Reinforcement Learning

Data Science Africa 2018

Abuja, Nigeria (12 Nov - 16 Nov 2018)

Chika Yinka-Banjo, PhD

University of Lagos

Nigeria

Ayorkor Korsah, PhD

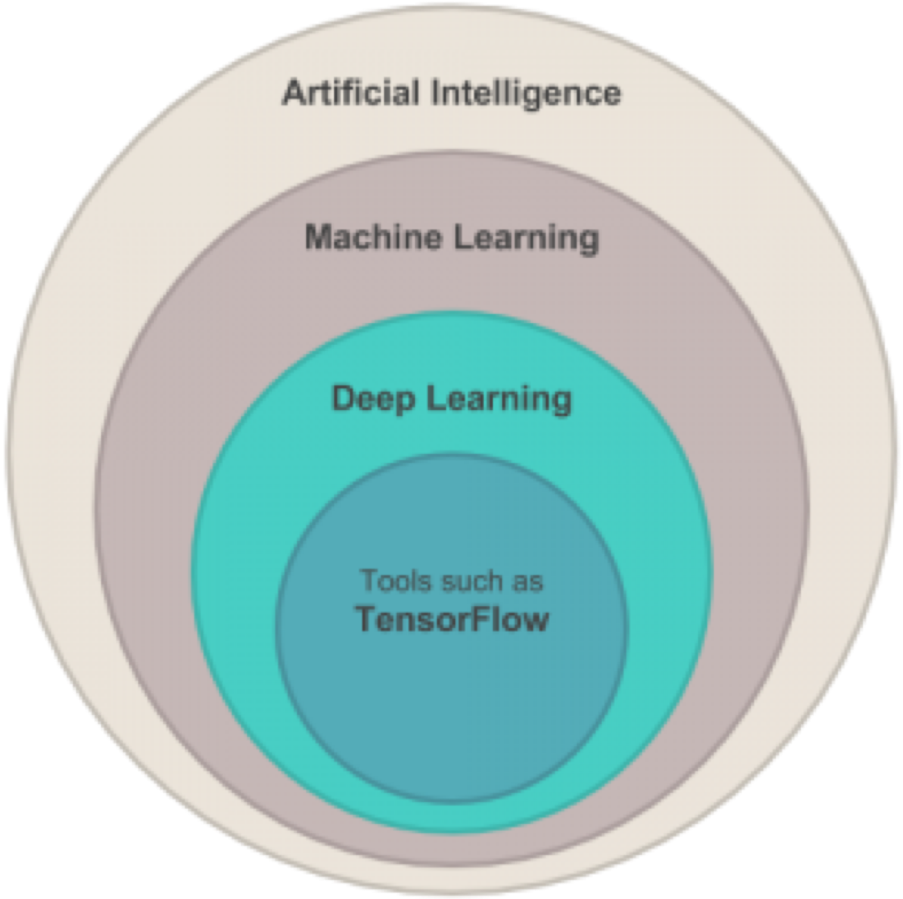
Ashesi University

Ghana

Outline

- Introduction to Machine learning
- Reinforcement learning definitions
- Example reinforcement learning problems
- The Markov decision process
- The optimal policy
- Value function & Q-value function
- Bellman Equation
- Q-learning
- Building a simple Q-learning agent (coding)
- Recap
- Where to go from here?

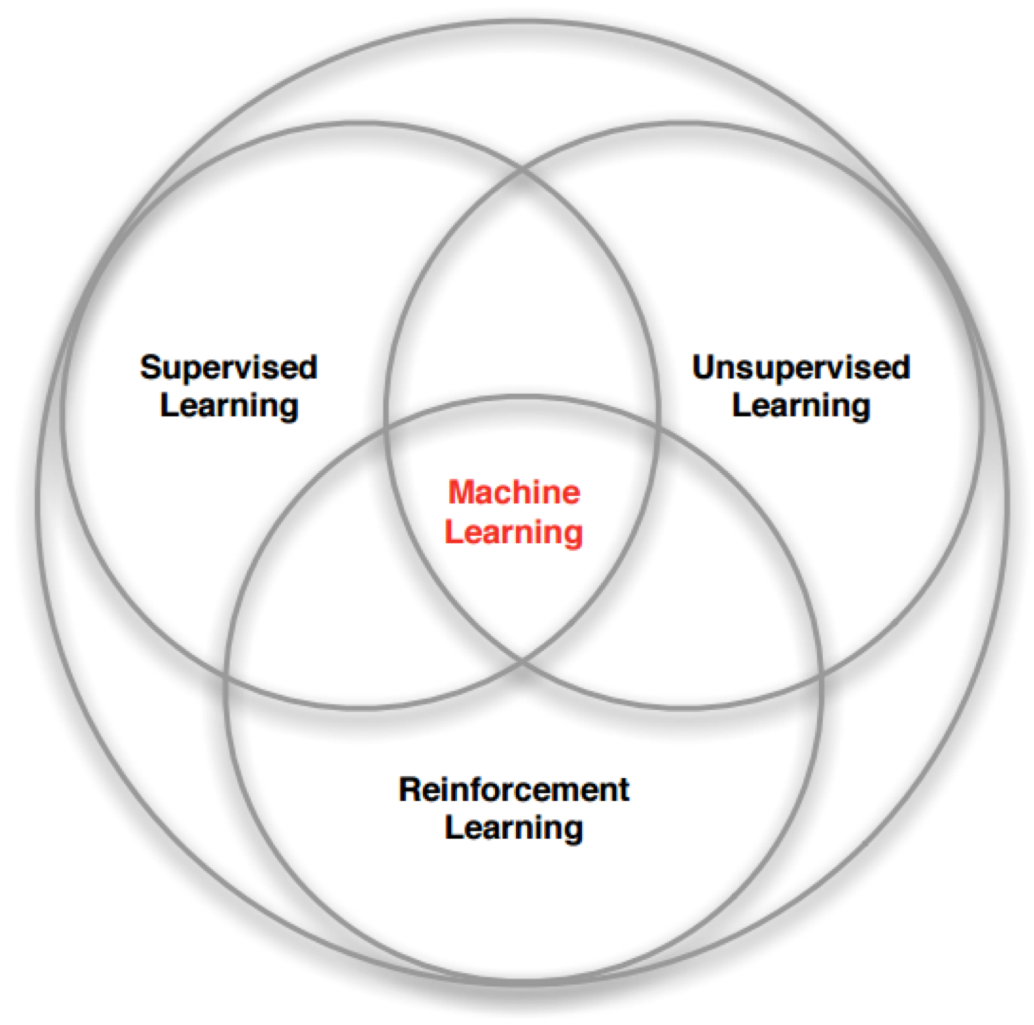
Introduction to Machine learning



Branches of AI

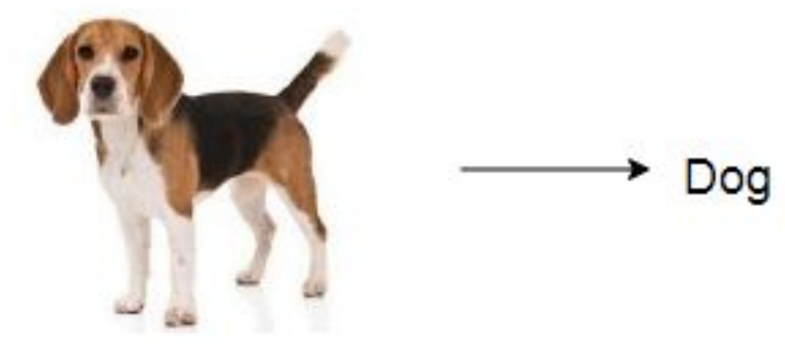
- Artificial Intelligence (AI) is the study and design of Intelligent agents.
- An Intelligent agent can perceive its environment through sensors and it can act on its environment through actuators.
- E.g. **Agent**: Humanoid robot
- **Environment**: Earth?
- **Sensors**: Camera, tactile sensor etc.
- **Actuators**: Motors, grippers etc.
- Machine learning is a subfield of Artificial Intelligence

Introduction to Machine learning



- Machine learning techniques learn from data without being explicitly programmed to do so.
- Machine learning models enable the agent to learn from its own experience by extracting useful information from feedback from its environment.
- Three types of learning feedback:
    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning

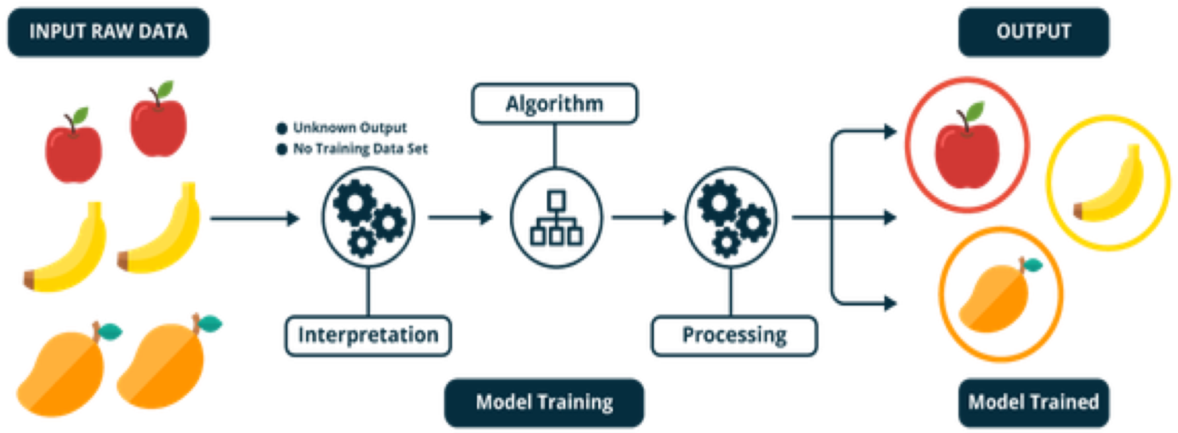
Supervised learning



Supervised learning task in the form of classification

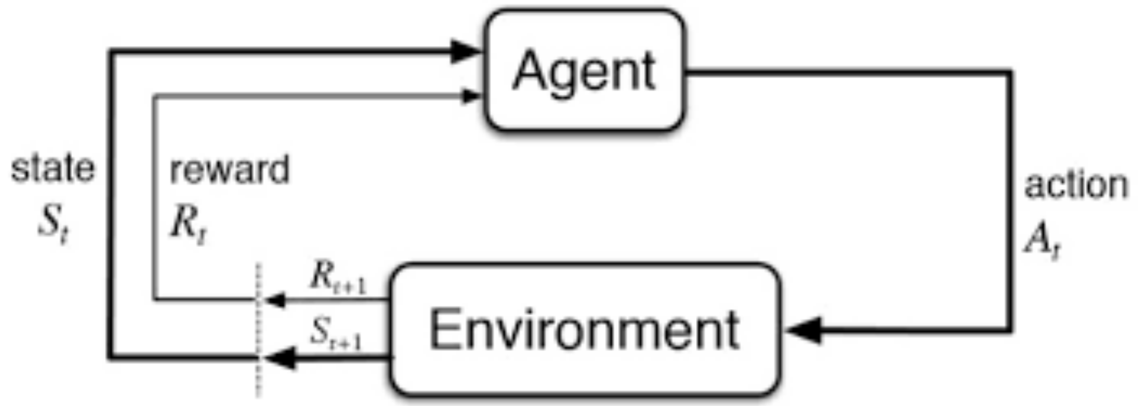
- Supervised learning: the machine learning model is trained on many labelled examples of input-output pairs.
- Such that when presented with a novel input, the model can estimate accurately what the correct output should be.
- Data(x, y): x is input data, y is label
- Goal: learn a function to map x -> y
- Examples include; Classification, regression object detection, image captioning etc.

Unsupervised learning



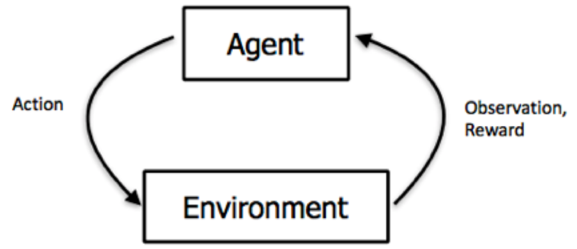
- Unsupervised learning: here the model extract useful information from unlabeled and unstructured data.
- Data: x (raw data)
- Goal: learn an underlining structure in the data that is not explicitly defined
- Examples include; Clustering, density estimation, dimensionality reduction, etc.

Reinforcement learning



- Reinforcement learning involves problems where an **agent** interacts with an **environment** that provides numeric reward signals
- Goal: learn how to take **actions** in a given **state**, in order to maximize the **reward**.

Reinforcement Learning



- Learning by interacting with an environment
- Trial-and-error learning: the agent takes actions and gets feedback through a numeral reward/penalty)

Action 1: try taking hot bread from oven **without** gloves

Action 2: take hot bread from oven **with** gloves

Reward: **-100**

Reward: **+100**

Supervised Learning versus Reinforcement Learning

Supervised Learning	**Reinforcement Learning**
Learns a function, $f: X \rightarrow Y$ that maps X (input) to Y (e.g. class label)	Learns a policy function $\pi: S \rightarrow A$ that maps from states to actions
We have labelled training data that indicates the correct Y for given X	We don't have "labelled" training data that indicates the correct action to take for a given state
Agent learns directly from labelled data	Agent learns indirectly from time-delayed rewards collected through experience (trial-and-error)

Reinforcement learning definitions

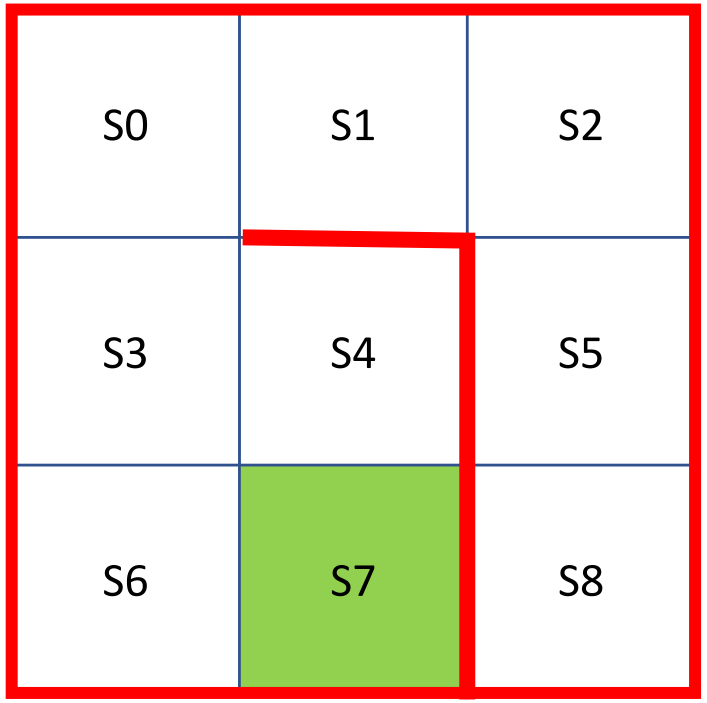
Let’s define the key terms in reinforcement learning.

- **Agent**: An agent takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is run by the agent. In life, the agent is you.
- **Actions (A)**: A is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents choose among a list of possible actions.
- **Environment**: The world through which the agent moves. The environment takes the agent’s current state and action as input, and returns as output the agent’s reward and next state.
- **State (S)**: A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes.
- **Reward (R)**: A reward is the feedback by which we measure the success or failure of an agent’s actions.

Reinforcement learning definitions contd....

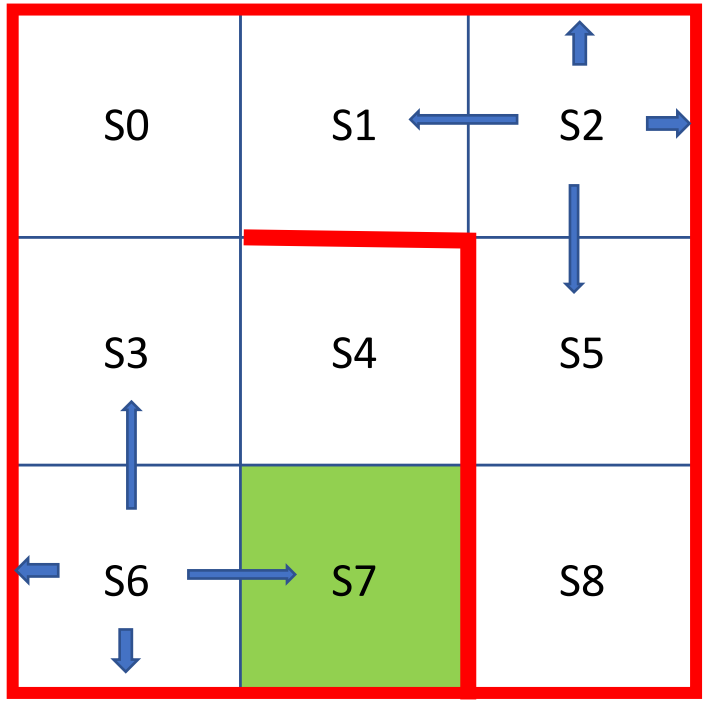
- **Discount factor:** The discount factor is multiplied with future rewards as discovered by the agent in order to dampen their effect on the agent's choice of action. It makes future rewards worth less than immediate rewards; i.e. it enforces a kind of greediness on the agent. Often expressed with the lower-case Greek letter gamma: γ.
- **Policy (π)**: The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.
- **Value (V):** The expected long-term return with discount, as opposed to the short-term reward R. Vπ(s) is defined as the expected long-term return of the current state under policy π. We discount rewards, or lower their estimated value, the further into the future they occur.
- **Q-value or action-value (Q)**: Q-value is similar to Value, except that it takes an extra parameter, the current action a. Qπ(s, a) refers to the long-term return of the current state s, taking action a under policy π. Q maps state-action pairs to rewards. Note the difference between Q and policy.
- **Trajectory**: A sequence of states and actions that influence those states. From the Latin "to throw across."

Toy Example



- 9 possible states: {S0, S1, …, S8}
- 5 possible actions in each state: {*up*, *right*, *down*, *left*, *stay*}
- Rewards:
  - Attaining the goal state has a reward of +50
  - Trying to walk into a wall has a penalty (negative reward) of -5
  - All other actions have a reward of 0
- By trial-and-error, the agent needs to learn an optimal policy, i.e. what the best action is to take in each state

Toy Example – Examples of Effects of Actions



- Results of actions from S2:

up:	Reward = -5, Ends up still in S2
right:	Reward = -5, Ends up still in S2
down:	Reward = 0, Ends up in S5
left:	Reward = 0, Ends up S1
stay:	Reward = 0, Ends up still in S2

- Results of actions from S6:

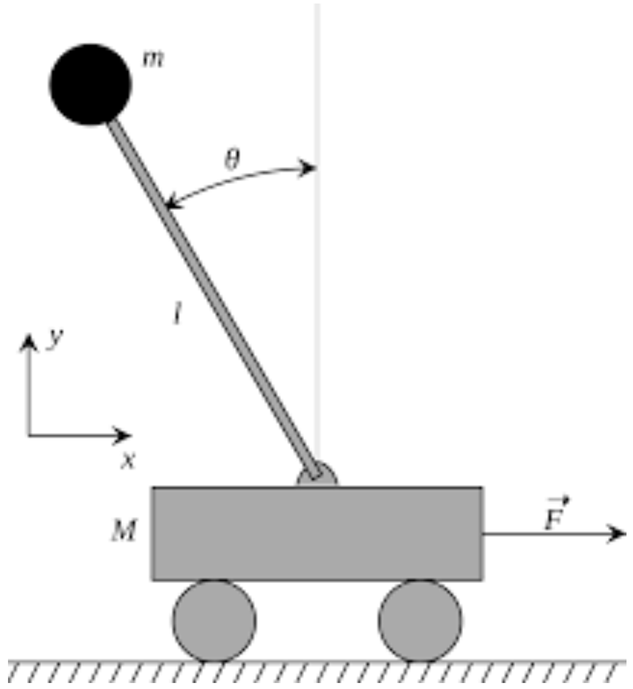
up:	Reward = 0, Ends up still in S3
right:	Reward = +50, Ends up in S7
down:	Reward = -5, Ends up still in S6
left:	Reward = -5, Ends up still in S6
stay:	Reward = 0, Ends up still in S6

Reinforcement Learning Examples

- Pancake flipping robot (2010)

Reinforcement Learning Examples

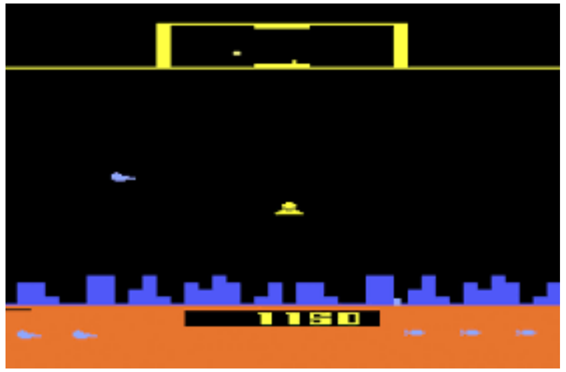
Cart-pole problem

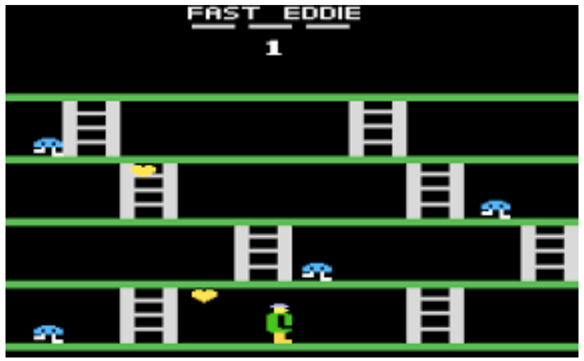


- Objective: Balance a pole on top of a movable cart
- State: angle, angular speed, position, horizontal velocity
- Action: Horizontal force applied on cart
- Reward: 1 at each time step if the pole is upright

Reinforcement Learning Examples

Atari games





- Objective: Complete the game with the highest score
- State: Raw pixel inputs of the game state
- Action: Game controls e.g. up, left, right , down.
- Reward: Score increase or decrease at each time step

Reinforcement Learning Examples

Go (Board game)

- Objective: Win the game!
- State: Position of all the pieces
- Action: Where to put the next piece down
- Reward: 1 if win at the end of the game, 0 otherwise

Markov Decision Process

- The Markov state provides a means of formulating the reinforcement learning process mathematically
- **Markov property:** The current state completely encapsulates the state of the world

Defined by ($\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma$)

$\mathcal{S}$: set of possible states

$\mathcal{A}$: set of possible actions

$\mathcal{R}$*:* distribution of reward given (state, action) pair

$\mathbb{P}$: transition probability i.e. distribution over next state given

γ: discount factor

Markov Decision Process

- At time step t = 0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t = 0 until done:

- Agent selects action a_t
- Environment samples reward r_t ~ R(.| s_t, a_t)
- Environment samples next state s_{t+1} ~ P(.|s_t, a_t)
- Agent receives reward r_t and next state s_{t+1}

- A policy $\boldsymbol{\pi}$ is a function from S to A that specified what action to take in each state
- **Objective:** find the optimal policy $\boldsymbol{\pi}^*$ that maximizes cumulative discounted reward:$\sum_{t>0} \gamma^t r_t$

The optimal policy

- We want to find the optimal policy $\boldsymbol{\pi}^*$ that maximizes the sum of rewards.
- How do we handle the randomness (initial state, transition probability...)?
- Maximize the expected sum of rewards!

- Formally: $\pi^* = \arg max\, \mathbb{E}\, [\, \sum_{t>0} \gamma^t r_t]$ with
  $s_0 \sim p(s_0), \quad a_t \sim \pi(.|\, s_t), \quad s_{t+1} \sim \mathrm{P}(.|s_t, a_t)$

Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots..$

How good is a state?

The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\,[\sum_{t>0} \gamma^t r_t \,|s_0 = s, \pi]$$

How good is a state-action pair?

The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\,[\sum_{t>0} \gamma^t r_t \,|s_0 = s, a_0 = a, \pi]$$

Q-Table

- A table of Q-values for each state-action pair
- Q-value estimates the expected long-term reward for taking a given action from a given state

	Action 1	**Action 2**	**Action 3**
State 1	Q(s1,a1)	Q(s1,a2)	Q(s1,a3)
State 2	Q(s2,a1)	Q(s2,a2)	Q(s2,a3)
State 3	Q(s3,a1)	Q(s3,a2)	Q(s3,a3)
State 4	Q(s4,a1)	Q(s4,a2)	Q(s4,a3)
State 5	Q(s5,a1)	Q(s5,a2)	Q(s5,a3)
State 6	Q(s6,a1)	Q(s6,a2)	Q(s6,a3)
State 7	Q(s7,a1)	Q(s7,a2)	Q(s7,a3)

Bellman Equation for updating the Q-values

R – Reward from taking action *A* in state *S*

- $\boldsymbol{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_{a} Q(S',a) - Q(S,A)\right]}$

S' – New state resulting from taking action *A* in state *S*

A – Chosen action

S – Current state

γ = Discount factor – how important are future rewards? $(0 \leq \gamma \leq 1)$

α = Learning rate – how much do we update the q-value in each iteration? $(0 < \alpha \leq 1)$

Equivalent to:

- $\boldsymbol{Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha\left[R + \gamma \max_{a} Q(S',a)\right]}$

Q-learning

We can use the bellman equation as an iterative means of determining the optimal policy. Such a value iteration algorithm is referred to as Q-learning and will be of the form:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Where α is the learning rate ($0 < \alpha \leq 1$)

The Q-learning algorithm is also referred to as temporal difference learning.

Q-learning - Algorithm

1. Initialize Q-values ($Q(s,a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s,\cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s,a) := Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

Q-Learning Algorithm

- Initialize: $Q(s, a)$, for all $s \in S$, $a \in A(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
- Repeat (for each episode):
  - Initialize starting state S
  - Repeat (for each step of episode):
    - Choose action A to take from S using a policy derived from Q
    - Take action A, observe reward R, and new state S'
    - Update Q-value: $\boldsymbol{Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a} Q(S', a) - Q(S, A)\right]}$
    - Update current state: $\mathrm{S} \leftarrow \mathrm{S}'$
  - until S is terminal

Sometimes the agent ***explores*** - tries random actions

Sometimes the agent ***exploits*** – chooses the best action based on its prior experience

Building a simple Q-learning agent (Coding session)

Source Code: https://github.com/chikayinkabanjo/Reforcement-Learning

Development Environment: https://colab.research.google.com/

Recap

- Reinforcement learning as a branch of machine learning, involves an agent-environment relationship where the agent chooses actions that maximize a reward within that environment
- A policy is the strategy the agent employs to determine the next action given its current state. The goal is to optimize the policy that returns the maximum cumulative future reward.
- The Q-value function is a key function for estimate the maximum expected reward given a state-action pair
- The Q-value function optimizes and updates the agents policy
- The Q-learning algorithm is an iterative algorithm for estimating the Q-value for state-action pairs.

Where to go from here?

- Function approximators: The Q-learning algorithm is computationally infeasible for very high dimensional state-action spaces. We need a function approximator such as a deep neural network, to estimate Q^*(s, a).
- This will introduce the subject of Deep Reinforcement Learning using deep learning techniques.

Further Reading

- Reinforcement Learning: An introduction by Sutton and Barto
- Deep Reinforcement Learning: An overview by Yuxi Li
- Jens Kober, J. Andrew Bagnell and Jan Peters, "Reinforcement learning in robotics: A survey", *International Journal of Robotics Research*, 32(11) 1238–1274
- Florentin Woergoetter and Bernd Porr (2008), Scholarpedia, 3(3):1448 http://www.scholarpedia.org/article/Reinforcement_learning
- Satwik Kansal and Brendan Martin, "Reinforcement Q-Learning from Scratch in Python with OpenAI Gym", https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

Thank you