

Part I- Reinforcement Learning and Deep Q Learning

I discussed and went through the Deep Q Learning Algorithm for the Cartpole game and provided the mathematical intuition and part of the codes during the lecture. See the figure below.



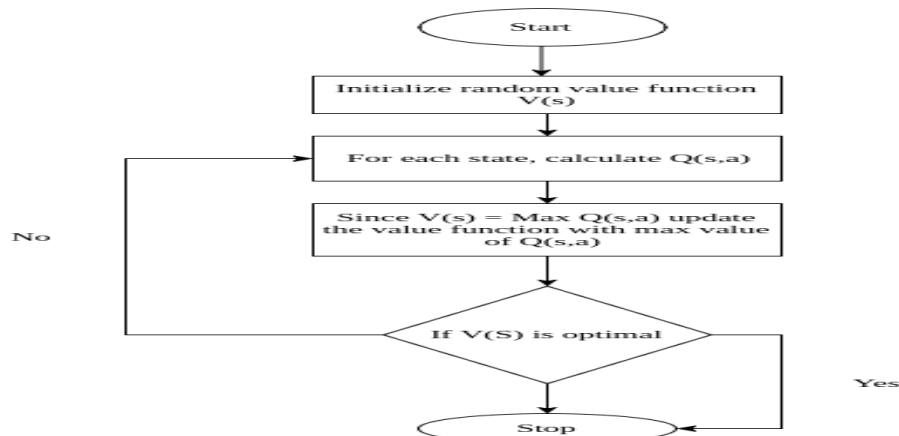
“Markov Decision Process”

- S : all possible states
- A : all possible actions
- R : reward distribution given (s, a)
- P : transition probability to s_{t+1} given (s, a)
- γ : discount factor

Furthermore, I explained the Value Iteration for solving the Bellman equation with an example.

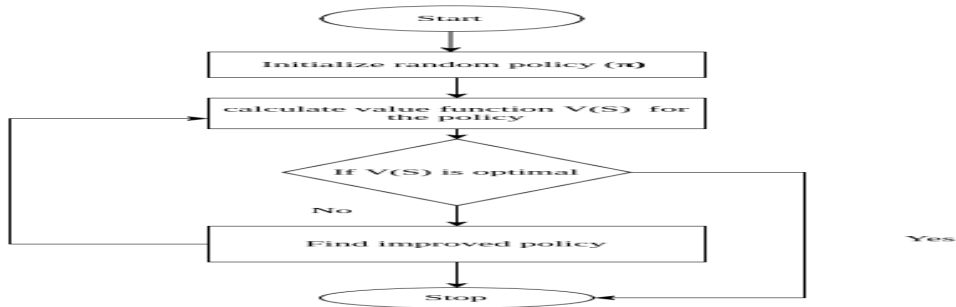
$$Q^\pi(s, a) = \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \sum_{a'} Q^\pi(s', a') \right]$$

As I stated during the lecture, we start with a random value function in Value Iteration. Obviously, the random value function might not be optimal, so we look for a new, improved value function in an iterative fashion until we find the optimal value function. Once we find the optimal value function, we can easily derive an optimal policy from it. See the figure below.



[A]. In policy Iteration, we start with the random policy and find the value function of that policy; if the value function is not optimal, we find the new, improved policy. See the figure below. You need to explain Policy Iteration with a clear example, as I did for the Value Iteration. You must

elucidate all the steps in Policy Iteration with an example. [Length of your recording ≤ 10 minutes]. You should provide your slides.



[B]. You must explain the following codes line by line and use hyperparameter optimization to improve the entire notebook I deposited in our AI Google Drive for Deep Q Learning. [Length of your recording ≤ 15 minutes]. You should provide your notebook. Use Google Colab.

```

def train(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        # Training logic would go here

for e in range(n_episodes):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    done = False
    time = 0

    while not done:
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else -10
        next_state = np.reshape(next_state, [1, state_size])
        agent.remember(state, action, reward, next_state, done)
        state = next_state

    if done:
        print("episode: {}/ {}, score: {}, e: {:.2}"
              .format(e, n_episodes-1, time, agent.epsilon))

    time += 1
    env.render()

    if len(agent.memory) > batch_size:
        agent.train(batch_size)

    if e % 50 == 0:
        agent.save(output_dir + "weights_"
                  + '{:04d}'.format(e) + ".hdf5")

```

[C]. FrozenLake game. We have studied the Cartpole game. The idea behind the FrozenLake game is quite simple. It consists of 4 x 4 grid blocks, where each block can have one of the following four states:

S: Starting point/Safe state

F: Frozen surface/Safe state

H: Hole/Unsafe state

G: Goal/Safe or Terminal state

















In each of the 16 cells, you can use one of the four actions, namely up/down/left/right, to move to a neighboring state. The game's goal is to start from state S and end at state G. In this part, you should elucidate the FrozenLake problem statement, write the codes, and show how we can use a neural network-based Q-learning system to learn a safe path from state S to state G. **[Length of your recording \leq 15 minutes]. You should provide your notebook. Use Google Colab. See the figure below.**

GYM (Open AI)



Q-Learning: Frozen Lake

Reinforcement Learning

 S	 F	 F	 F
 F	 H	 F	 H
 F	 F	 F	 H
 H	 F	 F	 G

S = Start ; F = Frozen ; H = Hole; G = Goal

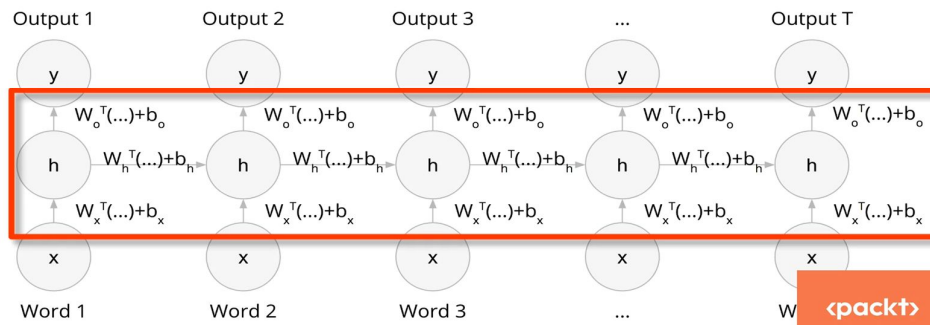
Rules: You have to start from point 'S' and reach point 'G-Goal'. You can only walk upon the 'F-frozen' pieces. If you step on the 'H-hole' then you are doomed! You lose.

Part II- Recurrent Neural Network (RNN) and Long short-term memory (LSTM)

During the lecture, I elucidated the mathematical intuition behind the Recurrent Neural Network (RNN).

Why is it “recurrent”?

Note: activation functions omitted for simplicity



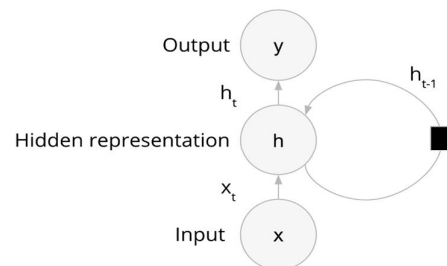
Why is it “recurrent”?

- Every step is the same!

Recall the most famous recurrence:

$$Fibonacci : x_n = x_{n-1} + x_{n-2}$$

$$h_t = \sigma(W_h^T h_{t-1} + b_h + W_x^T x_t + b_x)$$



[A]. You must provide the complete mathematical intuition behind Long short-term memory (LSTM) by an example. LSTM has several gates: forget, input, and output. All these gates must be explained. The reason that we need to improve RNN should be elucidated. Furthermore, you should show all the steps for backpropagation in LSTM using the example you employed in the first part of this question. **[Length of your recording ≤ 10 minutes]**

[B]. I have provided the following notebook[RNN-GRU-LSTM] in our AI course Google Drive

https://drive.google.com/file/d/1QO_7PQonRDRbbIHZcbd-XhLQwNCQXpPm/view?usp=sharing

In this section, you must explain all the codes in the notebook line by line in Part I. Then, you should improve the model's R^2 score (preds, y_{test}) in Part I. You should also explain all the codes line by line in Part II. Furthermore, you should improve the [Train Score 14.93 RMSE] and [Test Score: 46.19 RMSE] of the model in Part II. **[Length of your recording \leq 15 minutes]. You should provide your notebook. Use Google Colab.**

Part III- Search Algorithms

Informed search, also known as heuristic search, is an algorithm that uses both breadth-first search and depth-first search approaches combined with some intelligence. The search is guided by heuristics, given some predefined knowledge of the problem at hand. We can employ several informed search algorithms, depending on the nature of the problem, including Greedy Search (also known as Best-first Search). The most popular and useful informed search algorithm is A*. A* search algorithm usually improves performance by estimating heuristics to minimize the cost of the next node visited. Total cost is calculated with two metrics: the total distance from the start node to the current node and the estimated cost of moving to a specific node by using a heuristic. When we are attempting to minimize cost, a lower value indicates a better-performing solution (Figure below).

$$f(n) = g(n) + h(n)$$

$g(n)$: cost of the path from the start node to node n

$h(n)$: the cost from the heuristic function for node n

$f(n)$: the cost of the path from the start node to node n plus the cost from the heuristic function for node n

[A]. Provide two manual examples [sequence of tree processing using A* search] and write a python script for each example. **You should provide your notebook. Use Google Colab. [Length of your recording \leq 7 minutes].**

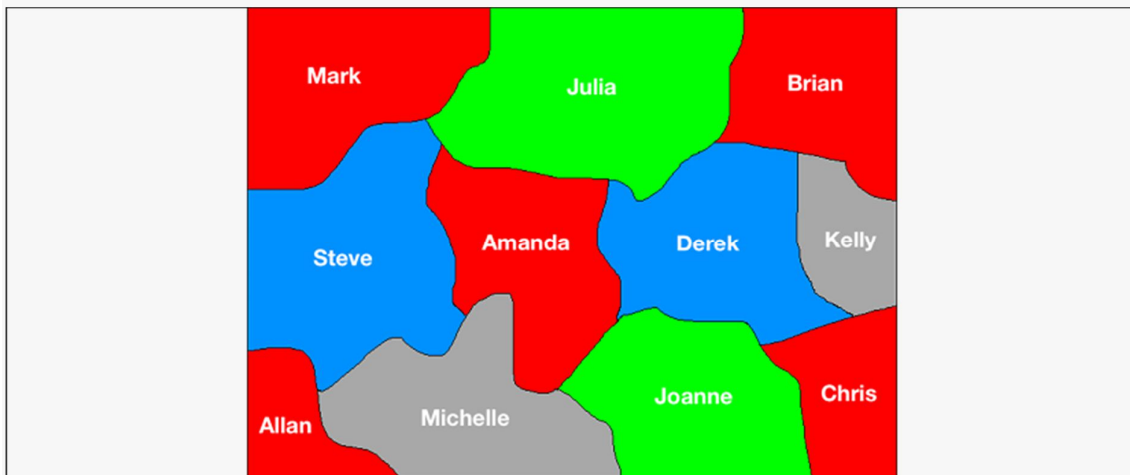
[B]. Min-max search aims to build a tree of possible outcomes based on moves that each player could make and favor paths that are advantageous to the agent while avoiding paths that are favorable to the opponent. To do so, this type of search simulates possible moves and scores the state based on a heuristic after making the respective move. Min-max search attempts to discover as many states in the future as possible; but due to memory and computation limitations, discovering the entire game tree may not be realistic, so it searches to a specified depth. Min-max search simulates the turns taken by each player, so the depth specified is directly linked to the number of turns between both players. A depth of 4, for example, means that each player has had 2 turns. Player A makes a move, player B makes a move, player A makes another move, and Player B makes another move. Provide two manual examples [sequence of tree processing using A* search] and write a python script for each. Provide two manual examples [The sequence of tree processing using min-max search] and write a python script for each. **You should provide your notebook. Use Google Colab. [Length of your recording \leq 7 minutes].**

[C]. Depth First Search (DFS), Breadth First Search (BFS), and Uniform Cost Search (UCS) are search techniques that are commonly used on graphs to get to the solution. These are examples of uninformed searches. They do not use any prior information or rules to eliminate some paths. They check all the plausible paths and pick the optimal one. Provide one example for Depth First Search (DFS), Breadth First Search (BFS), and Uniform Cost Search (UCS) and write a python script of each example. [Length of your recording ≤ 8 minutes]. You should provide your notebook. Use Google Colab.

[D]. Consider the following figure. We have a few regions in the figure that are labeled with names. The goal is to color with four colors so that no adjacent regions have the same color. This is a simple constraint satisfaction problem. Write a Python script and solve this problem and explain your approach. [Length of your recording ≤ 8 minutes]. You should provide your notebook. Use Google Colab.

Hint: You can use the following to start your notebook

```
[1] from simpleai.search import CspProblem, backtrack
[2] def constraint_func(names, values):
    return values[0] != values[1]
# Define the function that imposes the constraint
# that neighbors should be different
```



Important Note

1. For this project, you should record a video. You should show your face. Your voice must be clear. It is your responsibility to make sure your video is working. You should put your video and notebook in Google Drive. Your folder should have your [student ID number, first name, and last name]. Only one video should be submitted for all parts of this project.
2. Your notebook should be well structured, and before each code block, you should explain the codes of the next block. You should follow the format of the notebooks I presented in the lectures.
3. You should make your Google link open access and submit your video to the auxiliary TA of our course, Mr Behzad Mohasel Afshari, via e-mail on June 10, 2024, before 4 PM. His e-mail for this course is ai.2024.cs@gmail.com.
4. You should use Google Colab. Insert all the slides into your Google Colab environment, then record all parts of this project.
5. You should submit the following three items
 - A. Your Google Colab notebook.
 - B. Your slides
 - C. Your video file. The extension of your video file should be .mp4. Other formats will not be accepted.
6. Your video file and notebook should be saved as [student ID number, first name, and last name]. Your total video file should not be more than 95 minutes. Your recording must be in English.
7. You should ensure that your codes and mathematical algorithms are flawless.

If further elucidation is warranted, please don't hesitate to contact me.