# Machine Learning (CS60050) Spring 2020-2021
# Instructor: Dr. Aritra Hazra

Scribed by: Urvashi Kumawat(20CS60R51)
Sandeep Shahu(20CS60R50)

31 March 2021

## 1 Introduction to Reinforcement Learning

We have seen two types of Learning so far that we generally use are as follows:

**1: Supervised Learning**

There is always a teacher who is going to guide us means given a input we are also having the output associated with it, formally In a supervised learning model, the algorithm learns on a labeled data-set, providing an answer key that the algorithm can use to evaluate its accuracy on training data.

**2: Unsupervised Learning**

There is no as such teacher who can guide us instead we need to find out certain kind of Clustering here we are provided the unlabeled data that the algorithm tries to make sense of by extracting features and patterns on its own.

### 1.1 Reinforcement learning (RL)

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.This is mixture of the above two learning, we don't have immediate feedback but not the case that we don't have any instead we have a delayed feedback over here more formally this is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

Reinforcement Learning: It depicts this delayed feedback that get reinforced into different positions of the game and therefore we try to accumulate the cumulative feedback to maximize our feedback.

it is going to follow sequence of steps to reach to the goal and in ML we call it as a Policy, for Example in a Game at the board position in chess we sense the state and take the action now here we won't get a immediate reward for this because there can be other move that can give more reward so here by sensing a state and taking action we get reward from looking at the board position, for

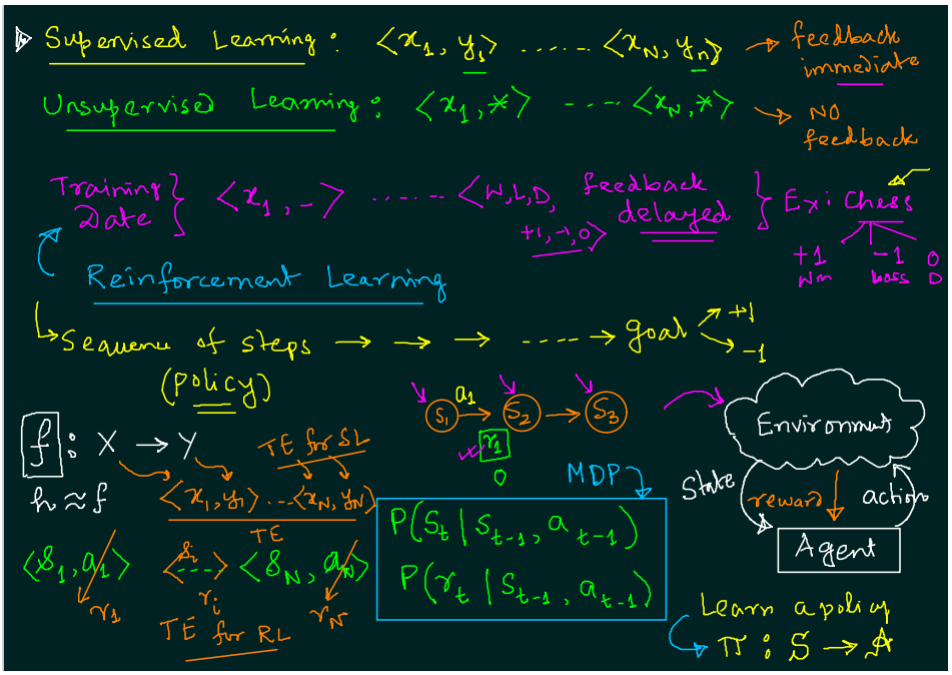


Figure 1: Reinforcement Learning with MDP setup

formalise such kind of setup we have set of environment states through which environment moves $S_1 \rightarrow S_2 \rightarrow S_3$

It is going to follow MarMarkov Decision Process(MDP)Formally, an MDP is used to describe an environment for reinforcement learning, where the environment is fully observable.

$P(s_t|s_(t-1), a(t-1))$

$P(r_t|s_(t-1), a(t-1))$

now under this setup our challenge is to learn a policy, so that we could guide the agent to reach its goal that means we want to learn the policy: $\Pi : S \rightarrow A$

Now as in supervised Learning we have data-set with the output $< X_1, Y_1 >$ .... $< X_n, Y_n >$ here also we have the same structure $< S_1, a_1 >$ ....., $< S_n, a_n >$ but instead of actual output we have a immediate reward $r_1, r_2..r_n$ associated with the current action that guides us to reach to the actual goal,

here if we can find out all possible paths then it will be a search problem instead but that is not possible for infinite state space like in chess.

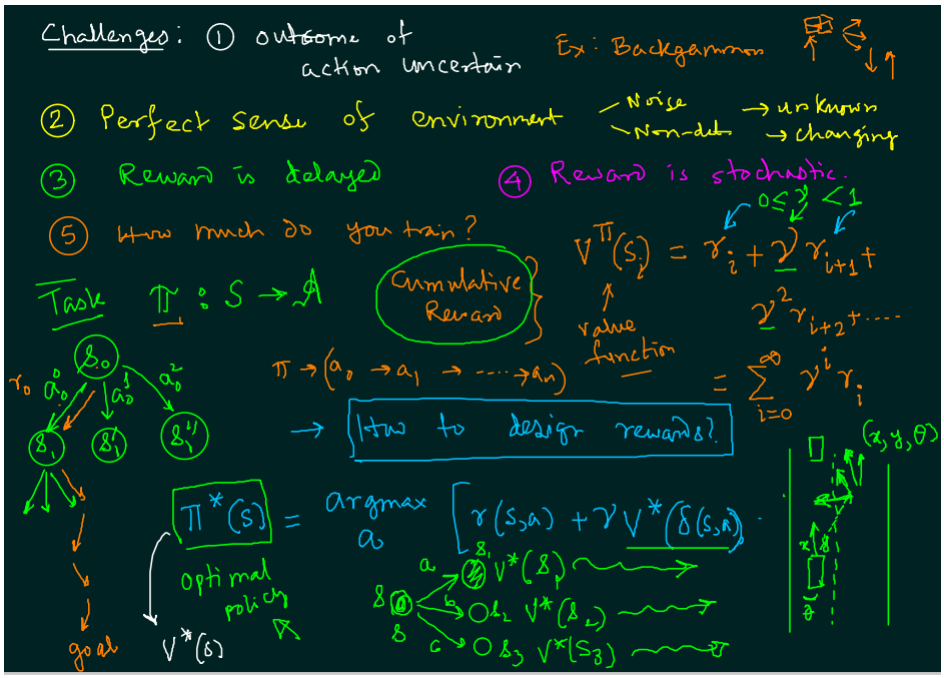


Figure 2: RL challenges and Policy

**These are the challenges we have in RL:**

1: Outcome of the action is uncertain. (Ex: in Backgammon we make a movie followed by a dice)

2: Perfect sense of Environment (means how environment will react over it)

3: Reward is Delayed (Ex in chess we play but nothing captured by that action so sometimes reward is Stochastic as well)

4: Reward is Stochastic.

5: How Much you train.

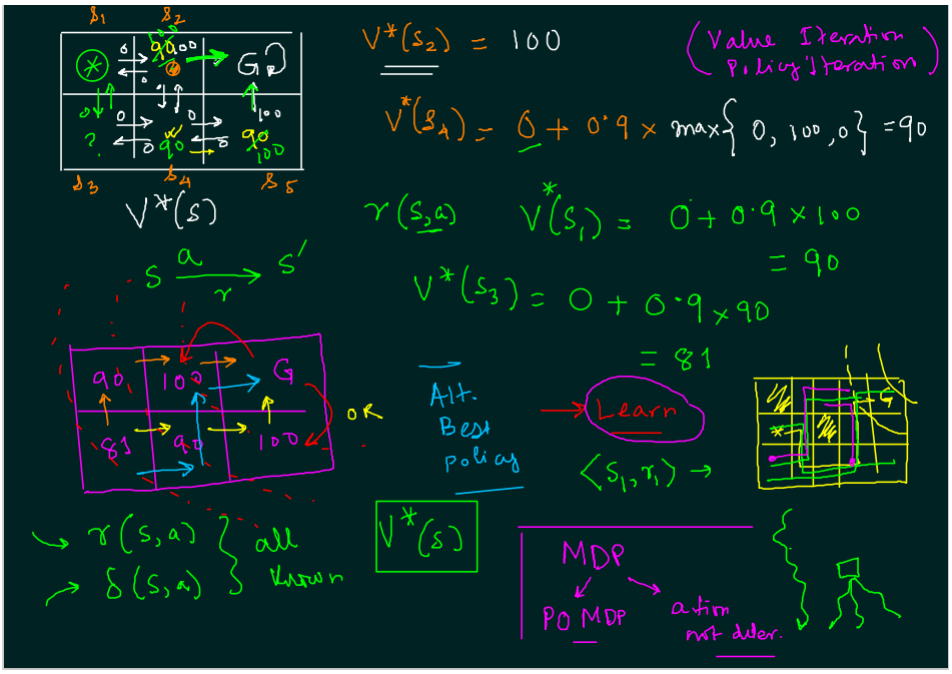


Figure 3: Grid Example of RL

### 1.1.1 Policy: $\{\Pi : S \to A\}$

Pathway to reach the goal, as reward is delayed so some how we need to embed the reward through the pathway, More formally,

A policy defines the learning agent's way of behaving at a given time. policy is a mapping from states of the environment to actions to be taken when in those states.

so We can write this policy as

$\Pi :\to (a_0, a_1 ....., a_n)$ by By taking any action we will get cumulative reward while moving into state $S_i$ that can be written by this value function as follows:

$V^{\pi}(S) = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + ...$

that can be be written precisely as

$= \sum_{i=0}^{\infty} \gamma^i r_i$

we can also write it as: $\Pi^*(S) = \arg\max_a [r(S,a) + \gamma V^*(\delta(S,a)]$

Now we can analyse the value function by this example: In this example we are trying to reach the goal state, here we will use the value function to move into any state which will give us highest cumulative reward, now if we start filling from goal state we keep on filling this by BFT(Breath First Traversal).

as in this figure we can see that for moving into goal state from $S_2$ we will get the reward of 100, now if we calculate the reward of moving into $S_1$ by using this value function,

$V^*(S) = 0 + 0.9 \max\{0, 0, 100\} = 90$

And so on we can fill this entire grid by traversing Breath First, but the constraint for this calculation is that we need to know these two $r(S, a)$ and $\delta(S, a)$ but we in actual infinite state space we can not know this for all and not possible to fill the entire grid.

There comes the learning where we can't have these two parameters $r(S, a)$ and $\delta(S, a)$ for whole state space because the state space is huge over here.

As we can see this grid example where we can just have few states given with the reward as $< S_i, r_i >$and we start from them and try to find the path to Goal and we keep on doing this exploration means we are given the states and set of moves depending on that we need to learn now.

As because of uncertainty of the model, the environment may have problem of going to a certain state not a deterministic flow, it is not always on MDP, it sometimes a partially observed (Where actions are not-deterministic in producing outcome).

### 1.1.2 Q-Learning

This is Most popular RL paradigm, formally, Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards without requiring adaptations. By this example of the Grid we can write these following equations:

$\Pi^*(S) = \arg\max_a [r(S, a) + \gamma V^*(\delta(S, a)]$

here this $r(S, a)$ is immediate reward and $\gamma V^*(\delta(S, a)$ discounted reward that we will make over this transition,

$V(S_1) = r(S_1, a) + \gamma V(S_2) + \gamma^2 V(G)$ and so on we can write this for each state that even have this matrix representation:

$$\begin{bmatrix} V(S_1) \\ . \\ . \\ V(S_5) \end{bmatrix} = \begin{bmatrix} . \\ . \\ R \\ . \end{bmatrix} + \gamma \begin{bmatrix} V \\ V \\ . \\ . \end{bmatrix}$$

here $Q(S, a) \leftarrow$ Best Action 'a' at state S

$\Pi^*(S) = \arg\max_a Q(S, a)$

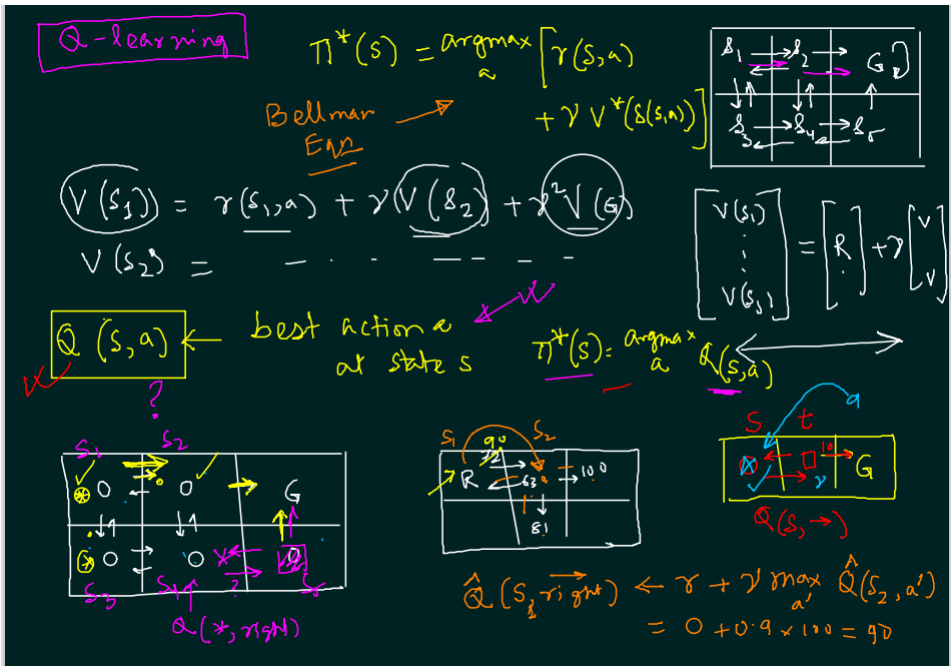


Figure 4: Q-Learning

### 1.1.3 Convergence in Q-Learning

a reinforcement learning algorithm is considered to converge when the learning curve gets flat and no longer increases. Q-Learning has been proven to converge towards the optimal solution.

These are some conditions for proving the convergence:

1: Actions are Deterministic

2: $|r(S,a)| < C$ C is a constant

3: Infinitely often visit of each state

### 1.1.4 Q-Learning Math

$$\hat{Q}(S,a) = r(S,a) + \gamma V^*(\delta(S,a))$$
$$= r(S,a) + \gamma \max_a \hat{Q}(\delta(S,a), a')$$

As we know that $\delta(S,a) = S'$

$\hat{Q}(S,a) = \max_a \hat{Q}(S', a')$ as we can see that this equation for $\hat{Q}$ is a recursive

equation so it is kind of dynamic approach, we recursively keep on exploring and update the reward for state until we converge.

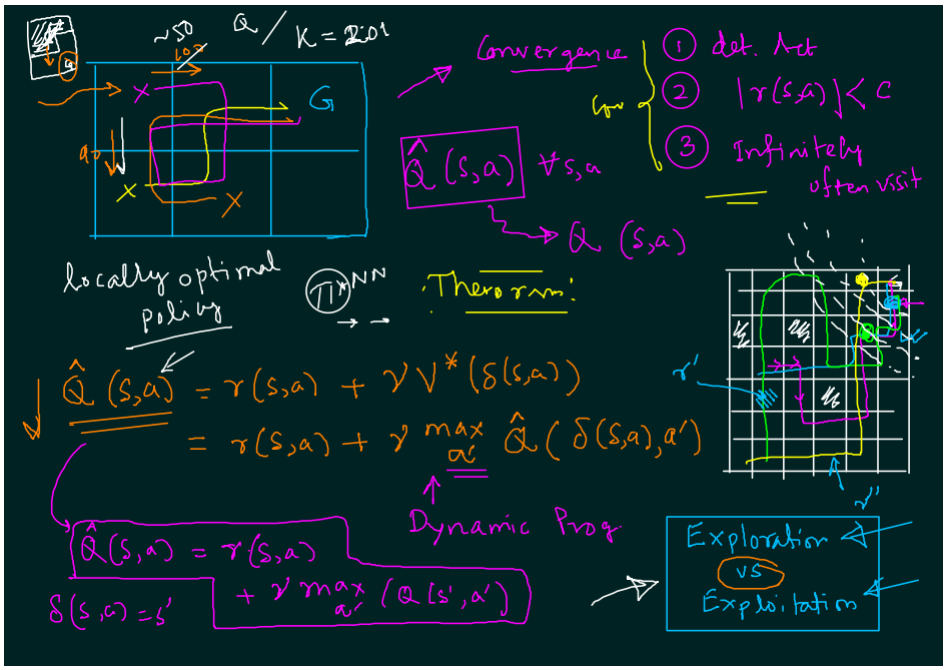


Figure 5: Convergence and Q-Learning Math

### 1.1.5 Exploration vs. Exploitation

The Q-learning algorithm does not specify what the agent should actually do. The agent learns a Q-function that can be used to determine an optimal action. There are two things that are useful for the agent to do:

**exploit** the knowledge that it has found for the current state s by doing one of the actions a that maximizes Q[s,a].

**explore** in order to build a better estimate of the optimal Q-function. That is, it should select a different action from the one that it currently thinks is best.

It may be the case that We won't even explore the portion of the state space that is kind of locally optimal policy, for that we will gradually decrease this chances of choosing the same path by using some constant K, so we can use this Q-Function like this:

$\Rightarrow \hat{Q}/K$ here we can change the value of $'K'$ in each epoch accordingly.

---

This scribe is based on lecture taught by **Prof. Aritra Hazra** on 31-March-2021 in Machine Learning(CS60050) course.
All the figures in this document are taken from Handout-14a