MACHINE LEARNING PROJECT REPORT : DYNAMIC MODELLING OF SEGWAYS USING REINFORCEMENT LEARNING

K Venkat Ramnan, PES 1201801319, Sec B , ECE 6th sem , March-April 2021
Guide : Prof Vanamala HR

1. INTRODUCTION

Artificial Intelligence (AI) is quickly becoming the paramount technology of the future. In fact it is termed as one of the major paradigm shifts in technology.Next-generation AI compels us to realize that machines do indeed think. Although machines do not think like us, their thought process has proven its efficiency in many areas and some in some situations much more capable than ours.

Among many methods used behind AI, supervised machine learning and unsupervised machine learning are predominantly used in many situations. But recently Reinforcement Learning (RL) came into light and is quickly becoming a mainstream in the field of AI and machine learning. This is mainly due to the fact that RL does not require data to model , it can explore space with a handful of instructions, analyze its surroundings one step at a time, and build data as it goes along for modeling.It achieves this by working in a trial and error fashion.

2. MOTIVATION

RL has the power to start a journey with no knowledge of what to try to do next. This approach requires constant trial and error because it collects data about its surroundings and figures out the way to accomplish its goal. This exposes interesting possibilities, what about recording additional information, like environmental details along the way that it's going to not fully understand until after it reaches its goal? And once reached, could it review that additional data to work out if any of it might have helped it reach its goal faster? The answers to these questions can be achieved using this method.It is believed that RL will take us a step closer to achieving true artificial general intelligence. This will be the first step in the learning process behind this technology.

3. UNDERSTANDING REINFORCEMENT LEARNING

RL is the science of making decisions. It is not strictly supervised as it does not rely only on a set of labelled training data but is not unsupervised learning because we have a reward which we want our agent to maximise. The agent needs to find the "right" actions to take in different situations to achieve its overall goal.Reinforcement learning involves no supervisor and only a reward signal is used for an agent to determine if they are doing well or not. Time is a key component in RL where the process is sequential with delayed feedback. Each action the agent makes affects the next data it receives.

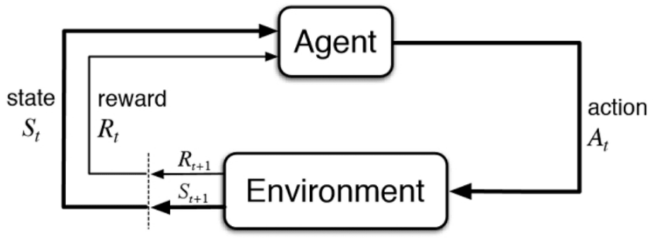


Fig 1. RL Explained

The major components of RL are:

Agent : software program that learns to make intelligent decisions.

Environment : world of the agent

State: A state is a position or a moment in the environment that the agent can be in.

Action : The agent interacts with the environment and moves from one state to another by performing an action.

Reward : Reward is a numerical value, +1 for right and -1 for wrong.

4. Q-LEARNING AND DEEP Q-LEARNING

Q-learning is one of the most used reinforcement learning algorithms. This is due to its ability to compare the expected utility of the available actions without requiring an environment model. Thanks to this technique, it is possible to find an optimal action for every given state in a finished MDP. A general solution to the reinforcement learning problem is to estimate, thanks to the learning process, an evaluation function. This function must be able to evaluate, through the sum of the rewards, the optimality/utility or otherwise of a particular policy. In fact, Q-learning tries to maximize the value of the Q function (action-value function), which represents the maximum discounted future reward when we perform actions a in the state s.

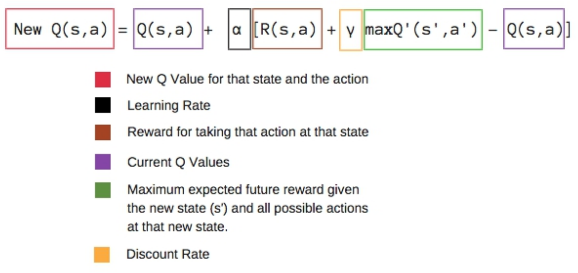

New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
🟥 New Q Value for that state and the action
⬛ Learning Rate
🟫 Reward for taking that action at that state
🟪 Current Q Values
🟩 Maximum expected future reward given the new state (s') and all possible actions at that new state.
🟧 Discount Rate

Fig 2. Q Learning Formula explained

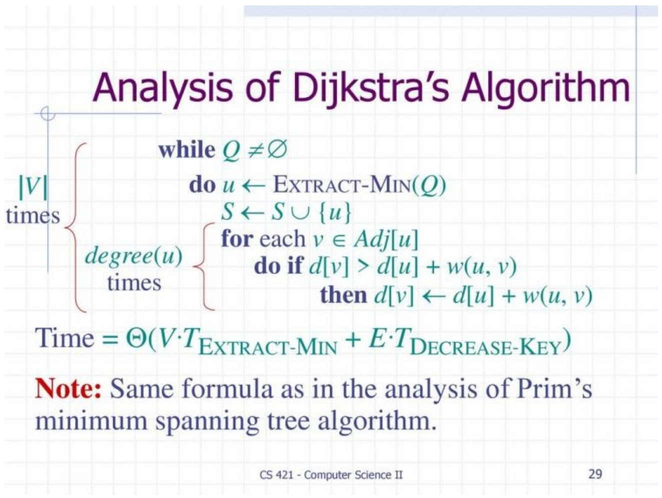
Deep Q-learning represents an evolution of the basic Q Learning method the state-action is replaced by a neural network, with the aim of approximating the optimal value function. Compared to the previous approaches, where it was used to structure the network in order to request both input and action and providing its expected return, Deep Q-learning revolutionizes the structure in order to request only the state of the environment and supply as many status-action values as there are actions that can be performed in the environment.

5. GRAPH THEORY

Graphs being data structures can be implemented directly in python. But for doing so, it will be required to not just code each function from scratch, but also it must be able to map to plotting. The python package NetworkX provides us this excellent feature for the creation, manipulation and analysis of structure, dynamics and functions of complex interconnected networks like graphs.

6. DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road (for simplicity, ignore red lights, stop signs, toll roads and other obstructions).



Analysis of Dijkstra's Algorithm

```
$|V|$ times {
    while $Q \neq \varnothing$
        do $u \leftarrow$ EXTRACT-MIN($Q$)
            $S \leftarrow S \cup \{u\}$
            $degree(u)$ times {
                for each $v \in Adj[u]$
                    do if $d[v] > d[u] + w(u, v)$
                        then $d[v] \leftarrow d[u] + w(u, v)$
```

Time = $\Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$

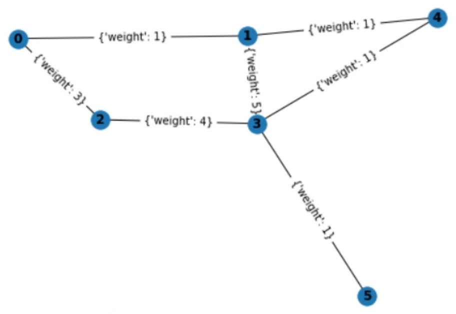
Note: Same formula as in the analysis of Prim's minimum spanning tree algorithm.

CS 421 - Computer Science II 29

Fig 3. Dijkstra Algorithm explained

7. CASE STUDY: SHORTEST PATH USING Q LEARNING

The Vehicle Routing Problem (VRP) ⇒ typical distribution and transport problem ⇒ optimizing the use of a set of vehicles with limited capacity to pick up and deliver goods or people to geographically distributed stations. Managing these operations in the best possible way can significantly reduce costs. So the case study shows how Q Learning can be used instead of Dijkstra's algorithm to find the shortest path. But it must be noted that for complex graphs we will need to go ahead with Deep Q Learning.

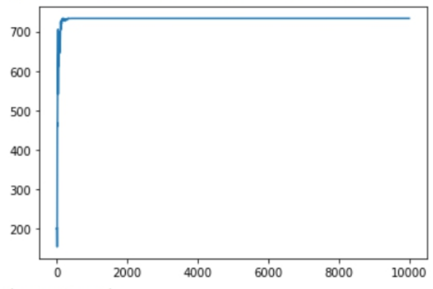


```
[0, 1, 4, 3, 5]
Time taken for Dijkstra Algorithm:
0.00018835067749023438
4
```

Fig 4. Graph using Dijkstra's and time

taken

```
Q matrix trained :
[[  0.    86.45  81.19   0.     0.     0.  ]
 [  0.     0.     0.    90.1   90.5    0.  ]
 [  0.     0.     0.    90.1    0.     0.  ]
 [  0.     0.     0.     0.     0.   100.  ]
 [  0.     0.     0.    95.     0.     0.  ]
 [  0.     0.     0.     0.     0.   100.  ]]
```



```
Shortest path:
[0, 1, 4, 3, 5]
Time Elapsed in finding shortest path:
0.00010275840759277344
```

Fig 5. Q Learning shortest path and time taken

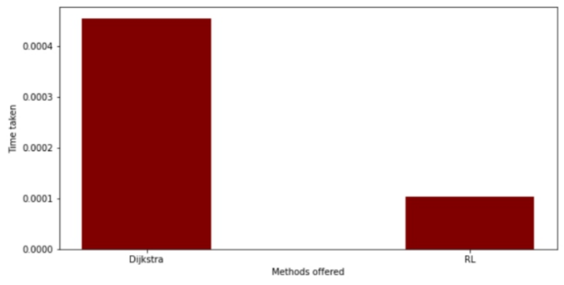


Fig 6. Comparison of Dijkstra and Q Learning

8. DYNAMIC MODELLING OF SEGWAY USING RL

8.1 How Segways work

The Segway was invented by Dean Kamen, who presented the prototype in 2001. It is an electric traction-transport vehicle for individual locomotion, a very advanced technological concept.To move the Segway PT forward or backwards, the driver simply leans forward or backwards. To turn left or right, the driver simple tilts the LeanSteer handlebars to the left or right. When the driver stands upright, the vehicle stops. Segway engineers call this "dynamic stabilisation". This body related steering belongs to the most intuitive worldwide and contributes substantially to the fun of driving.

8.2 System Modelling

The reverse pendulum represents a simple inverted pendulum, rigid, and without a fixed point. The lower part can therefore move to balance the oscillations of the highest part and thus ensure equilibrium; the control problem therefore leads back to wanting to stabilize the position of a rod constrained to a carriage free to move along a guide.

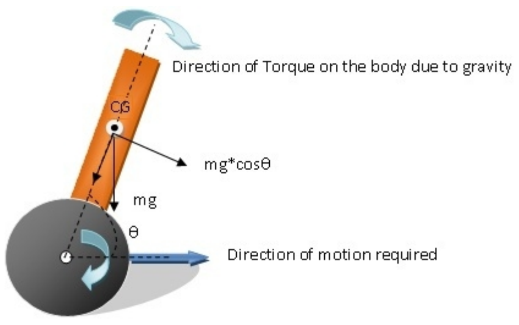

Direction of Torque on the body due to gravity
CG
mg*cosθ
mg
θ
Direction of motion required

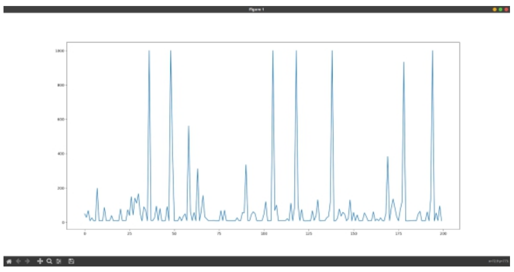
Fig 7. Physics behind System modelling

8.3 OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball. The gym library provides an easy-to-use suite of reinforcement learning tasks. Here we use the python implementation of gym.

8.4 Q-Learning Solution

The most demanding phase: the training of our system. The gym library is focused on the episodic setting of reinforced learning. The agent's experience is divided into a series of episodes. The initial state of the agent is randomly sampled by a distribution and the interaction proceeds until the environment reaches a terminal state. This procedure is repeated for each episode with the aim of maximizing the total reward expectation per episode and achieving a high level of performance in the fewest possible episodes. In the learning phase, we must estimate an evaluation function. This function must be able to evaluate, through the sum of the rewards, the convenience or otherwise of a particular policy. In other words, we must approximate the evaluation function. How can we do this? One solution is to use an artificial neural network as a function approximator. Recall that the training of a neural network aims to identify the weights of the connections between neurons. In this case, we will choose random values with weights for each episode. At the end, we will choose the combination of weights that will have collected the maximum reward. The state of the system at a given moment is returned to us by the observation object. To choose an action from the actual state, we can use a linear combination of the weights and the observation. This is one of the most important special cases of function approximation, in which the approximate function is a linear function of the weight vector, w. For every state, s, there is a real-valued vector, x(s), with the same number of components as w. Linear methods approximate the state-value function by the inner product between w and x(s). This explains the working of the Q Learning methodology.



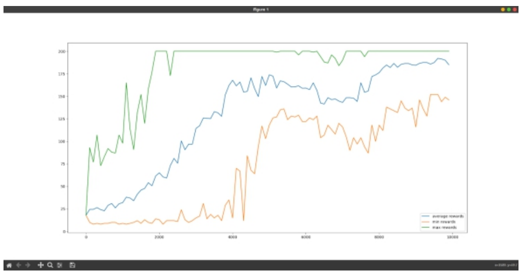


Fig 8. Plots of iterations vs rewards for 100 and 10000 epochs

9. EXPERIMENT SETUP

For both the experiments of Dynamic Modelling and the shortest path, the programs were coded in Python Language. Since the usage of OpenAI is more supportive in Linux based systems, the codes were run on Ubuntu 18.04 system. To achieve good results in RL, a minimum of 5000 epochs of training are required. So a powerful system with a 8GB RAM and 4GB GPU is used. The codes, videos and report are hosted at : https://github.com/venkatramnank/MLProject_reinforcement_learning.

10. CONCLUSION AND FUTURE SCOPE:

Basic concepts of RL and Q Learning were learned.Applications in balancing of physics based tools as well as in finding shortest path is explored. The future scope of RL to be learned include Inverse reinforcement learning, learning by demonstration, reinforcement learning from human preferences, DDPG, and Hindsight Experience Replay.

ACKNOWLEDGMENTS

We would like to thank Prof Vanamala HR for guiding us throughout the concepts of Machine Learning as well as throughout the project. We would also like to extend our regards to ECE Dept HOD Dr. Anuradha M. Finally we would like to extend our thanks to PES University.

REFERENCES

[1] Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

[2] Keras Reinforcement Learning Projects , Giuseppe Ciaburro

[3] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

[4] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[5] https://gym.openai.com/docs/

[6] https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

[7] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[8] https://heartbeat.fritz.ai/automating-an-ai-to-find-the-shortest-route-using-reinforcement-learning-19dc9a3c0411?gi=1ac3a024b9a0

[9] Github Based Codes

APPENDIX

A. Code for Shortest Distance

```
# Implementing Dijkstra's Algorithm
import time
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()
G.add_node(0)
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_edge(0,1,weight=1)
G.add_edge(0,2,weight=3)
G.add_edge(1,3,weight=5)
G.add_edge(2,3,weight=4)
G.add_edge(1,4,weight=1)
G.add_edge(3,4,weight=1)
G.add_edge(3,5,weight=1)
pos=nx.spring_layout(G,scale=3)
nx.draw(G,pos,with_labels=True,font_weight='bold')
edge_labels=nx.get_edge_attributes(G,'r')
nx.draw_networkx_edge_labels(G,pos)
plt.show()
start=time.time()
print(nx.shortest_path(G,0,5,weight='weight'))
end=time.time()
print('Time taken for Dijkstra Algorithm:')
print(end-start)
dijkstra_time=end-start
print(nx.nx.shortest_path_length(G,0,5,weight='weight'))
# Q Learning Approach
RMatrix = np.matrix([ [-1,50,1,-1,-1,-1],
          [-1,-1,-1,1,50,-1],
          [-1,-1,-1,1,-1,-1],
          [-1,-1,-1,-1,-1,100],
          [-1,-1,-1,50,-1,-1],
          [-1,-1,-1,-1,-1,100] ])
QMatrix = np.matrix(np.zeros([6,6]))
gamma = 0.9
InitialState = 0
def AllActions(state):
  CurrentState = RMatrix[state,]
  AllAct = np.where(CurrentState >= 0)[1]
  return AllAct
AvAct = AllActions(InitialState)
def NextAction(AvActRange):
  NextAct = int(np.random.choice(AvAct,1))
  return NextAct
Action = NextAction(AvAct)
def Update(CurrentState, Action, gamma):

  MaxIndex = np.where(QMatrix[Action,] ==
np.max(QMatrix[Action,]))[1]
  if MaxIndex.shape[0] > 1:
    MaxIndex = int(np.random.choice(MaxIndex, size = 1))
  else:
    MaxIndex = int(MaxIndex)
  MaxValue = QMatrix[Action, MaxIndex]

  QMatrix[CurrentState, Action] = RMatrix[CurrentState, Action] +
gamma * MaxValue
  #print('max_value', RMatrix[CurrentState, Action] + gamma *
MaxValue)
  if (np.max(QMatrix) > 0):
   return(np.sum(QMatrix/np.max(QMatrix)*100))
  else:
   return (0)
Update(InitialState,Action,gamma)
scores=[]
for i in range(10000):
  CurrentState = np.random.randint(0, int(QMatrix.shape[0]))
  AvAct = AllActions(CurrentState)
  Action = NextAction(AvAct)
  score=Update(CurrentState,Action,gamma)
  scores.append(score)
  #print ('Score:', str(score))

print("Q matrix trained :")
print(QMatrix/np.max(QMatrix)*100)
CurrentState = 0
Steps = [CurrentState]
while CurrentState != 5:
  startr=time.time()
  NextStepIndex = np.where(QMatrix[CurrentState,] ==
np.max(QMatrix[CurrentState,]))[1]

  if NextStepIndex.shape[0] > 1:
    NextStepIndex = int(np.random.choice(NextStepIndex, size = 1))
  else:
    NextStepIndex = int(NextStepIndex)
   Steps.append(NextStepIndex)
   CurrentState = NextStepIndex
endr=time.time()
plt.plot(scores)
plt.show()
print("Shortest path:")
print(Steps)
print('Time Elapsed in finding shortest path:')
print(endr-startr)
Reinforcment_time=endr-startr
```

B. Code for Segway Modelling

```
import gym
import numpy as np
import time
import matplotlib.pyplot as plt
env=gym.make('CartPole-v0')
# Contains the maximum reward obtained up to the current episode
HighReward=0
# BestWeights variable will contain sequence of weights that will have
the maximum reward
BestWeights = None
scores=[]
for i in range(200):
   observation=env.reset()
   # sequence of weights equal in number to the observations of the
environment,
   # which is four (cart position, cart velocity, pole angle, and pole
velocity at tip)
   Weights=np.random.uniform(-1,1,4)
   SumReward = 0
   for j in range(1000):
      env.render()
      #time.sleep(0.05)
      # if product is <0, the action is 0 (move left); otherwise, the action
is 1 (move right)
      action = 0 if np.matmul(Weights,observation) < 0 else 1
      observation,reward,done,info = env.step(action)
      SumReward+= reward
   print(i,j,Weights,observation,action,SumReward,BestWeights)
   if SumReward > HighReward:
      HighReward = SumReward
      BestWeights = Weights
   scores.append(SumReward)
print(scores)
x=[i for i in range(200)]
plt.plot(x,scores)
plt.show()
```