# Reinforcement Learning Report 1

Rugved Hattekar 50320920

1st March 2020

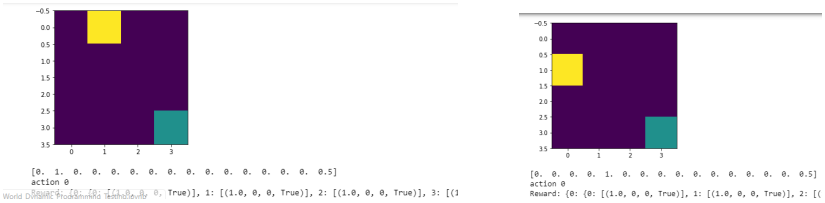
## 1 Introduction

The aim of the project is to build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. Reinforcement Learning is an area of Machine Learning which focuses on creating agents capable of taking actions in an environment in a way that maximizes rewards over time. In this assignment, the agent will learn an optimal policy through "Dynamic Programming" which will allow it to take actions to reach a goal. The environment and agent will be built to be compatible with OpenAI Gym environments. OpenAI provides simple physics or gaming simulations that allow you easily test out your RL skills. The objective of the reinforcement learning is to learn to act in a way that will maximize its expected long-term rewards. The algorithm used by the agent to determine its actions is called its policy. Hence we are here to design a policy which would help the agent to navigate to the goal location in shortest path with maximum reward.

## 2 Stochastic vs Deterministic Environment

Stochastic environment simply implies that there is a certain probability p that agent will not follow the action provided by the policy and will perform some other defined random action. If that happens than the agent will take different route than expected and try to take action based on the new action. It is may possible that robot may not end up at the exact goal location it is supposed to during the process but it may end up in better exploratory environment.
A stochastic environment, there is always some level of randomness. Any games that involve dice are good examples - you can never be certain that a specific number will be rolled.
Whereas for Deterministic environment, Given the current state of the environment and an action of an AI agent, the AI can know with certainty the next state of the environment.

f(state i, action i) = state f



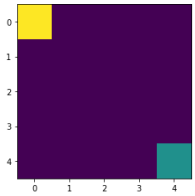
(a) Stochasticity (b) Expected

Figure 1: How Stochasticity Makes a difference

So the resulting state does not change given the same state and action at all times. This is applicable for both single and multi-agent environments (with only one, or more than one AIs interacting with the environment)
In this project the, The there is a choice to train the agent in deterministic as well stochastic environment. The Stochsticity is introduced at the action function in the code. There is a 5% probability that the agent will pursue some other action than defined policy action in that case it may land up in some other state.

# 3 Environment

The environment we provide is a basic deterministic and Stochastic 4 4 grid-world environment (the initial state for a 4 4 grid-world is shown in Figure) where the agent (shown as the yellow square) has to reach the goal(shown as the green square) in the least amount of time steps possible. The environment's state space will be described as an 4 * 4 matrix with real values on the interval [0, 1] to designate different features and their positions. The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of -1 for each action and 0 when it reaches terminal state and +6 additional reward if it goes to one of the states and -4 if it goes to the one particular defined space.



Action space : {Down, Up, Left, Right}
State Space: {Total 16 states, in 4x4 Environment}
Rewards Space: {-4, -1, 0, 6}

# 4 Gym Environments

This Project refers to the gym environments. Gym is a toolkit for developing and comparing reinforcement learning algorithms. Gym Environment uses a particular strcuture which has a class describing all the number of states, actions and reward etc. It defines the all the actions in details, all the states and probabilities involved with it.
The Basic Structure of Gym environment and which is used in this course is as follows. It contains four building blocks as follows:

1. Init() function in class:
– It contains the information for number of state spaces, total number of actions and rewards

2. Reset() Function
–It defines the initial and final state of the Robot. The initialization values assigned with the final and terminal states. This function is used to reset the agent after number of time-steps. Every time the reset function is called the agent will comeback to initial position as it was in beginning.

3. Action() Function
–It describes the action which the agent is taking. In this project the actions are only left, right, up and down. The stochasticity is also introduced in the action space. The Stochasticity is introduced here so that for 5% probability the agent will take random action, than expected.

4. Render() Function
–It helps in rendering the environment.

# 5 Dynamic Programming

The term Dynamic Programming refers to the collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov Decision Process. The MDP is based on the Bellman Equation, we use this equation to solve for each state and action.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[ r + \gamma v_\pi(s') \Big], \end{aligned}$$

We can break the above equation in following parts.
The first piece $\pi(a\|s)$, doesn't actually refer to any value, but instead, it refers to a policy. A policy is simply a probability of performing an action (a) when in state (s) and our goal is to tune the policy to maximize the agents reward.
$P(s', r\|s, a)$ refers to the probability of ending up in a particular state-reward pair given the current state and action. For example, there is a variation of grid world called windy grid world where every time you make take an action, there is a 50% probability you will take another random action. A real world application of this is in games where information is incomplete and we don't always end up where we want or expect to.
The last part [r $+0\ \gamma * V\pi(s')$] is the combination of the current reward and an exponentially decayed reward of future states s' for a given policy $\pi$.

Dynamic Programming can be implemented as two methods 1. Policy Iteration and 2. Value Iterations. This report refers to the Value Iteration Method.

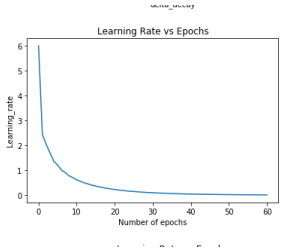
$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

The above equation refers to the value iteration equation used in this report.

While Implementing the value function, I initialize the value iteration policy $\pi$. Computed the values of the value function using the equation given in the above figure. The algorithm implementation is summoned in the figure given below.

Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s, a) \leftarrow E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V(s')$
        $V(s) \leftarrow \max_a Q(s, a)$
Until $V(s)$ converge

Initially the agent starts with all states as zeros, as Random Policy with uniform distribution is used. At the next Iteration values considering rewards are pasted into the matrix and after every iteration the policy values are calculated using the the value iteration formula.




(a) epochs Learning vs Rate


(b) Learning rate vs reward

Figure 2: Plots

```
# obs = env.reset()
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[ 0. -1. -1. -1.]
 [-1. -1.  6. -1.]
 [-1. -1. -1. -1.]
 [-4. -1. -1.  0.]]
[[ 0.    -1.735 -0.265 -1.98 ]
 [-1.735 -0.265  5.02  -0.265]
 [-2.715 -1.98  -0.265 -1.735]
 [-6.45  -2.715 -1.735  0.   ]]
[[ 0.       -1.554925 -0.7452   -2.10005 ]
 [-2.155175 -1.10535   5.7403   -0.7452  ]
 [-4.1556   -2.4602   -1.10535  -1.554925]
 [-8.49085  -4.1556   -2.155175  0.      ]]
```

After Implementation of the algorithm, the convergence is observed as a learning rate. The value iteration policy can be observed after the model is conversed at the end. The converged matrix gives the optimum policy which the agent can follow to reach the goal as early as possible.

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Coverged_matrix [[  0.          -5.78803296  -5.90314123  -7.61363812]
 [ -8.98798014  -7.87544261  -0.74298598  -5.90314123]
 [-15.76426991 -12.57491582  -7.87544261  -5.78803296]
 [-22.97029895 -15.76426991  -8.98798014   0.        ]]
(61,)
```

The Learning rate obtained from the above iterations and policy is given below. The delta(Learning rate factor) is saved to 0.01 in the code for better accuracy and results. As compared to the stochastic learning rate, it does not changes. As even though the state changes but the agent manages to reach the goal.
The Policy evaluation and optimal policy calculations are visible in this calculations done in the image below.

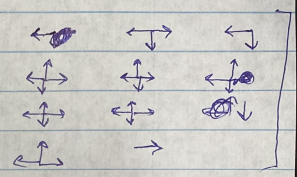
From Computation :

We get the converged Matrix as follows :

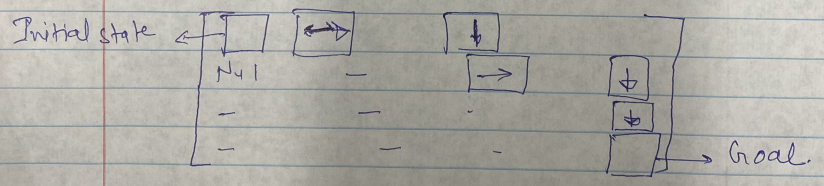
Converged Matrix :

| | | | |
|---|---|---|---|
| 0 | -5.78 | -5.90 | -7.61 |
| -8.98 | -7.87 | -0.74 | -5.90 |
| -15.76 | -12.57 | -7.87 | -5.78 |
| -22.97 | -15.76 | -8.98 | 0 |

Directions.



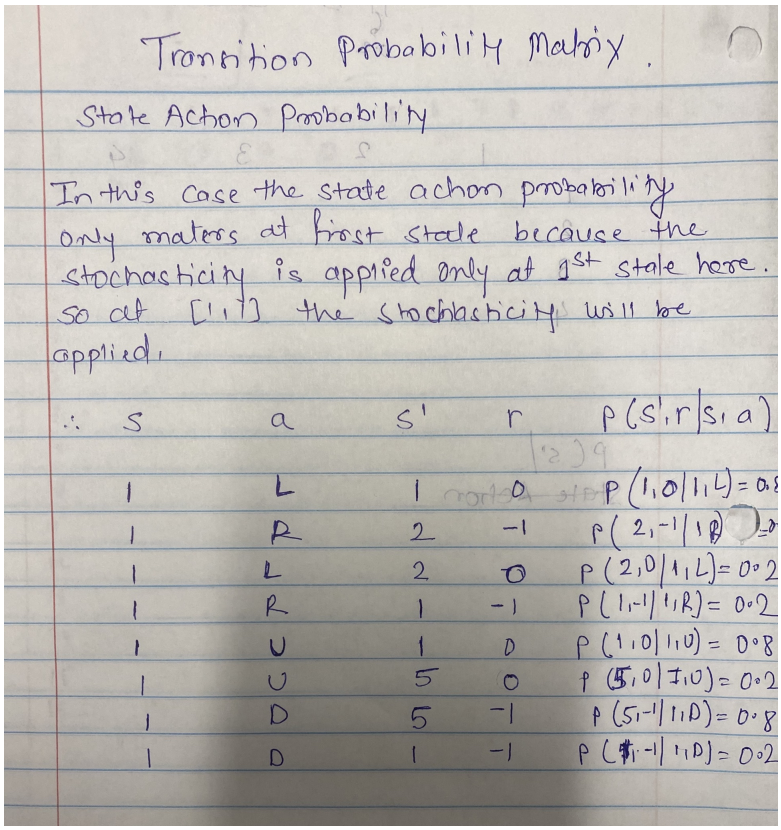
But According to converged values the Optimal Policy of the Value Iteration can be concluded as follows.




This can be conclude after looking at the values of the matrix.

# 6 Transition Probability Matrix

The Action Transition Probabilities are given in the table written below. The stochasticity is given only to one initial state. Hence describedd only for one state.



Transition Probability Matrix.

State Action Probability

In this case the state achon probability only maters at first state because the stochasticity is applied only at 1st state here. so at [1,1] the stochasticity will be applied.

| s | a | s' | r | p(s',r\|s,a) |
|---|---|---|---|---|
| 1 | L | 1 | 0 | p(1,0\|1,L)=0.8 |
| 1 | R | 2 | -1 | p(2,-1\|1,R)=0.2 |
| 1 | L | 2 | 0 | p(2,0\|1,L)=0.2 |
| 1 | R | 1 | -1 | p(1,-1\|1,R)=0.2 |
| 1 | U | 1 | 0 | p(1,0\|1,U)=0.8 |
| 1 | U | 5 | 0 | p(5,0\|1,U)=0.2 |
| 1 | D | 5 | -1 | p(5,-1\|1,D)=0.8 |
| 1 | D | 1 | -1 | p(1,-1\|1,D)=0.2 |

# 7 Observation and Conclusion

In this assignment, The basic setup of Gym Environment and Dynamic Programming has been studied. The results are obtained at different epochs and the best ones are presented here.

One incident to mention here is that, if the learning rate is 0.1 and $\gamma$ is 1. If the rewards are -1 for all states and 0 for terminal state, then the conversion will occur after some iterations. But, if we consider the reward to be 0 for every action and +1 for terminal state, then the model goes in the infinite loop and never converges. For this I figured out that as all states are zero the incrimination in the values does not occur leading it to a non convergent policy.

Hence, Dynamic Programming Algorithm is Studied and Applied.