MAI Project Report

Topic : Reinforcement Learning

A Group Project PBL Report Submitted to

DR. VISHWANATH KARAD

MIT WORLD PEACE UNIVERSITY

Submitted by,

NETRA MADLE (PC7)

SHUBHASMITA ROY (PB11)

VEDASHRI DEBRAY (PA12)

Under the Guidance of

Prof. Rupali Kute, ECE, MITWPU

School of Electronics and Communication Engineering

MIT World Peace University, Pune-411038

(June, 2021)

1. Introduction

With the advancements in Robotics Arm Manipulation, Google Deep Mind beating a professional Alpha Go Player, and recently the OpenAI team beating a professional DOTA player, the field of reinforcement learning has really exploded in recent years. These trending topics in news articles and social media forwards is the reason and inspiration behind choosing the topic - “Reinforcement Learning” for our MAI’s PBL.

Reinforcement learning (RL) 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 is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

We will explore RL through both theoretical as well as practical point of view in this PBL. We have implemented a program to make an AI learn and master a game from a practical point of view.

2. Objective

Objects of PBL project are:

- To study and explore Reinforcement Learning
- To gain insight through the experience of solving an open ended problem in RL
- To implement a program on RL algorithm

3. What is Reinforcement Learning?

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions.
- In Reinforcement Learning, the agent learns automatically using feedback without any labeled data, unlike supervised learning.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

- For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- Since there is no labeled data, the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.
- We will be exploring Q- Learning, an algorithm in RL, in this PBL.

4. RL - future of machine learning?

There are many situations where we just can't label data effectively. In RL, the AI agent has to learn from rewards and since supervised learning relies on large labeled datasets reinforcement learning has a much higher scope of application than any form of supervised learning.

Advantages of Reinforcement Learning are -

1. Reinforcement learning **doesn't require large labeled datasets**. It's a massive advantage because as the amount of data in the world grows it becomes more and more costly to label it for all required applications.
2. It's **Innovative**, unlike reinforcement learning supervised learning is actually imitating whoever provided the data for that algorithm. In supervised learning, the algorithm can learn to do the task as well or better than the teacher but can never learn a completely new approach to solving the problem. On the other hand, reinforcement learning algorithms can come up with entirely new solutions that were never even considered by humans.
3. **Bias Resistance** - if there is bias in the way the data is labeled then a supervised learning algorithm will pick up that bias and learn inherited bias. In this sense, reinforcement learning algorithms are better tools to find solutions free from bias or discrimination.
4. **Online Learning**, reinforcement learning runs in real-time reinforcement learning combines exploration when the machine tests new approaches on the fly to find better

solutions and exploitation when the machine exploits the best solutions which it has found thus far. This means that it can bring results while improving at the same time other algorithms will actually require retraining and redeployment in order to accomplish that reinforcement learning just keeps going.

5. **Goal-oriented**, Reinforcement learning can be used for sequences of actions while supervised learning is mostly used in an input-output manner. Reinforcement learning can be used for tasks with objectives such as robots playing soccer or self-driving cars getting to their destinations or an algorithm maximizing return on investment on ads spend.
6. Reinforcement learning is **Adaptable**, unlike supervised learning algorithms, reinforcement learning doesn't require retraining because it adapts to new environments automatically on the fly.

5. Q Learning

Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.

- The main objective of Q-learning is to learn the policy which can inform the agent what actions should be taken for maximizing the reward under what circumstances.
- The goal of the agent in Q-learning is to maximize the value of Q.
- To perform any action, the agent will get a reward R(s, a), and also he will end up on a certain state, so the Q -value equation will be:

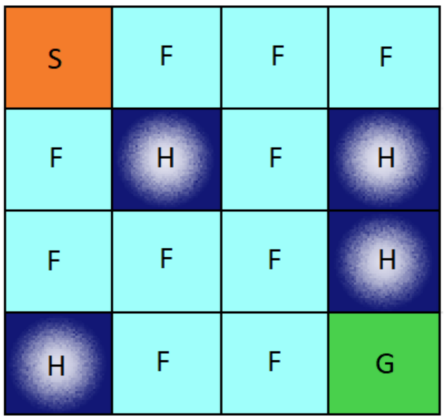
$$Q(S, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s`)$$

Q- represents the quality of the actions at each state.

6. Problem Based Learning

- We have implemented Q- learning using Python. The program is a game named Frozen Lake.
- Explanation:
  - The Frozen Lake environment is a 4×4 grid which contains four possible areas — Safe (S), Frozen (F), Hole (H) and Goal (G).
  - The agent moves around the grid until it reaches the goal or the hole.
  - The agent in the environment has four possible moves — Up, Down, Left and Right.
  - If it falls into the hole, it has to start from the beginning and is rewarded the value 0.
  - The process continues until it learns from every mistake and reaches the goal eventually.

  Here is visual description of the Frozen Lake grid (4×4):



Explanation of Code:

```
## Step -1: Install the dependencies on Google Colab
!pip install numpy
!pip install openai-gym

#import the required libraries.
import numpy as np
import gym
import random

#create the environment usign OpenAI Gym
env = gym.make("FrozenLake-v0")
```

We are using openai-gym library which is a special python library for developing reinforcement learning models.

```
## Step 2: Create the Q-table and initialize it

action_size = env.action_space.n
state_size = env.observation_space.n

print(f"Action Space : {action_size} | State Space: {state_size}")

qtable = np.zeros((state_size, action_size))
print(qtable.shape)
qtable
```

Q table stores the rewards w.r.t. Actions and States.

```
## Step 3: Create Required Hyperparameters

total_episodes = 15000        # Total episodes
learning_rate = 0.8           # Learning rate
max_steps = 99                # Max steps per episode
gamma = 0.95                  # Discounting rate

# Exploration parameters
epsilon = 1.0                 # Exploration rate
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.01            # Minimum exploration probability
decay_rate = 0.005             # Exponential decay rate for
exploration prob
```

We train the model for 15000 iterations, and gradually reduce the exploration rate so that efficiency of the agent improves.

```
## Step 4 : Q-Learning Algorithm

# List of rewards
rewards = []
```

```
#until learning is stopped
for episode in range(total_episodes):
    # Reset the environment
    state = env.reset()
    step = 0
    done = False
    total_rewards = 0

    for step in range(max_steps):
        #Choose an action a in the current world state (s)
        exp_exp_tradeoff = random.uniform(0, 1)

        ## If this number > greater than epsilon --> exploitation
(taking the biggest Q value for this state)
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])

        # Else doing a random choice --> exploration
        else:
            action = env.action_space.sample()

        # Take the action (a) and observe the outcome state(s') and
reward (r)
        new_state, reward, done, info = env.step(action)

        # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a')
- Q(s,a)]
        # qtable[new_state,:] : all the actions we can take from new
state
        qtable[state, action] = qtable[state, action] + learning_rate
* \
                                                        (reward +
gamma * np.max(qtable[new_state, :]) - qtable[state, action])

        total_rewards += reward

        # Our new state is state
        state = new_state
```

```
        # If done (if we're dead) : finish episode
        if done == True:
            break

    # Reduce epsilon (because we need less and less exploration)
    epsilon = min_epsilon + (max_epsilon - min_epsilon) *
np.exp(-decay_rate * episode)

    rewards.append(total_rewards)

print ("Score over time: " +  str(sum(rewards)/total_episodes))
print(qtable)
```

The q learning model is coded from scratch.

```
for episode in range(150):
    state = env.reset()
    step = 0
    done = False
    print("****************************************************")
    print("EPISODE ", episode)

    for step in range(max_steps):

        # Take the action (index) that have the maximum expected future reward
given that state
        action = np.argmax(qtable[state,:])

        new_state, reward, done, info = env.step(action)

        if done:
            # Here, we decide to only print the last state (to see if our agent
is on the goal or fall into an hole)
            env.render()

            # We print the number of steps it took.
            print("Number of steps", step)
            break
        state = new_state
env.close()
```

Episodes are initiated to start the training- after multiple cycles the model uses the reward and tries to perform optimally.

Inference :

The Frozen Lake game we implemented was an example of using RL in games. Once the agent has learnt the environment, our model converges to a point where a random action is not required. We can trace the state at every iteration of every episode to see how the weights vary. The key in the off-policy method is the allowance of the greedy action and the move to a next state. Since we allow this action, the agent will converge faster at the local minima and learn the environment sooner than an on-policy method.

7. What's in it for us? (Future Scope)

- Optimizing Compilers: It's framable as an RL problem aiming to *optimize* for the compiler optimization strategy tailored to a specific intermediate representation(IR). The states get represented by the current code IR and history of optimization phases. The reward depends on the runtime of the new IR compared to the previous state IR. The future of RL in compiler optimization will also involve agents that can meta-learn or do self-supervised learning in new target systems. That would save on computational resources as the current design requires programs to be compiled, and benchmarked for performance on unseen systems.

- Recommender systems: Recommendations help personalize a user's preferences. In music and video platforms, for example, recommendations help members find entertainment to engage with and enjoy while maximizing satisfaction and user retention. RecSys approaches like filtering, causality or bandits may have the ability to remember historical experiences from user interactions and be able to play them back when needed. But they are restricted in the sense that they cannot plan.This need for the ability to plan is a motivation for using RL in recommender systems. The key benefit of RL is that it personalises for the user to maximize long term satisfaction.

- RL based Database systems: Database queries and configurations are representable as Markov Decision Processes (MDPs) that optimize database access. For example, in the ordering of join queries, we can frame the MDP as:
  State: The set of relations to be joined
  Action: A valid join between any two relations
  Next State: Present record-relations after two joined relations get removed from the set
  Reward: Cost estimate of the join

- AI Feynman 2.0: The paper is AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity, submitted June 18th, 2020. The first author is Silviu-Marian Udrescu, who was generous enough to hop on a call with me and explain the backstory of this new machine learning library. The library, called AI Feynman 2.0, helps to fit regression formulas to data. More specifically, it helps to fit formulas to data at different levels of complexity (defined in bits). The user can select the operators that the solver will use from sets of operators, and the solver does its thing. Operators are things like exponentiation, cos, arctan, and so on.

- Reinforcement learning and General AI: Titled "Reward is Enough," the paper, which is still in pre-proof as of this writing, draws inspiration from studying the evolution of natural intelligence as well as drawing lessons from recent achievements in artificial intelligence. The authors suggest that reward maximization and trial-and-error experience are enough to develop behavior that exhibits the kind of abilities associated with intelligence. And from this, they conclude that reinforcement learning, a branch of AI that is based on reward maximization, can lead to the development of artificial general intelligence. The researchers also discuss the reward-powered basis of language, social intelligence, imitation, and finally, general intelligence, which they describe as "maximising a singular reward in a single, complex environment."

- Intelligently trade stocks: Supervised time series models can be used for predicting future sales as well as predicting stock prices. However, these models don't determine the action to take at a particular stock price. Using Reinforcement Learning (RL) this can be easily

corrected. An RL agent can decide on such a task; whether to hold, buy, or sell. The RL model is evaluated using market benchmark standards in order to ensure that it's performing optimally.

- Advanced self-driving cars: In self-driving cars, there are various aspects to consider, such as speed limits at various places, drivable zones, avoiding collisions—just to mention a few. Some of the autonomous driving tasks where reinforcement learning could be applied include trajectory optimization, motion planning, dynamic pathing, controller optimization, and scenario-based learning policies for highways. For example, parking can be achieved by learning automatic parking policies. Lane changing can be achieved using Q-Learning while overtaking can be implemented by learning an overtaking policy while avoiding collision and maintaining a steady speed thereafter.

- Smart traffic signals- Traffic congestion is increasing globally, and this problem needs to be addressed by the traffic management system. Traffic signal control (TSC) is an effective method among various traffic management systems. In a dynamically changing and interconnected traffic environment, the currently model-based TSCs are not adaptive. In addition, with the rise of smart cities and IoT, there is a need for efficient TSCs that can handle large and complex data. RL models can solve the issue here as well.

8. What are the Roadblocks?

Disadvantages of Reinforcement learning:

- Too much reinforcement learning can lead to an overload of states, which can diminish the results.
- The curse of dimensionality limits reinforcement learning heavily for real physical systems
- Reinforcement learning needs a lot of data and a lot of computation. It is data-hungry. That is why it works really well in video games because one can play the game again and again and again, so getting lots of data seems feasible.

- Reinforcement learning assumes the world is Markovian, which it is not. The Markovian model describes a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

To solve many problems of reinforcement learning, we can use a combination of reinforcement learning with other techniques rather than leaving it altogether. One popular combination is Reinforcement learning with Deep Learning.

9. Conclusion

We can conclude RL models learn by a continuous process of receiving rewards and punishments on every action taken, it is able to train systems to respond to unforeseen environments. Industrial systems, including supply chain management and industrial robotics, are good examples of large problems well spaces perfectly suited to be solved with reinforcement learning. With reinforcement learning, domain experts and organizations typically know what they want a system to do — but they want to automate or optimize a specific process. As a result, your system must be highly adaptive. Again, this is where reinforcement learning techniques are especially useful since they don't require lots of pre-existing knowledge or data to provide useful solutions.

10. References

- **A brief introduction to reinforcement learning (freecodecamp.org)**
- **Reinforcement Learning Tutorial - Javatpoint**
- **Frozen Lake: Beginners Guide To Reinforcement Learning With OpenAI Gym (analyticsindiamag.com)**
- **Bias and discrimination in AI: whose responsibility is it to tackle them? | VentureBeat**
- **Pros And Cons Of Reinforcement Learning – Pythonista Planet**