The folder consists of 5 files.

**Files**

------

1**) *expected\_reward\_average\_case.py***: This file is responsible for **estimation of the state distribution ratios**. This file is written in **accordance to** the **code given in the paper** namely "**Breaking the curse of horizon**"(*average\_case*). **To execute this file**, one must **pass** the **number of instances for which our simulator will generate the data** denoted by **T**. The **number of states** present in the state space of our MDP denoted by **nS**. The **number of actions** should **essentially be** so please keep the **nA = 2(always)**. The **start state needs to be passed** which is **denoted by state**. The user **must pass** the **replacement cost denoted by rep\_cost** which is a **value** in **between 0 and 1**. The **learning rate** is **denoted by l\_rate.** Here the **testing was done on 4 states** so, **behaviour policy is defined only for 4 states**. If a **different number of states** is to be tested, **fix the behaviour policy** appropriately. Each row is the behaviour policy sum as 1. behaviour\_policy takes state value first followed by the action.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

***(Example:***

***if at 2nd state you try to get the probability of 1st action then use behaviour\_policy[1][0])***

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*This module will itself call the "simulate\_episode" class present in the "transition\_simulator.py" module which responsible to return the data and the target\_policy.* Variable **data** is a **dictionary**. This is a **dictionary consisting of the simulated data for 'T' instances**. The **keys** are.

**I) state:** store a list of the states visited by the simulator at each of the T instances.

**II) action:** stores the actions take by the agent in the state 's' present in the corresponding list under key 'state' stored in the form of a list.

**III) next\_state:** The next state our agent travelled to after taking an action 'a' in state 's' in the corresponding list with keys 'action' and 'state' respectively stored in the form ofa list.

**IV) reward:** The reward obtained at each step stored in the form of a list.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

After getting these data, **we get the stationary state distribution ratios by calling the weight\_parameterization module**.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

After we have the state distribution, simulated data and other parameters as mentioned above, we can call the estimate\_reward module to find the final estimated reward.

**===========================end of first file=====================**

***2) transition\_simulator.py :*** The **MDP** that **we are generating** is for **Machine Replacement problem**. So the t**ransition simulator** just t**akes the number of states(nS), number of actions(nA), behaviour\_policy, number of instances(T) and replacement cost(rep\_cost) as input.** **It performs** the **simulation** and **returns the state, action, next\_state and reward value in the form of a dictionary** named as **data**. Along **with data** it also **passes the target\_policy**. One does **not need** to create the **Probability transition function** **or** the **reward function** which are **automatically generated** as long as **number of states and rep\_cost is known**. In this file, **we call** a **module named Machine\_Replacement\_create\_env** which is **responsible** for **creating the model**(**transition probability matrix and reward matrix**)

------------------------------------------------------------------------------------------------------------------------

***(example: P=obj.gen\_probability() and R=obj.gen\_reward() where P and R are probability and Reward matrix respectively and obj is the instance of the Machine\_replacement class).***

------------------------------------------------------------------------------------------------------------------------=========================end of file desciption===============================

***3) Machine\_replacement\_env\_create.py:*** This module **creates our MDP**. **One just needs to pass** the **number of states** and the **replacement cost denoted by(rep\_cost),** and the **functions gen\_probability()** and **gen\_reward()** will **give us the Probability** and **Reward matrix.**

**-----------------------------------------------------------------------------------------------------------------------**

**(example:**

**nS = 4;**

**nA=2;**

**rep\_cost=0.7**

**obj = Machine\_Replacement(rep\_cost,nS,nA);**

**print("Probability matrix");**

**print(obj.gen\_probability());**

**print("Reward Matrix");**

**print(obj.gen\_reward()); )**

**-----------------------------------------------------------------------------------------------------------------------**

=======================end of file description================================

***4) weights\_parameterization.py*** : In this file, we **create** a **neural network model** using **pytorch**. This **neural network** will be **accepting** the **input and output dimenision.** The **input dimension** is essentially **equal** to the **number of states (input = nS)**. **Output dimension** is essentially **1(output = 1)**. The **network** **contains a function** named **forward().** This **function** **takes** a **state 's' as input**. **Converts it into one-hot vector.**

**------------------------------------------------------------------------------------------------------------------------**

**one\_hot\_vector = zeros(nS);**

**one\_hot\_vector[s] = 1;**

------------------------------------------------------------------------------------------------------------------------

**and then passes it to our network.**

**-----------------------------------------------------------------------------------------------------------------------**

**(example:**

**obj = weights(ns,1)**

**state =[0,1,2,3]**

**for i in state:**

**obj.forward(i);**

**)**

=============================end of file description==========================

***5) estimate\_reward.py:*** In this file, **we find the estimated reward**. This **module** is **performing the reward calculation** **as per the paper mentioned above.** The **returned output** is the **expected reward** by **using the formula SUM(state\_distribution(state\_s) \* beta(s,a)\* reward)/SUM(state\_distribution(state\_s)\*beta(s,a)).**

-----------------------------------------------------------------------------------------------------------------------

**(example:**

**obj = est\_reward(state\_distribution,target\_policy,behaviour\_policy,data,nS,nA);**

**print(“reward is:”,obj.find\_reward());**

**)**

========================end of file desciption================================