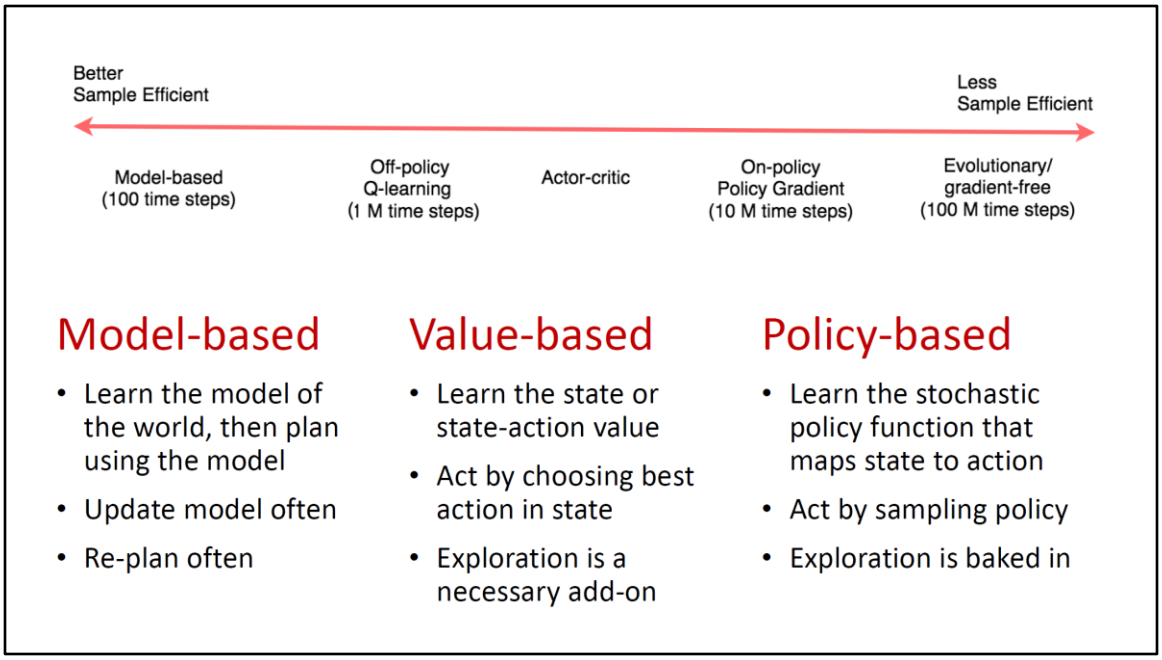
現在我們知道強化學習策略可以分為三種: model-based, value-based和policy-based.

3 Types of Reinforcement Learning



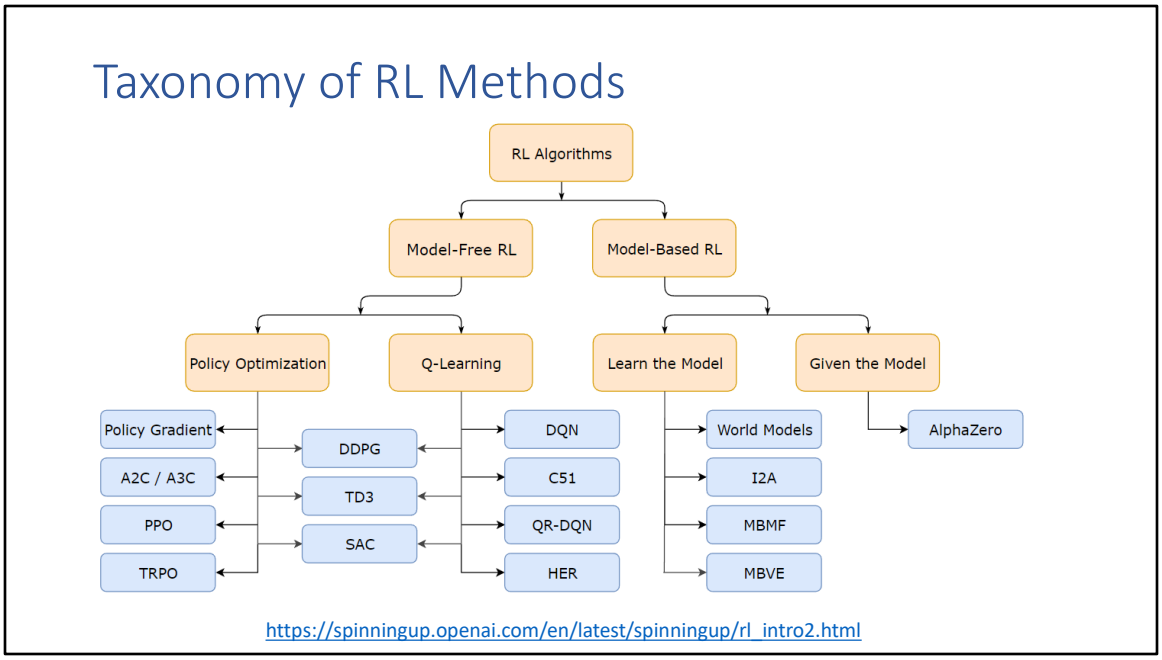
Most RL algorithms are model-free and belong to value-based or policy-based methods, such as Deep Q Networks (DQN) and Policy Gradient. Some algorithms are mix of two types, such as Actor-critic.

大多數RL算法是無環境模型的，所以屬於value-based或基於policy-based的方法，例如Deep Q Networks（ DQN ）和Policy Gradient。 某些算法是兩種類型的組合，例如Actor-critic。



If we have a good model of the environment, then we can use the model to plan the actions, and the learning is more efficient. However, it's difficult to model a complex environment in practice. The value-based methods learn the possible reward of each state or action, and use those values to select best actions. On the other hand, the policy-based methods directly optimize policy functions and are almost always on-policy, which requires more samples for learning.

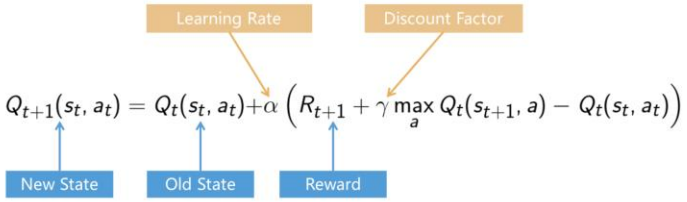
如果我們有一個好的環境模型，那麼我們可以使用該模型來計劃行動，這樣學習就會更加有效。 但是，實務上很難為複雜的環境建模。 Value-based方法學習每個狀態的可能獎勵，並使用這些值選擇最佳操作。 另一方面，Policy-based的方法直接優化策略函式，並且幾乎都使用on-policy，因此需要更多的學習資料。



This is the reinforcement learning taxonomy from OpenAI. This tree diagram includes most state-of-the-arts RL algorithms. OpenAI classify the model-free methods into two categories: policy optimization and Q-learning. Q-learning family aims to learn an action-value function Q(s,a). The most famous Q-learning based algorithm is Deep Q Networks (DQN), the breakthrough algorithm proposed by DeepMind, which combines Q-learning with neural networks.

這是來自OpenAI的強化學習分類法。 該樹形圖包含了大多數最新的RL算法。 OpenAI將無環境模型(Model-free)方法分為兩類：策略優化和Q-learning。 Q-learning類的演算法旨在學習行動價值函數Q（s，a）。 最著名的基於Q學習的算法是Deep Q Networks（DQN），它是DeepMind提出的突破性算法，它將Q學習與神經網絡相結合。

Q-Learning: Value Iteration



	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

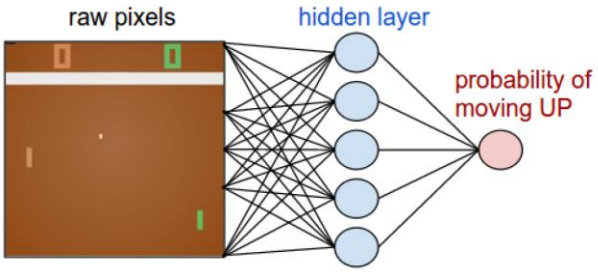
Let's take a look at the fundamental concepts of Q-learning. Q-learning learns to estimate the expected future reward for different actions under various states. We can formulate it as $Q(s_t, a_t)$, and update Q function using Bellman equation. If the number of states and actions are few, we can build an action-state table, which is also called Q-table, to calculate the corresponding rewards of all action-state pairs, as shown at the lower-left corner. However, the number of states are very large in real world, it is inefficient and nearly impossible to build a complete action-state table. DeepMind solved this problem by using neural network to approximate the Q-table.

讓我們看一下Q-learning的基本概念。 Q-learning學習估計在不同狀態下各種動作的預期未來回報。 我們可以將其公式化為Q（s_t，a_t），並使用Bellman方程更新Q函數。 如果狀態和動作的數量很少，我們可以構建一個動作狀態表(又稱Q-table)來計算所有動作狀態對的相應獎勵，如左下角所示。 但是，在現實世界中，狀態的數量非常多，建立完整的動作狀態表是很沒有效率的。 DeepMind通過使用神經網絡近似Q table來解決此問題。

Policy Gradient (PG)

- PG (on-policy): Directly optimize policy function

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t]$$



Policy Network

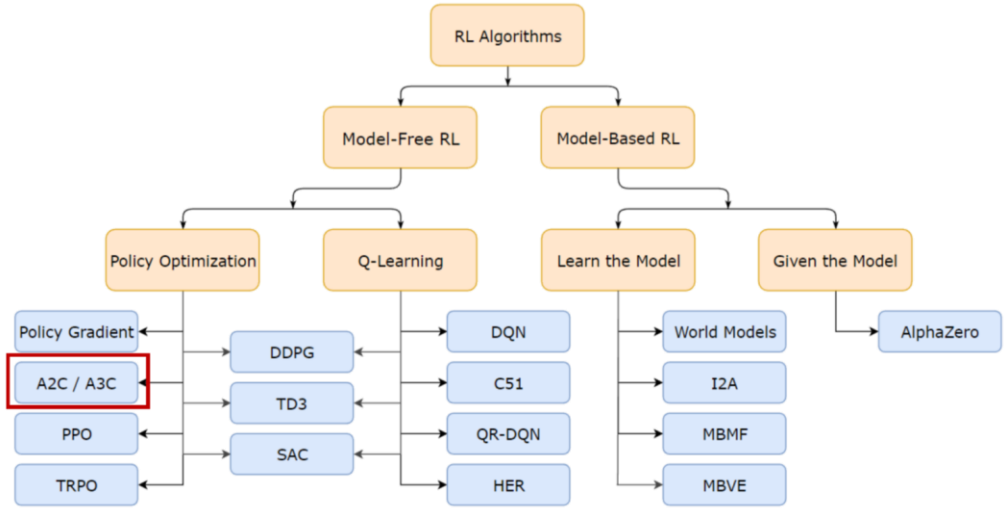
Good illustrative explanation:
http://karpathy.github.io/2016/05/31/rl/

"Deep Reinforcement Learning: Pong from Pixels"

Policy gradient is an intuitive method that directly optimizes the policy function. This is a on-policy method because we only have one policy function. Comparing to off-policy methods such as DQN, policy gradient is easier to train and converge, but also tend to stuck at local minima. Anderj Karparthy has made an interesting experiment of training agent to play game Pong using policy gradient. You can refer to his Github.

策略梯度(Policy gradient)是一種直接優化策略功能的直觀方法。 這是一種On-policy的方法，因為我們只有一項策略函式。 與DQN等Off-policy方法相比，Policy gradient更易於訓練和收斂，但也傾向於停留在局部最小值。 Anderj Karparthy有一個有趣的實驗，培訓代理使用策略梯度來玩Pong遊戲。 您可以參考Karpathy的Github。

Actor-Critic Methods



Actor-Critic Methods

- Actor-Critic: combine policy gradient and Q-learning
  - Actor: decides which action to take
  - Critic: measure how good the action was and how to improve

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] && \text{TD Actor-Critic}
\end{aligned}
$$

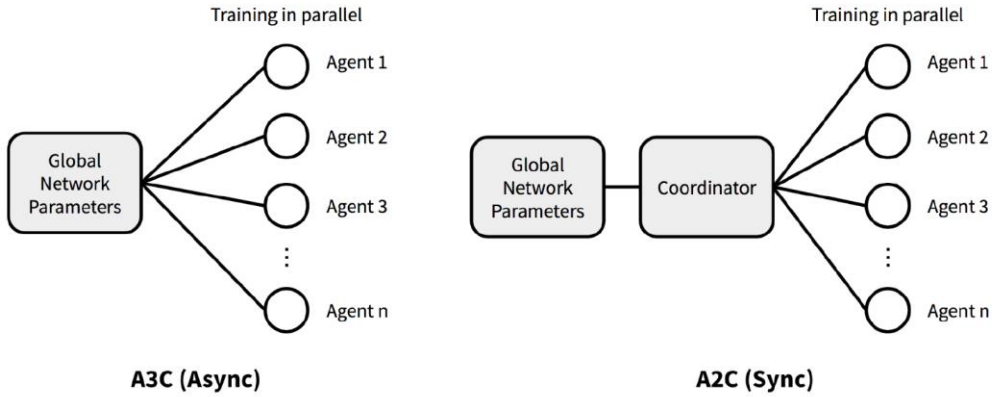
Image taken from CMU CS10703 lecture slides

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

The Actor-Critic methods combines the policy gradient and Q-learning. The Actor learns to optimize the policy and decides which action to take. The Critic adopts a value-based method and evaluates the performance of the Actor. Depends on types of the value function, the Actor-Critic methods can be classified into Q Actor-Critic, Advantage Actor-Critic, and Temporal-Difference (TD) Actor-Critic.

Actor-Critic方法結合了policy gradient和Q-Learning。 Actor學會優化政策並決定採取哪種行動。 Critic採用基於價值的方法並評估Actor的表現。 根據值函數的不同，Actor-Critic方法可以分為Q Actor-Critic，Advantage Actor-Critic和Temporal-difference（TD）Actor-Critic。

Asynchronous Advantage Actor-critic (A3C)



https://theaisummer.com/Actor_critics/

DeepMind proposed the Asynchronous Advantage Actor-critic (A3C) method in 2016. Multiple agents learn parallelly in their own copy of the environment, and then update the global network periodically. The updates are not happening simultaneously and that's where the asynchronous comes from. After each update, the agents resets their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again. We see that the information flows not only from the agents to the global network but also between agents as each agent resets his weights by the global network, which has the information of all the other agents.
OpenAI suggested that A3C can be more efficient by adding a coordinator and updates the global network parameters synchronous. They proposed Synchronous Advantage Actor-Critic (A2C) and show that A2C outperforms A3C on many reinforcement learning tasks.

DeepMind在2016年提出了“非同步Advantage Actor-critic（A3C）”方法。多個代理在自己的環境副本中並行學習，然後定期更新全局網絡。 更新不是同時發生的，而這正是所謂的”非同步”。 每次更新後，代理將其參數重置為全局網絡的參數，並繼續進行n步的獨立探索和訓練，直到他們再次重設自己的參數。 我們看到，信息不僅從代理流向全球網絡，而且還在代理之間流動，因為每個代

理都通過具有所有其他代理信息的全局網絡重置其權重。
OpenAI建議通過添加協調器並同步更新全局網絡參數來提高A3C的效率。 他們提出了A2C，並表明A2C在許多強化學習任務上勝過A3C。

Policy Optimization

- Avoid taking bad actions that collapse the training performance

Line Search:
First pick direction, then step size

Trust Region:
First pick step size, then direction

TRPO updates policies by picking step size that satisfy a special constraint on how close the new and old policies are. The space is called Trust Region. The constraint is expressed in terms of KL-Divergence, a measure of distance between probability distributions. This is different from normal policy gradient, which picks direction first, then step size. In practice, a single bad step can collapse the policy performance. This makes it dangerous to use large step sizes with traditional policy gradients. TRPO nicely avoids this kind of collapse, and tends to improve performance quickly. (https://spinningup.openai.com/en/latest/algorithms/trpo.html#background)

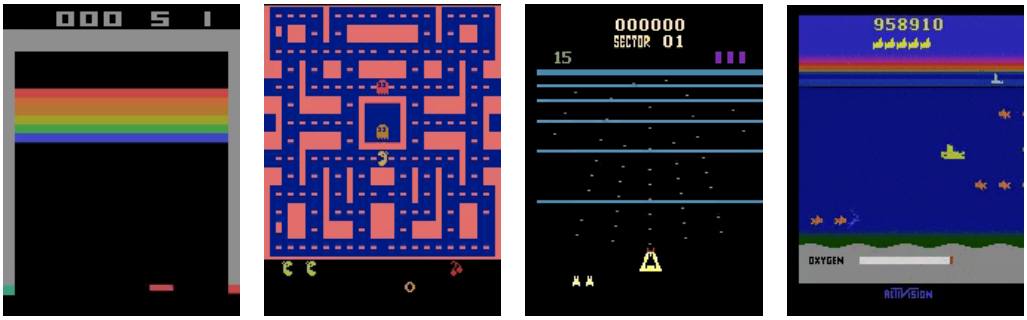
TRPO通過選擇更新step size來更新策略，這些step size滿足新舊策略的緊密程度之間的特殊約束。 該空間稱為“信任區域”。 該約束用KL-Divergence表示，KL-Divergence是用來量測概率分佈之間的距離。與一般的策略梯度的差別在於，後者先選擇方向，然後再選擇step size。 實際上，一個不好的步驟可能會使策略性能崩潰。 這使得在傳統策略梯度中使用大step size非常危險。TRPO很好地避免了這種崩潰，並可以快速提高學習效能。

DeepMind was founded by Demis Hassabis, Shane Legg and Mustafa Suleyman in 2010. The goal of the founders is to create a general-purpose AI that can be useful and effective for learning almost anything. Because the founders has background in game industry, they started training AI agents how to play old Atari games that were popular in 1970s and 80s. On 26 January 2014, Google announced the company had acquired DeepMind for $500 million. The company made headlines worldwide in 2016 after its AlphaGo program beat a human world champion Go player Lee Sedol.

DeepMind由Demis Hassabis，Shane Legg和Mustafa Suleyman於2010年創立。創始人的目標是創建一種通用的AI，可以學習任何事物。 由於創始人具有遊戲行業背景，因此他們開始培訓AI代理如何玩1970年代和80年代流行的舊Atari遊戲。 2014年1月26日，Google宣布該公司已以5億美元的價格收購了DeepMind。 在其AlphaGo計劃擊敗了人類世界冠軍圍棋選手Lee Sedol之後，該公司在2016年成為全球頭條新聞。

Learn to Play Atari Games

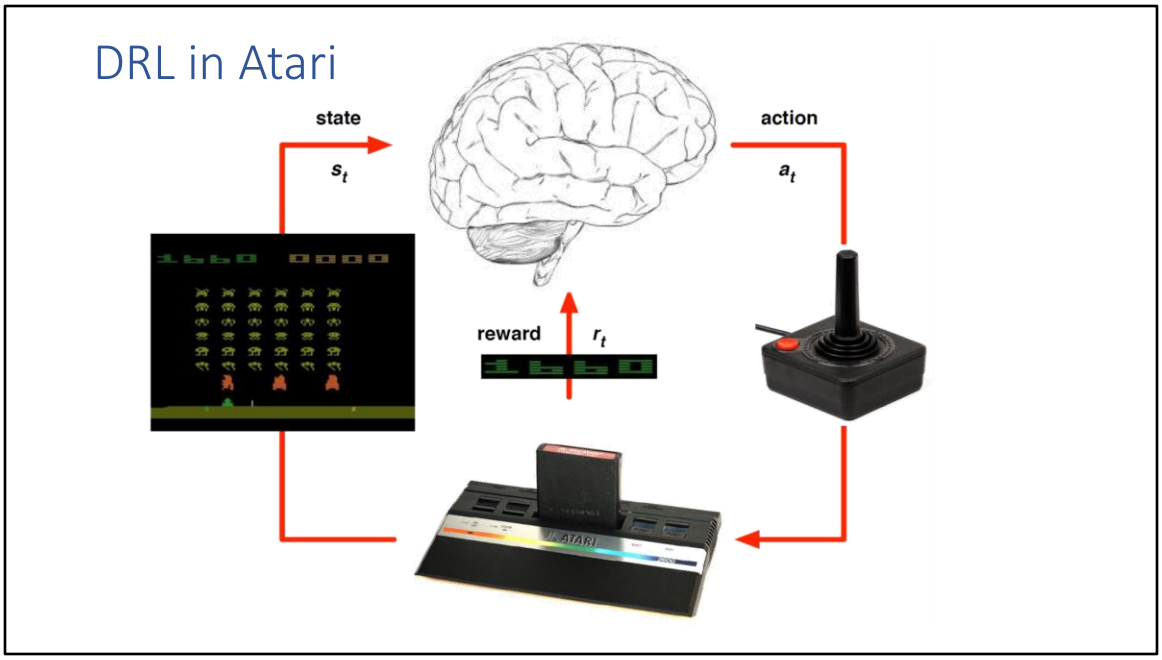
- Mnih et al., “Human Level Control through Deep Reinforcement Learning,” *Nature*, 2015



29

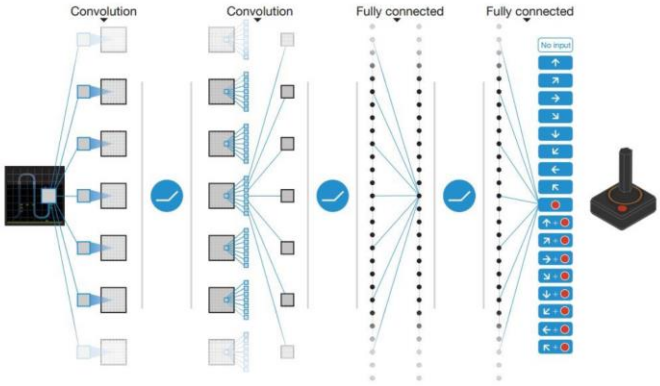
DeepMind published their first breakthrough algorithm, DQN, on Nature in 2015. Here are some examples of AI playing Atari games in OpenAI Gym. OpenAI Gym is a open-source RL simulation environments released by OpenAI.

DeepMind於2015年在《自然》雜誌上發表了他們的第一個突破性算法DQN。這是一些AI在OpenAI Gym中玩Atari遊戲的AI的範例。 OpenAI Gym是由OpenAI發布的開源RL模擬環境。



When training AI playing Atari games via reinforcement learning, the environment is the Atari game console, the reward is the score of the game, the state is the screenshots, and the actions are the directions and buttons of joy sticks.

通過強化學習訓練AI玩Atari遊戲時，環境是Atari遊戲機，獎勵是遊戲的得分，狀態是屏幕截圖，動作是操縱桿的方向和按鈕。



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

DeepMind adopted a small neural network that has three convolution layers and two fully connected layers. Surprisingly, the simple network can successfully learn many different Atari games and achieve human-level performance.

DeepMind採用了一個小型神經網絡，該網絡具有三個卷積層和兩個完全連接的層。 令人驚訝的是，這個簡單的網絡可以成功學習許多不同的Atari遊戲並達到人類水平的性能。

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

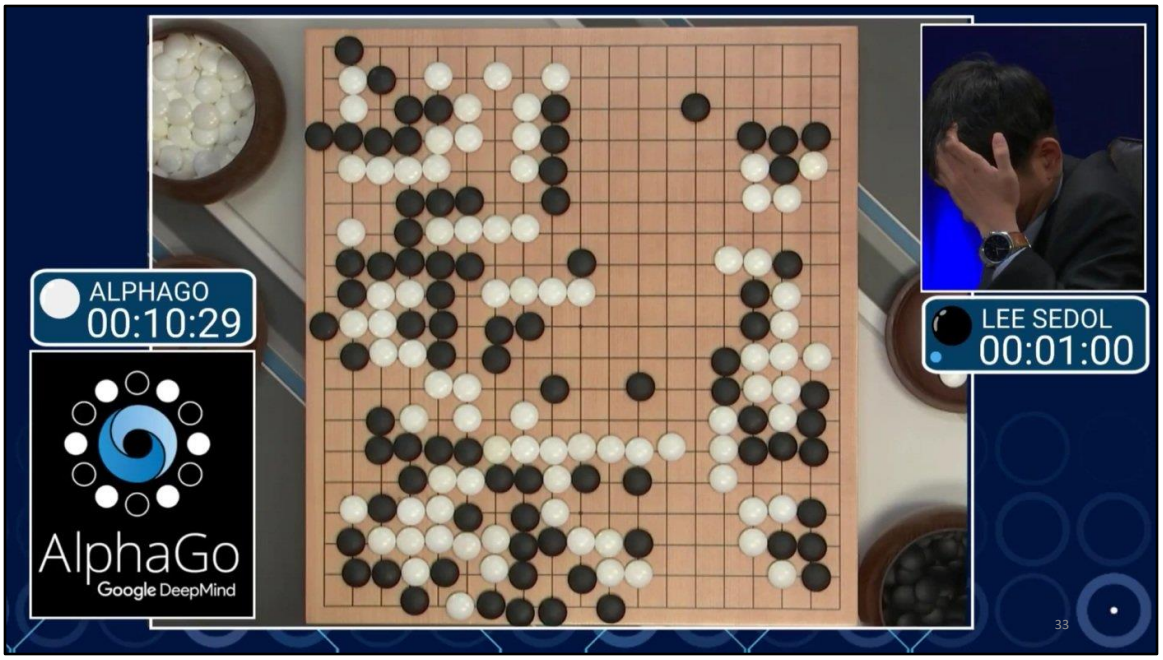
$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

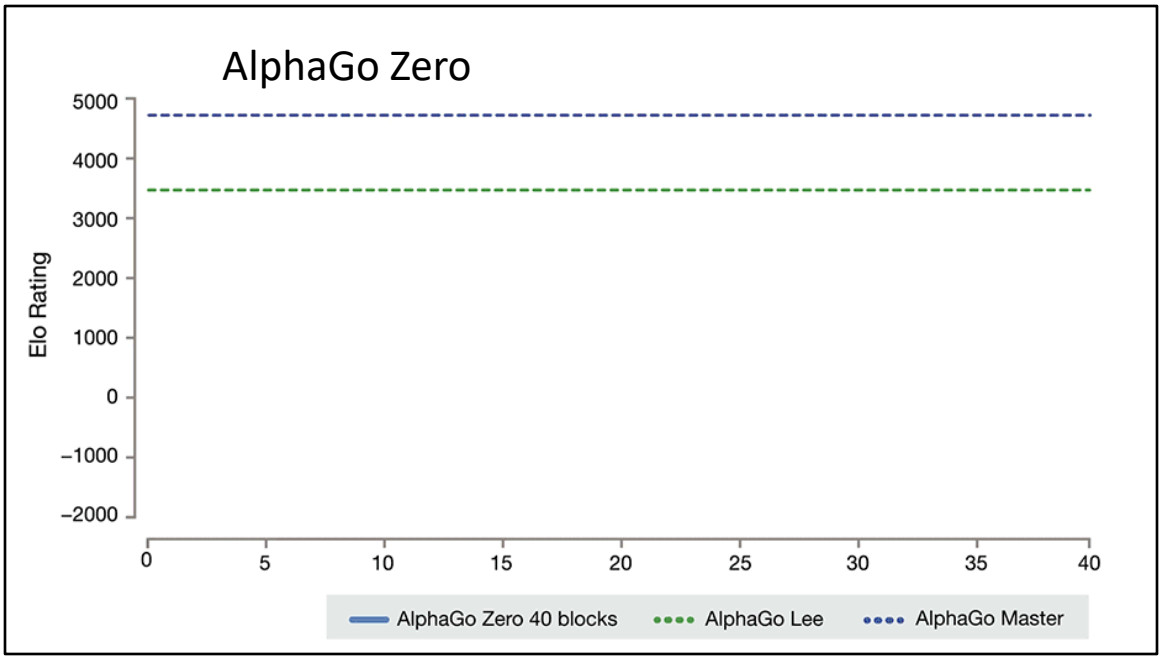
Both reinforcement learning and neural networks are hard to train. DeepMind introduces two tricks to make training DQN stable. The first trick is experience replay, which stores the previous actions, states and rewards, and provide them again for later training. This trick allows DQN to “review” important actions. The second trick is using off-policy strategy, which separates learning into the behavior policy and target policy, and updates the target network every 1000 steps.

強化學習和類神經網絡都很難訓練。 DeepMind引入了兩個技巧來使DQN可以穩定訓練。 第一個技巧是經驗重播，將以前的動作，狀態和獎勵儲存起來，並再次提供它們以供以後的訓練。 DQN可以使用此技巧“複習”重要操作。 第二個技巧是使用off-policy，將學習分成動作網路跟目標網路，並每1000步更新一次目標網絡。



AlphaGo is the most famous success story of deep reinforcement learning. In 2016, Alpha won the five-game Go match against the 18-time world champion Lee Sedol, and lost only one game, which is the last game won by humans. After that AlphaGo has swept the human opponents.

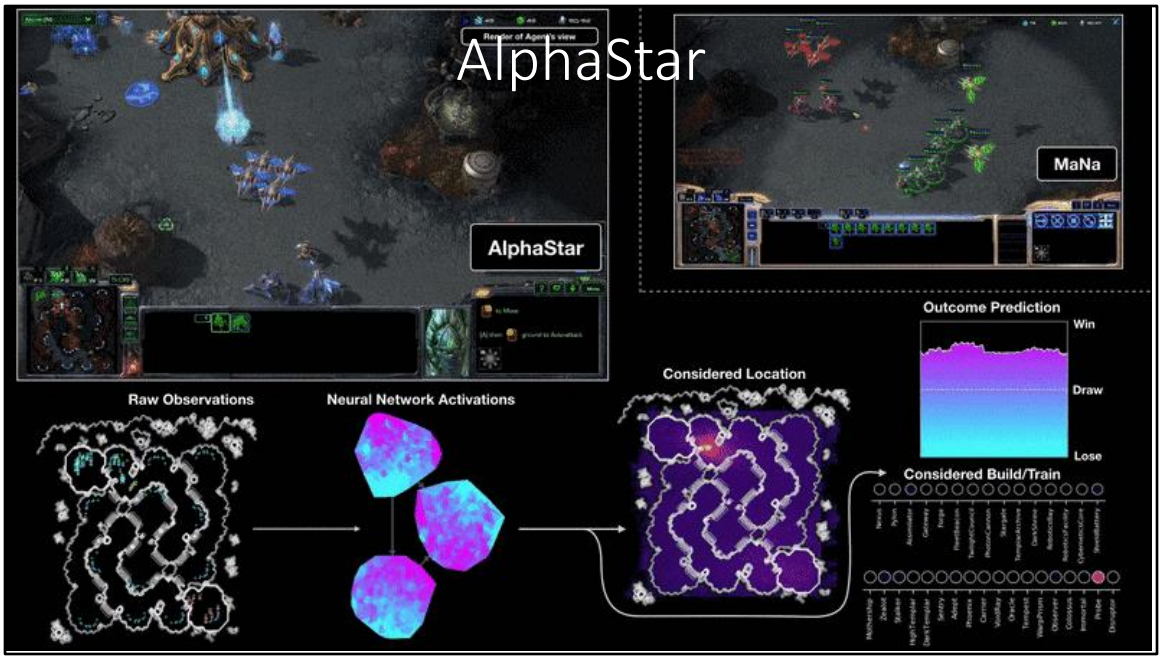
AlphaGo是深度強化學習最著名的成功故事。 2016年，阿爾法在與18屆世界冠軍李·塞多爾（Lee Sedol）的五場圍棋比賽中獲勝，並且只輸了一場比賽，這是人類贏得的最後一場比賽。 之後，AlphaGo席捲了人類對手。



DeepMind continue pushing the limit after beating the human world champion. In Oct. 2017, DeepMind introduced AlphaGo Zero, which learns Go without referring any human play records.
Eventually, no one can understand the strategies behind the moves of AlphaGo Zero, which are beyond human comprehension.

DeepMind在擊敗人類世界冠軍後繼續挑戰極限。 2017年10月，DeepMind推出了AlphaGo Zero，可在不參考任何人為演奏記錄的情況下學習Go。最終，沒人能理解AlphaGo Zero行動背後的策略，這些策略超出了人類的理解力。

AlphaStar

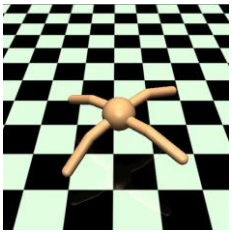


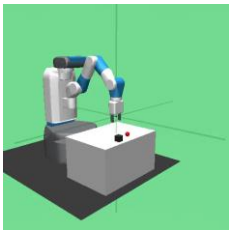
The latest work from DeepMind is AlphaStar, which learns to play the famous real-time strategy game StarCraft. Although StarCraft is easier to play than Go for most people, it is a partially observable game and hard for reinforcement learning. In the beginning of the game, the map is covered with “war fog”, and each player can only see the neighborhood of his own troops. On the contrary, Go is a fully observable game that all opponents’ moves can be viewed. Therefore, StarCraft is actually more difficult than Go.

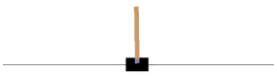
DeepMind的最新作品是AlphaStar，它學會了玩著名的實時策略遊戲《星際爭霸》。 儘管對於大多數人來說，《星際爭霸》比《圍棋》更容易玩，但它是部分可觀察的遊戲，很難進行強化學習。 在遊戲開始時，地圖上覆蓋著“戰爭迷霧”，每個玩家只能看到自己部隊的附近地區。 相反，圍棋是一款完全可觀察的遊戲，可以查看所有對手的舉動。 因此，《星際爭霸》實際上比Go更難。

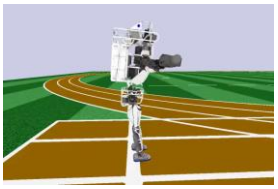
OpenAI Gym

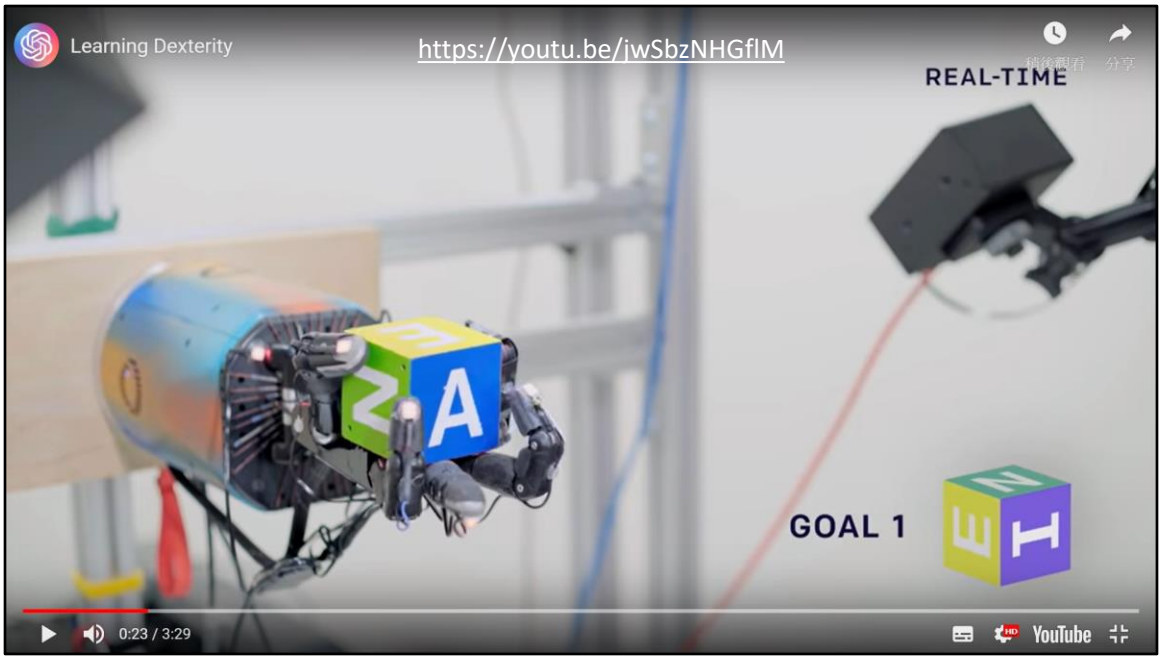
- https://gym.openai.com/envs/









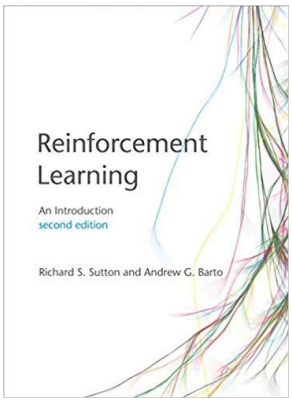

Learning Dexterity
https://youtu.be/jwSbzNHGflM
REAL-TIME
GOAL 1
0:23 / 3:29
YouTube

References

- David Silver, “Tutorial: Deep Reinforcement Learning,” Google DeepMind
- MIT Deep Learning, https://deeplearning.mit.edu/
- Arthur Juliani, “Simple Reinforcement Learning with TensorFlow”

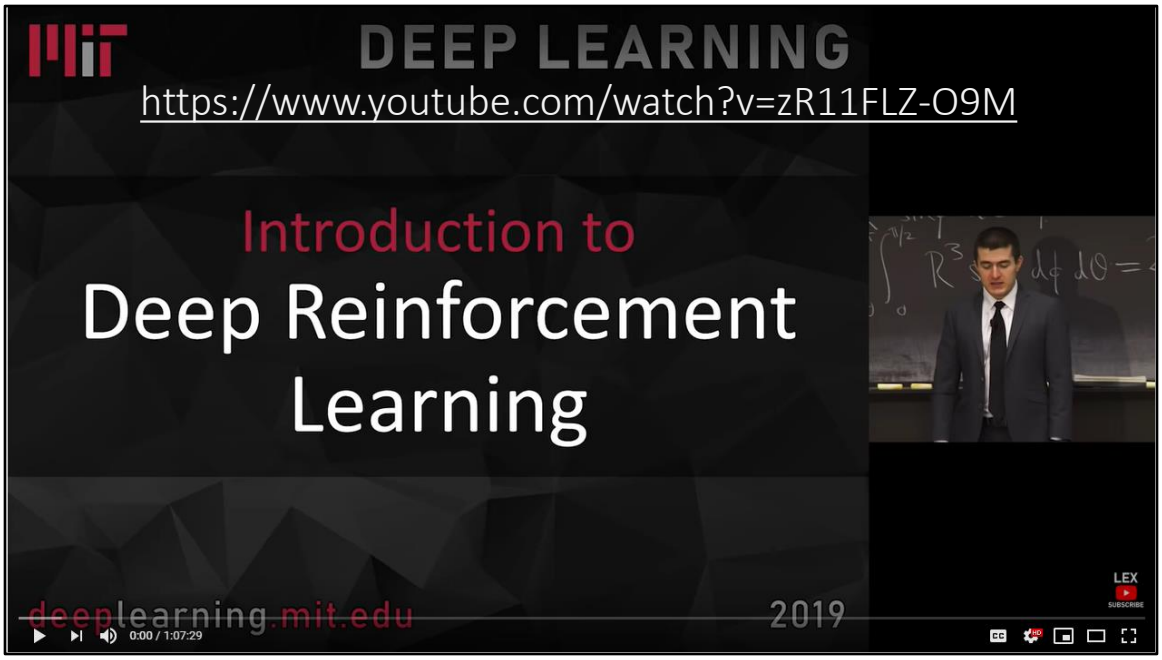
Sutton: Reinforcement Learning

- Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction,” 2nd edition, Nov. 2018



https://en.wikipedia.org/wiki/Richard_S._Sutton

Richard S. Sutton is a Canadian computer scientist. Currently, he is a distinguished research scientist at DeepMind and a professor of computing science at the University of Alberta. Sutton is considered[1] one of the founding fathers of modern computational reinforcement learning, having several significant contributions to the field, including temporal difference learning and policy gradient methods. (Wikipedia)


DEEP LEARNING
https://www.youtube.com/watch?v=zR11FLZ-O9M
Introduction to
Deep Reinforcement Learning
deeplearning.mit.edu
2019
0:00 / 1:07:29
LEX
SUBSCRIBE