Deep Reinforcement Learning (DRL)

Deep reinforcement learning (DRL) uses deep learning and reinforcement learning principles in order to create efficient algorithms that can be applied on areas like robotics, video games, finance and healthcare. Implementing deep learning architecture (deep neural networks or etc.) with reinforcement learning algorithms (Q-learning, actor critic or etc.), a powerful model (DRL) can be created that is capable to scale to previously unsolvable problems.[2] That is because DRL usually uses raw sensor or image signals as input as can be seen in DQN for ATARI games[3] and can receive the benefit of end-to-end reinforcement learning as well as that of convolutional neural networks.

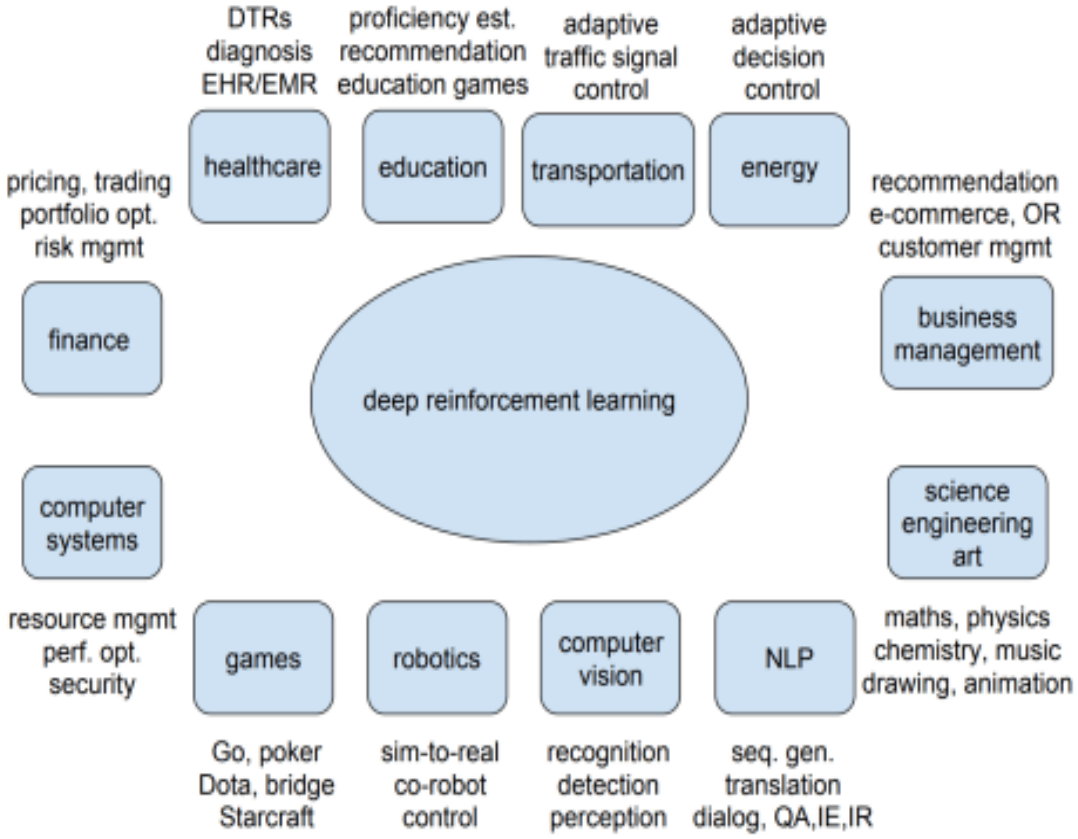


Figure 4: Deep Reinforcement Learning Applications

Here are two of the most commonly cited Deep RL use cases:

- Google's Cloud AutoML
- Facebook's Horizon Platform

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. This field of research has been able to solve a wide range of complex decision-making tasks that were previously out of reach for a machine. Thus, deep RL opens up many new applications in domains such as healthcare, robotics, smart grids, finance, and many more. This manuscript provides an introduction to deep reinforcement learning models, algorithms and techniques. Particular focus is on the aspects related to generalization and how deep RL can be used for practical applications. We assume the reader is familiar with basic machine learning concepts.

RL Agent-Environment

A reinforcement learning task is about training an **agent** which interacts with its **environment**. The agent arrives at different scenarios known as **states** by performing **actions**. Actions lead to rewards which could be positive and negative.

The agent has only one purpose here – to maximize its total reward across an **episode.** This episode is anything and everything that happens between the first state and the last or terminal state within the environment. We reinforce the agent to learn to perform the best actions by experience. This is the strategy or **policy**.

Let's take an example of the ultra-popular **PubG** game:

- The soldier is the agent here interacting with the environment
- The states are exactly what we see on the screen
- An episode is a complete game
- The actions are moving forward, backward, left, right, jump, duck, shoot, etc.

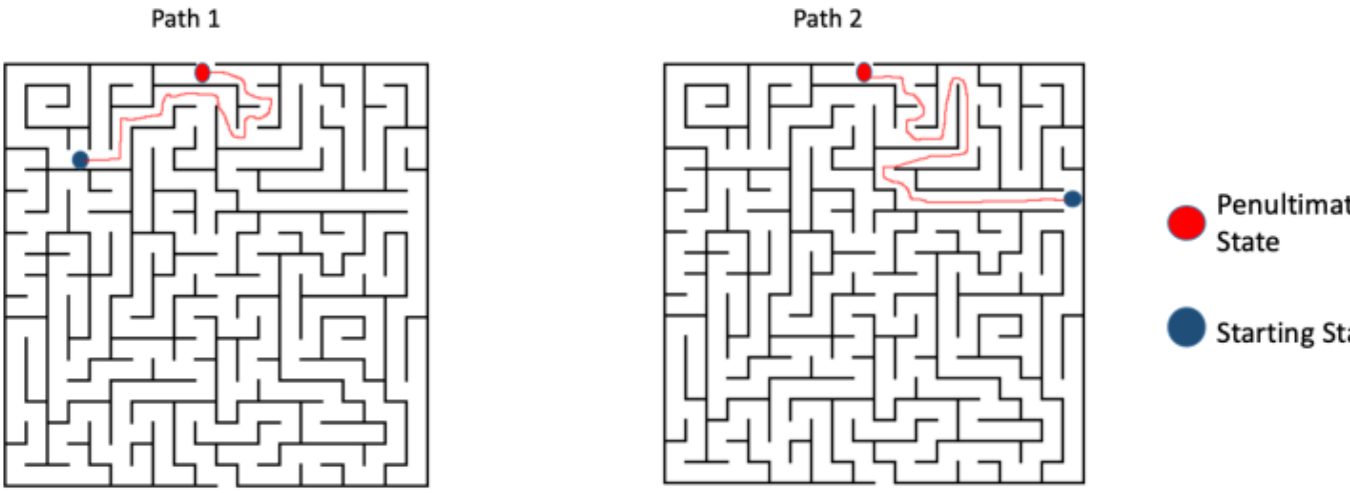
- Rewards are defined on the basis of the outcome of these actions. If the soldier is able to kill an enemy, that calls for a positive reward while getting shot by an enemy is a negative reward

Now, in order to kill that enemy or get a positive reward, there is a sequence of actions required. This is where the concept of delayed or postponed reward comes into play. The crux of RL is learning to perform these sequences and maximizing the reward.

Markov Decision Process (MDP)

An important point to note – each state within an environment is a consequence of its previous state which in turn is a result of its previous state. However, storing all this information, even for environments with short episodes, will become readily infeasible.

To resolve this, we assume that each state follows a Markov property, i.e., each state depends solely on the previous state and the transition from that state to the current state. Check out the below maze to better understand the intuition behind how this works:



Now, there are 2 scenarios with 2 different starting points and the agent traverses different paths to reach the same penultimate state. Now it doesn't matter what path the agent takes to reach the red state. The next step to exit the maze and reach the last state is by going right. Clearly, we only needed the information on the red/penultimate state to find out the next best action which is exactly what the Markov property implies.