# Lecture\_5b

Hello and welcome to the second lecture in this week. We are going to be discussing reinforcement learning algorithms. in this lecture video, we are going to look at the recap of reinforcement learning, the reward and common reinforcement learning algorithms.

The three popular broad subfields or subsets of machine learning are: supervised learning, unsupervised learning and reinforcement learning. All the subfields of machine learning premised on making machines, specifically computers to learn by teaching them what comes naturally to humans or animals. In contrast to supervised learning and unsupervised learning frameworks, which are implemented using static data, reinforcement learning is implemented with data from a dynamic environment. The goal of reinforcement learning is not to cluster data, as carried out in unsupervised learning, nor label data, as carried out in supervised learning. Rather, the goal of reinforcement learning is to find the best sequence of actions that will generate the optimal outcome or reward, and to achieve this goal, reinforcement learning uses a piece of software called an agent to explore, interact with and learn from the environment. Note that an agent could also be a hypothetical entity reinforcement learning.

Within the agent in reinforcement learning, a function takes in the state observations, that is the inputs and maps them to the actions, that is the outputs. This function is called the policy and given a set of observations, the policy decides which actions to take. Reinforcement learning algorithms are primarily used to update the policy based on the actions taken, the observations from the environment and the amount of reward collected as illustrated. The agent employs reinforcement learning algorithms to learn the optimum policy as it interacts with the environment. In reinforcement learning, the reward helps the reinforcement learning algorithms to understand when the policy is getting better and ultimately it converges to the desired optimal. How the agent is rewarded depends on the environment, that is the settings and features or the environment and the specific tasks the agent is designed to accomplish. Take for example, the agent could designed to defeat a human opponent in a game of pong. It could also be designed to ensure the safe navigation of a drone back to a landing site. It could also be designed to buy units of stock with the most yields and so on. Generally, in reinforcement learning, the reward is a function that produces a scalar number, that is the value which represents the goodness of an agent being in a specific state and taking a specific action. The reward function tries to maximise this value and from a mathematical cost function viewpoint, it can be viewed as a negative minimisation cost function because negative minimisation is also maximisation as illustrated in this mathematical relations. In reinforcement learning, the rewards can assume various forms.

We can have sparse rewards, many goals to be incentivised do not occur until several sequences of actions. We could also have instantaneous rewards that are given for every time step. We can have delayed rewards which are given at the very end of an episode occurring after long periods of time. Rewards can also be calculated from a non-linear function or even deduced hundreds or even thousands of parameters. The derivation or deduction for the reward totally depends on what it takes to effectively and efficiently train the agent. In reinforcement learning, sparse rewards are often undesired in machine learning. Takes for instance, an agent in a reinforcement learning-based autonomous drone could have a reward function that rewards the drone once it gets to a specific geolocation indicating the region of its landing site. However, it could be that the drone tried several actions and flight paths without getting any rewards before reaching the landing site, and consequently did not learn anything in the process. In reinforcement learning, sparse rewards can be improved using what we call reward shaping. A reward shaping simply means providing small or intermediate rewards that guide the agent. Looking at the example of the reinforcement learning based autonomous drone again, the agent in this case can be given intermediate rewards as it flies along the correct flight path. Note that if the rewards are not properly shaped in reinforcement learning, this may cause the agent to converge on a solution that is not ideal, even if that solution yields the most rewards. Take for instance, the reinforcement learning-based autonomous drone may decide to follow a shortcut facing many obstacles, but this shortcut gives the most cumulative intermediate rewards to reach the landing site and the drone may also crash land to get the reward associated which reaching the landing site.

Reward shaping can also be used to inject domain-specific knowledge into the agent. For example, in algorithmic trading extra useful information such as other market participants' trading strategies, confidential information from companies, similar stock market behaviours, rumours, advice from experts and others can be used to reward the agents if and when actions are taken in consonance with such extra useful information. As earlier defined, in reinforcement learning the reward is the instantaneous benefit the agent receives for being in a specific state and taking a specific action. The total rewards an agent expects to receive from a state and onwards into the future is called value. Primarily, reinforcement learning algorithms work by guiding the agent's exploration and exploitation of the dynamic environment. Exploitation begs the question: should the agent work by choosing actions that collect the most rewards that it already knows about? In reinforcement learning pure exploitation will only make the agent not to receive additional information about environment and eventuality, this may lead the agent to coverage to a suboptimal policy. In reinforcement, learning, exploration begs the question: should an agent choose actions that explore parts of the environment, but are still unknown? Note that pure exploration will make the agent to spend a lot of time in the bigger portion of the environment and this may help or lead to finding a global optimum for the policy.

However, this could also slow down the learning rate so much so that no sufficient solution is found within a reasonable time. Reinforcement learning algorithms offer a simple, effective and efficient way to balance exploration and exploitation of the time of the dynamic environment to find the most optimal policy. In most practical cases, it's not always very obvious where to set the balance between exploration and exploitation of the environment throughout the learning process, so that the agents converges to a near-optimum policy within the time allocated for learning. In general, agents should explore more at the initialisation or start of learning and gradually transition to more of exploitation roles by the end. There are many algorithms or computational intelligence techniques which can be employed to optimise the policy in reinforcement learning. Depending on the agent's knowledge of the dynamic environment, these algorithms can be grouped as: model-free or model-based, and dependent on how the agent's policy improved or optimised, these algorithms can be group as: on-policy learning methods and off-policy learning methods. For on-policy learning methods, they attempt to evaluate or improve the policy that is used to make decisions, that is they are based on actions taken whereas off-policies learning methods evaluate or improve the policy different from that used to generate the data, that is a different policy is used for value evaluation than what is used to select the next action.

Some of the most common algorithms for reinforcement learning include but are not limited to q-learning, which is an off-policy learning methods, state-action-reward-state-action (SARSA), which is an on-policy learning method. Q-learning is used to learn the value of an action in a particular state and it is model-free. Given a finite Markov decision process, q-learning works by finding an optimal policy in the sense of maximising the expected value of the total reward over any and all successive steps, starting from the current state. In mathematical sciences, a Markov decision process is simply a discrete-time stochastic process which provides a mathematical framework for modelling decision making in situations where outcomes are partly stochastic, that is random and partly under the control of a decision maker, such as an agent in reinforcement learning. Q-learning can adequately identify an optimal action-selection policy for any given factored Markov decision process,

given infinite exploration time and a partly-random policy. The q in q-learning refers to the function that the q-learning algorithm computes the expected rewards for an action taken in a given state. Q-learning premises on value iteration by deciding the next action so as to maximise the next state's q value, as opposed to following the current policy and this makes it an off-policy learning method. Policy iteration. This runs a loop between policy evaluation and policy improvement. Policy evaluation estimates the value function with the greedy policy obtained from the last policy improvement. Policy improvement updates the policy with the action that maximises the value function for each of the state and the update equations of policy improvement are based on their Bellman equation. This keeps iterating till convergence. Value iteration. Value iteration contains just one component and updates the value function based on the optimal Bellman equation.

After the iteration converges, the optimal policy is straight-forwardly derived applying an argument-max function for all of these states. Both policy iteration and value iteration required the knowledge of the transition probability considering the states. In q-learning, the value, that is the q-value to be maximised is derived using the Bellmaan equation. The q-value is initialised to an arbitrary estimate and at each time instant it is updated as a sum of three factors. The first factor is the current q-value, weighted by the learning rate, which ranges from 0 to 1. Values of the learning rate close to 1 lead to faster changes in q. The second factor is the instantaneous reward to obtain, if an instantaneous action is taken when in an instantaneous state weighted by the learning rate. And the third factor is the maximum reward that can be obtained from an estimated optimal future state weighted by the learning rate and discount factor. SARSA. State-action-reward-state-action algorithm is a slight variation of the q-learning algorithm, also called modified connectionist q-learning (MCQ-L). SARSA is a off-policy learning method because a SARSA agent interacts with the dynamic environment and updates the policy based on actions taken. In the SARSA algorithm, similar to q-learning, the main function for updating the q-value depends on: the current state of the agent, the action agent chooses. the reward the agent gets when choosing this action, the state that the agent enters after taking that action and finally the next action the agent chooses in its new state.

The q-value for a state action in the SARSA algorithm is updated by an error, adjusted by the learning rate. In this video, we've looked at a recap of reinforcement learning, we've discussed the rewarding in reinforcement learning and we've discussed some common reinforcement learning algorithms, particularly q-learning and SARSA.