Introduction to Reinforcement Learning

Aditya Das Analytics Club, IIT-M
Nihal John George Computer Vision Club, IIT-M

Logistics

- All queries in Zoom public chat, one of us will respond in due time. Requesting all participants to **mute their mics and switch off video**.

- Short doubt clearing sessions after each topic if needed, feel free to unmute and ask that time.

- For any software install issues, please consult our document (available on Shaastra.org -> Workshops -> Reinforcement Learning).

- We will walk through all code and gameplay on the presentation

- Please tell on chat if presentation is not visible/audible.

- Timings are 9-12, 1-4

Content Outline

Forenoon Session

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

Content Outline

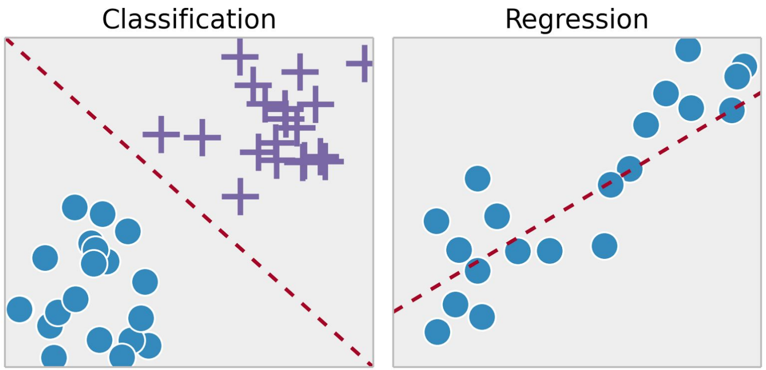
Forenoon Session

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

What are the different ways to **learn**?

Supervised Learning - Given input data and target value/class, have to predict the target for unseen input

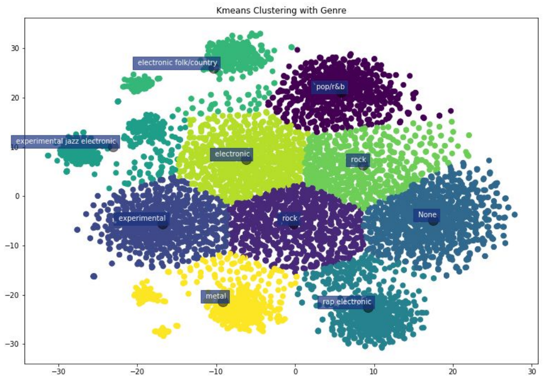
eg. Tumour classification given many scans with labelled tumours, House Price prediction given previous year trends)

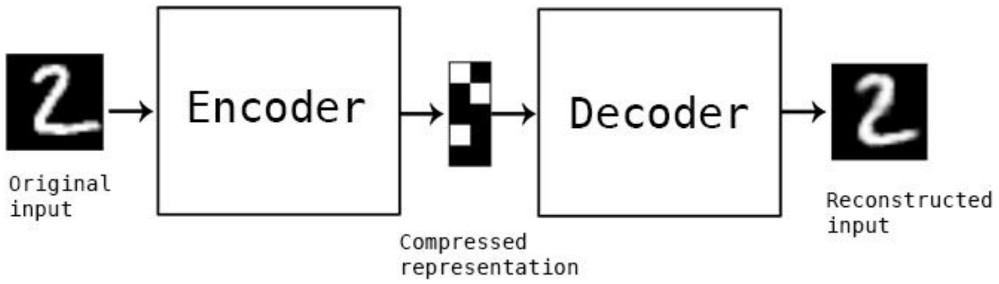


What are the different ways to **learn**? (cont.)

Unsupervised Learning - Given input data but no targets, have to find patterns in the input data

eg. Clustering music into genres, Compressing images using autoencoders



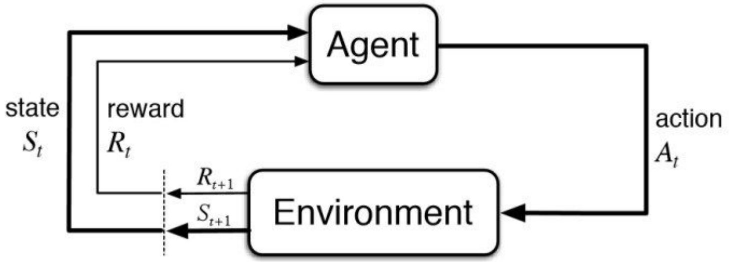


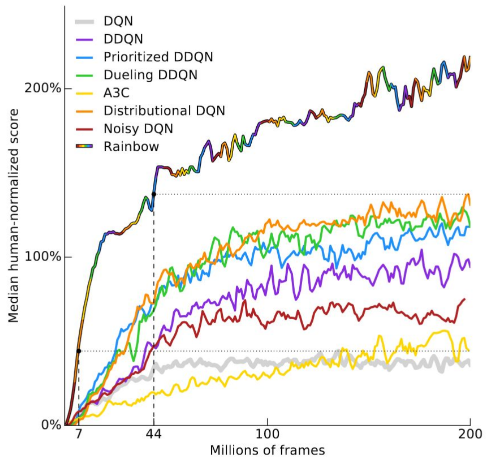
What are the different ways to **learn**? (cont.)

Reinforcement Learning - Model learns to take the correct actions to maximise reward, given the state of the environment.

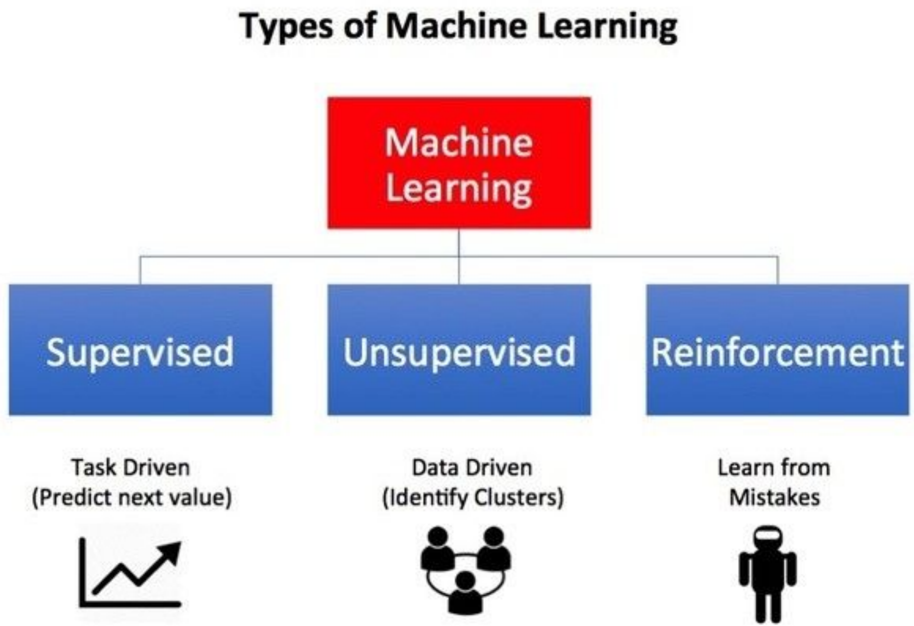
Input and a score is given, model takes action to increase that score. No particular score to achieve, just higher the score, the better.

eg. Optimally playing Poker, Chess, Improving network load balancers





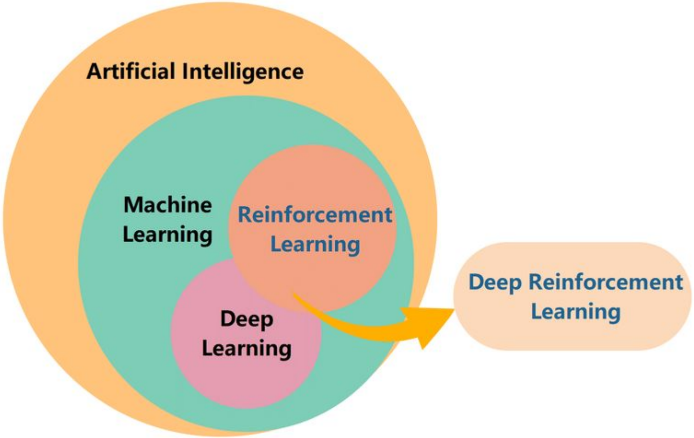
Side by Side Comparison



RL vs DL vs ML vs AI

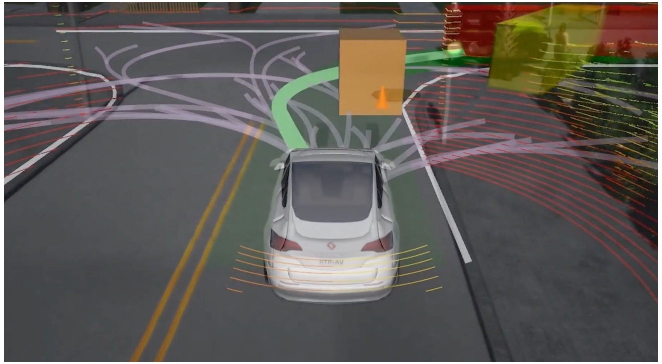
RL is a form of machine learning, which in turn is a subset of AI. This stems from the layman description of ML - a machine which improves performance on some task over time.

Deep learning involves the use of neural networks, which can be useful for RL in approximating value functions. This is called **Deep RL**

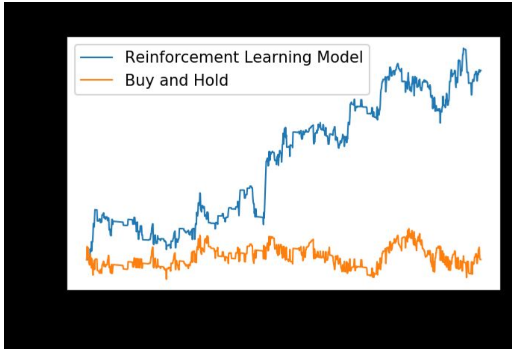


Real World Uses of RL (Other than fancy games)

Robotics and Self Driving Cars - Control, trajectory planning



Quantitative Trading - Decide portfolio, buy and sell actions, maximise earnings



Recommender Systems and Advertising - Using activity history, recommend options to user, maximise engagement

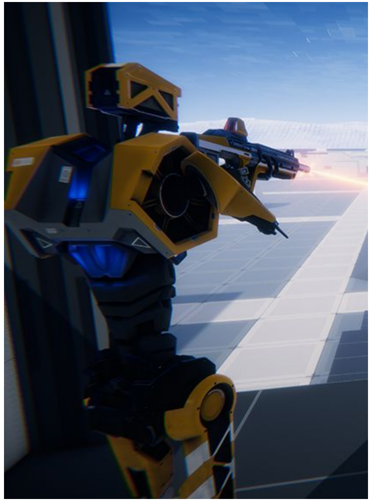
Content Outline

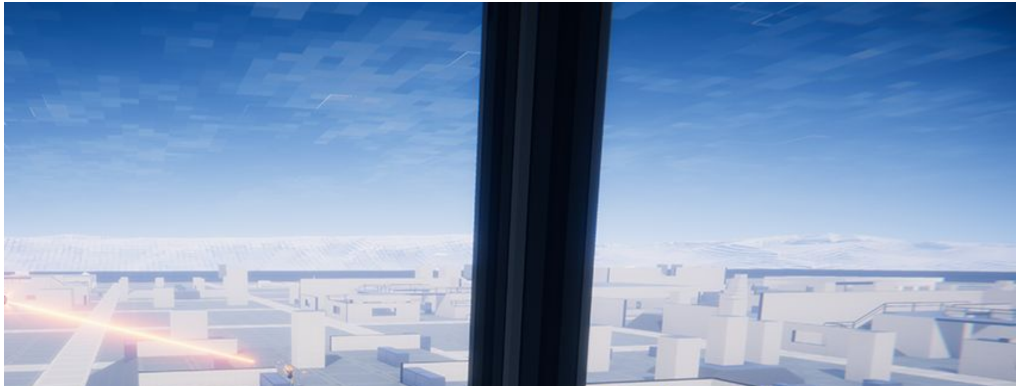
Forenoon Session

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

An Example Situation

- Consider a shooting game (like Counter-Strike, Call Of Duty)
- We can break down the game into the following parts





Players

- Objective - Maximise score (by shooting all opponents and not dying)
- Take actions to achieve goal
- Are bounded by rules of their environment

Environment

- Consists of the game area and its points system
- Enforce rules on player actions (movement and attack)
- Provides the objective to players
- Records the actions of players at each timestep, computes new positions and reward for each agents.

Terminology

- **Environment** - (*Game arena and points allocation code*) - The world in which the agent operates. It defines what actions an agent can take and how the situation of the agent evolves over time.

- **Agent** - (*Players*) - The entity which interacts with the environment in order to achieve its objective.

- **Action** - (*Shoot/Buy/Hide*) - The methods by which agents interact with the environment.

- **Reward** - (*Points*) Feedback given by the environment to an agent when the agent takes an action.

- **Objective** - (*Win the game*) - Goal pursued by all agents => Maximise reward.

Devising a Strategy - What are States?

- Some things you'll keep in mind
  - Where am I currently located in the battlefield?
  - Where is everyone else (teammates and opponents) on the battlefield?
  - How much ammunition (bullets/grenades etc.) do I have left?
  - What is my health condition?
  - *For now, let's settle with this information*

- **Observation** - Information of the environment given to the agent.

- **State** - Encompasses the current situation of the agent.

- For certain RL problems, state is completely defined by the observations (for eg, Tic Tac Toe, Chess).

- For others, state may not be completely defined by the observations (eg. Poker - can't observe other players cards). We will not delve into the formalities of these problems.

- Taking an action causes the agent to transition between states (sometimes back to the same state).

Content Outline

<u>Forenoon Session</u>

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

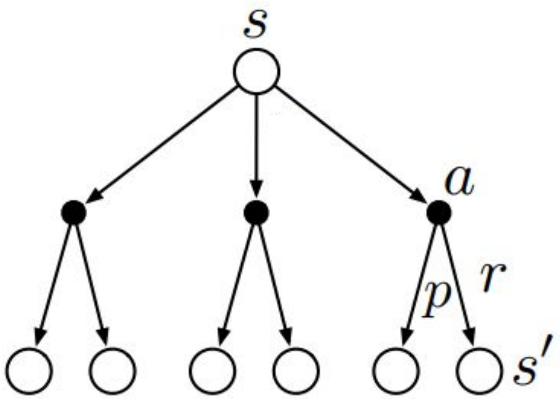
Markov Decision Process (MDP)

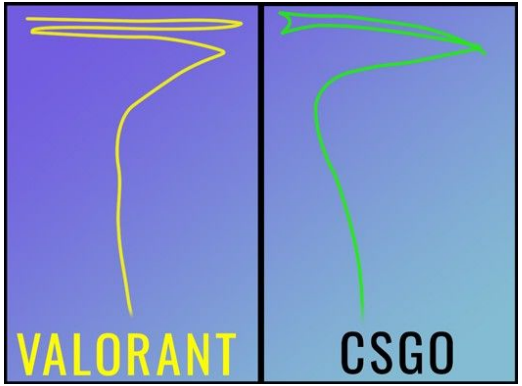
- Mathematical construct for defining RL problems.

- Given an environment and an agent,
  - Environment is in some current state *s*
  - Agent receives state information and decides to take an action *a*
  - Action causes environment to transition to state *s'*
  - Reward *r* is given to agent during this transition

- But the transition may be random!
  - May end up at $s' = s_2$ instead of $s' = s_1$
  - Different probabilities of transitioning to different states
  - Rewards also may be random for the same state transition

- In the shooter game, the gun has some recoil (upward tilt due to bullet momentum) which can be modelled using probability distributions. So an opponent may or may not be hit even for the same action.





MDPs (cont.)

- This process of transitioning is called Markov Decision Process (MDP)

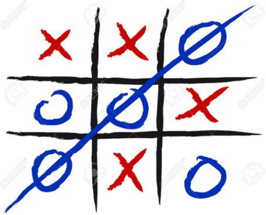
- Probability function *P(s', r | s, a)* read as "Probability of getting reward *r* and going to state *s'* given the current state *s* and action *a* taken in this state".

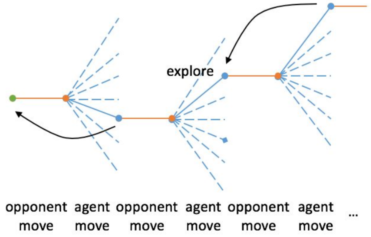
- **Markov Property** - The dynamics of the environment depend only on *P(s', r | s, a)* - no dependence on states before *s*

- In other words, the state *s* encompasses all information about the present and past situation of the environment, after undergoing the actions taken by the agent

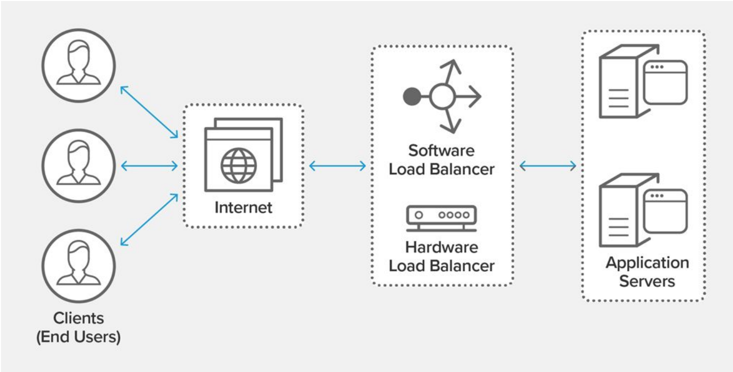
Examples

Tic-Tac-Toe: Define the state as which cells are occupied by which sign. Actions are drawing an X for the human, the environment responds randomly by putting an O somewhere. No reward is given during the game, except +1 when the human wins, or -1 if the human loses, or 0 if it's a draw.





Load balancing: Given a queue of requests from users and a number of servers (may not be identical), assign requests to servers. States are the occupancy of each server along with the current queue length. Rewards are -1 for each timestep, -100 for failed response, +10 for successful response



Try Making Your Own MDP!

Using the rules of the MDP, try to define your own real-life MDP and put it on the Zoom chat

For reference, the rules are -

1. Define the environment, states, actions and rewards
2. Explain why the Markov property holds for your example

We will stick to finite MDPs (finite number of states and actions). In case of continuous variables like position, we “discretize” to to minimum quantity to make it finite.

Content Outline

<u>Forenoon Session</u>

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

Policy

Strategy of the agent is called policy

Formally, the policy is defined as a mapping of states to actions. It is represented with $\pi(s) : S \rightarrow A$

The reinforcement learning problem is to achieve the optimal policy for a given environment

Finding this policy may be difficult to do directly

Eg. Tic Tac Toe’s optimal policy is already found - using exhaustive search of all possible games

How do we quantify the ‘goodness’ of a state or an action?

Expected Rewards

Define G_t as the total reward obtained for all time instants t+1, t+2, ... , T

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

T is considered as the final timestep

One example: points you get at each time step of a shooting game

But for unending tasks, T tends to infinity, and G_t may also blow up

So we use a different formulation, known as the **discounted** return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Where $0 \leq \gamma \leq 1$ is known as the **discount rate**

Quick Task

Suppose I give you an unfair coin P(heads) = 0.4, P(tails) = 0.6

Suppose to toss the coin you have to pay 0.5$. Also assume that you will toss till you run out of money. If I give you $1 for each time you get heads, what is the expected amount of money you get?

Here your choice of actions are - toss or do nothing. I enforced a policy which always chooses to toss till the wallet is empty.

Let us find expected R_t
$E[R_t]$= sum(profit*Probability of getting that profit) = 0.5*0.4 + (-0.5)*0.6 = -0.1 for all t due to independence of tosses

Now G_t = sum(R_i) for all $i \geq t$
= -0.1 -0.1 -0.1 …. = $-\infty$
But if the wallet is limited, you eventually run out of money
So G_t = wallet money at time t

Discounting

Notice that G_t has now become a weighted sum of rewards. If $\gamma < 1$, rewards closer in the future receive higher weight than the ones farther away.

In a way, we are forcing the agent to consider short term gains more than long term ones. This is helpful in very long MDPs where discovering the values of states and actions is tedious, and a high gamma may not lead our model to the correct policy in reasonable time

If $\gamma = 0$, then the agent only considers the next reward.

If $\gamma = 1$, then all the rewards are given equal weight, so the agent has become far-sighted.

This γ is a hyperparameter (set by us according to our liking), and can be tuned for different behaviours

Value Functions - How Good Are States and Actions?

The expected return $E[G_t]$ of a state (or) state-action pair. Higher expected reward means higher value, and so we should choose an action that is most valuable

Value function - The value of a state under a policy *π* is denoted as $V_\pi(s) : S \rightarrow R$

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

Action Value function - The value of a state-action pair under a policy *π* is denoted as $Q_\pi(s,a) : S \times A \rightarrow R$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

The Gridworld

For illustrating the process of determining value functions, we are going to use an environment called Gridworld

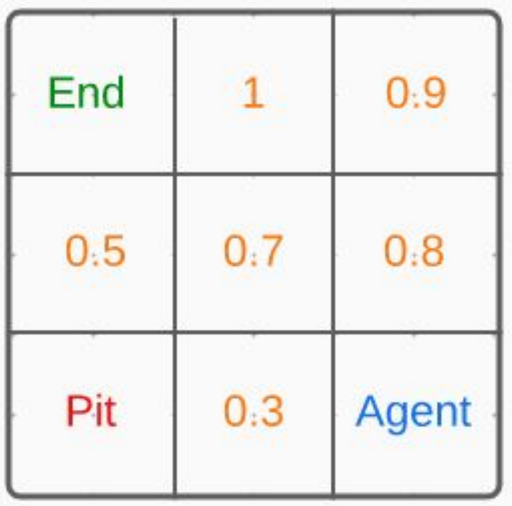
In this environment, the states are the cells of this table. The actions are moving up, down, left or right.

The starting state is Agent. The two terminal states are End (Winning) and Pit (Losing).

The reward for falling into the pit is -5, for reaching the end is +10 and for every other step is -1

Now how do we decide a policy to move?

Value Assignment



In the beginning, let's assign values to the states according to our understanding

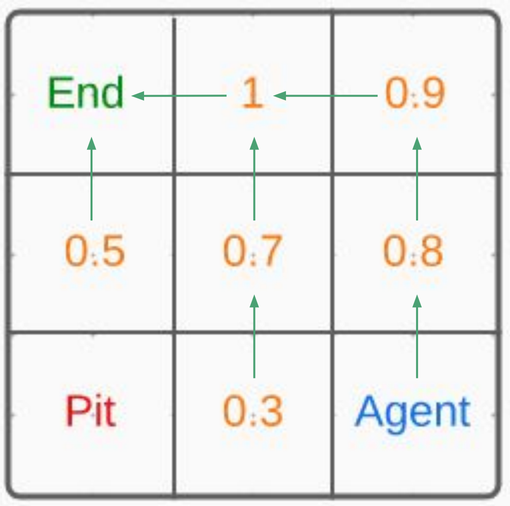
Given these values, how do we devise a strategy for moves?

Notice that you would always want to maximise your return.

Since value V(s) denotes the expected return after going to state s, we can choose the direction of the most valuable neighbour.

Now let's place the arrows indicating the strategy

Policy



Here is the policy. Notice that we have always chosen the best neighbour or the End state for each cell

Mathematically, given all the values, we can construct the policy $\pi(a|s)$ as

$$\pi(a|s) = 1 \text{ if } a = \text{argmax}_{(a)} V(s') \quad \text{[i.e, the best action]}$$
$$= 0 \text{ otherwise}$$

So we set a probability of 1 to choose the best action, so we see only one outgoing arrow from each non-terminal state

Determining the Value Function

Now we have a way of making the policy once we have a value function

But our previous value function was estimated by us.

How do we get the agent to find the true value function, in terms of expected return G_t?

For this, we use the **Bellman Equation.** It is derived by noticing a recursive expansion. Suppose you do an action in state s that leads to state s',

$V(s) = E[G_t|S_t=s] = E[R_{t+1} + \gamma G_{t+1}| S_t=s]$

($G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

$G_t = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \ldots)$

$V(s) = E[R_{t+1}| S_t=s] + \gamma E[G_{t+1}| S_t=s]$

$V(s) = E_a[R(s,a)] + \gamma V(s')$ **[The Bellman Equation for V]**

We Are Close to The Solution! But ....

Solving a system of |S| equations can run into tens of thousands of equations for complex environments

Equation solving takes time of the order $|S|^3$, which can take days or months to solve

We need a way of at least getting close to the optimal value functions with relatively few iterations

For this, we explore a method **Q-Learning**

Content Outline

Forenoon Session

1. Introduction, Clearing the Buzzwords
2. Sequential Decision Making - How to choose the best action?, Terminology
3. Markov Decision Process (MDP)
4. Value Functions, Bellman Equation
5. Q-Learning + Demo

Q Learning

For Q learning, we use the Q function

Recall that Q(s,a) gives the value of taking action *a* at state *s*

This is simpler to use than the V function, since computing the policy is rather simple --- For a particular state, check Q of all possible actions, and choose maximum

Previously, we had to compute the policy using V. So we need to know which action takes you to which state. For Gridworld, it is simple --- UP takes you to the upper neighbour etc.

But for complex environments, the same action can take us to different states.

So we choose not to bother with V, instead Q will directly tell us what to do at a state s.

Q-Learning

The Bellman Equation for Q has a similar form as V

$$\mathbf{Q(s,a) = E_a[R(s,a)] + \gamma \max_{a'} Q(s',a')}$$

However, we want to iteratively improve our Q(s,a) estimate --- our estimate should become closer to satisfying the Bellman Equation --- our error wrt Bellman RHS should reduce.

We will use a sampling version of the Bellman equation to avoid the expectation function.
The expectations above will be computed using many samples, which will eventually converge to the mean.

$$\mathbf{Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')}$$ **[Sampling version of Bellman]**

Temporal Difference (TD) error

Now when we are trying to learn from the game, like most ML algorithms, we need a cost function or an error that we need to minimize. In Q learning this is the TD error.

Suppose you perform an action a in a state s and reach state s' with a reward r. From the previous equation, you compute Q_calc = r + $\max_{a'}(s',a')$. This is the target value. According to what you have learnt so far, you know the predicted value as Q(s,a).

The difference between these two is called the TD error, which we are trying to minimize

TDerror = **Q_calc - Q(s,a)**

Exploration vs. Exploitation

When we are training our agent, we would want it to explore all possible states and figure out which are the best, and with enough exploration it can build a very accurate q table

So what is a good exploration strategy? Should we always just choose random actions? The problem with that is you have a very low chance of ever moving forward in the game. We do of course want to move forward in the game, but we want to explore along the way

This method is called **epsilon-greedy** strategy. We fix a base probability to explore which is called epsilon. Suppose epsilon is 0.2, so whenever we have to choose an action, there is a 20% chance of choosing a random action (**exploration**) and an 80% chance of choosing the best action (**exploitation**) as dictated by the qtable.

Q-Learning Algorithm

1. Start the game with a Q-table (table with S rows and A columns) filled with 0s
2. Repeat the following till TD < threshold or for a fixed number of episodes:

    a. With probability ϵ (epsilon), sample a random action, with 1-ϵ, choose the best action using Q(s,a)

    b. Send action to env, get reward R(s,a) and new state s'

    c. $Q_{target}(s,a) \leftarrow R(s,a) + \gamma \max_{a'} Q(s',a')$ [Target according to Bellman Sample]

    d. $TD \leftarrow Q_{target}(s,a) - Q(s,a)$ [TD between Bellman and current Q]

    e. $Q(s,a) \leftarrow Q(s,a) + \alpha * TD$ [Update current Q by reducing the error]

Time for a Demo!

Clone/Download Zip of the Github Repo

Link: https://github.com/AdityaDas-IITM/Shaastra-2021

We will be doing a ‘Code-Along’, which means I will share my screen showing myself typing the code and explaining the lines, and the participants can type the code on their computer

Google Colab will be used since platform issues are minimal

Visit https://colab.research.google.com

Problems with Q Learning

The major limitation of Q-learning is the memory requirement.

We have simplified the state space of the snake game such that there will never be more that 256 states and the q table will never be bigger than 256x4.

But what if that was not possible? Imagine we are playing a shooting game where your state is the screen you can see. It's not possible to make an easy state representation in this case and so you will have millions of states

Can you make such a big Q-Table and expect your laptop to survive?

So what do we do now?

You will find out in the next session

That marks the end of the forenoon session, and the basics of Reinforcement Learning!

The afternoon session deals with Deep RL, where we use neural networks to solve the RL problem

Do provide feedback on the session through Whatsapp/Zoom Private Chat