REINFORCEMENT LEARNING

YEWANDE OGUNBOWALE

March 8, 2021

## MDP APPROACH FOR 2 X 2 GRID EXAMPLE

Example; Learning an agent travelling through a 2\*2 grid (4 states) Red wall prevents direct moves from S1 to S2 States S1, S2, S3 give reward of -1; state S4 gives reward of +10 Action: Left, Right Up, and Down

#LIBRARY  
library(MDPtoolbox)

## Loading required package: Matrix

## Loading required package: linprog

## Loading required package: lpSolve

# MDP policy Iteration

# These transistion matrices reflect the following randomness of action taken vis a vis an action, there is a 70% probability that that action selected will occur. There is a 20% probability the agent will stay in the same (no action taken). And there is a 10% probability that the agent will move in lateral direction to the action selected.

library(devtools)

## Loading required package: usethis

library(ReinforcementLearning)

# 1. Defining the Set of Actions - Left, Right, Up and Down for 2 x 2 matrix

#Up Action  
up=matrix(c( 1, 0, 0, 0,  
 0.7, 0.2, 0.1, 0,  
 0, 0.1, 0.2, 0.7,  
 0, 0, 0, 1),  
 nrow=4,ncol=4,byrow=TRUE)  
  
  
#Agent is in: S1 S2 S3 S4  
#Up <- matrix(c( 1, 0, 0, 0,  
#Goes to: 0.7, 0.2, 0.1, 0,  
#Goes to: 0, 0.1, 0.2, 0.7,  
#Goes to: 0, 0, 0, 1),  
 # nrow=4,ncol=4,byrow=TRUE)   
  
#If agent is in state S1 and tries to go up, there is a 100% probability s/he will remain in State S1  
  
  
  
up=matrix(c( 1, 0, 0, 0,  
# If agent is in state S2 and tries to go up, there is a 70% prob that s/he will go up to S1, a 20% prob will remain in S2, and a 10% prob agent will go right to S3.  
 0.7, 0.2, 0.1, 0,  
# If agent is in State S3 and tries to go up, there is a 70% prob that s/he will go up to S4, a 20% prob will remain in S3, and a 10% prob s/he will move left to S2  
 0, 0.1, 0.2, 0.7,  
#Finally, if agent is in State S4 and tries to go Up, there is a 100% prob agent will remain in S4.  
 0, 0, 0, 1),  
 nrow=4,ncol=4,byrow=TRUE)  
  
  
  
  
#Down Action  
down=matrix(c(0.3, 0.7, 0, 0,  
 0, 0.9, 0.1, 0,  
 0, 0.1, 0.9, 0,  
 0, 0, 0.7, 0.3),  
 nrow=4,ncol=4,byrow=TRUE)  
  
  
  
  
#Left Action  
left=matrix(c( 0.9, 0.1, 0, 0,  
 0.1, 0.9, 0, 0,  
 0, 0.7, 0.2, 0.1,  
 0, 0, 0.1, 0.9),  
 nrow=4,ncol=4,byrow=TRUE)  
  
  
#Right Action  
right=matrix(c( 0.9, 0.1, 0, 0, 0.1, 0.2, 0.7, 0,  
 0, 0, 0.9, 0.1,  
 0, 0, 0.1, 0.9),  
 nrow=4,ncol=4,byrow=TRUE)  
  
  
  
#Aggregate previous matrices to create transistion probabilities into list T  
T <- list(up=up, left=left, down=down, right=right)  
T

## $up  
## [,1] [,2] [,3] [,4]  
## [1,] 1.0 0.0 0.0 0.0  
## [2,] 0.7 0.2 0.1 0.0  
## [3,] 0.0 0.1 0.2 0.7  
## [4,] 0.0 0.0 0.0 1.0  
##   
## $left  
## [,1] [,2] [,3] [,4]  
## [1,] 0.9 0.1 0.0 0.0  
## [2,] 0.1 0.9 0.0 0.0  
## [3,] 0.0 0.7 0.2 0.1  
## [4,] 0.0 0.0 0.1 0.9  
##   
## $down  
## [,1] [,2] [,3] [,4]  
## [1,] 0.3 0.7 0.0 0.0  
## [2,] 0.0 0.9 0.1 0.0  
## [3,] 0.0 0.1 0.9 0.0  
## [4,] 0.0 0.0 0.7 0.3  
##   
## $right  
## [,1] [,2] [,3] [,4]  
## [1,] 0.9 0.1 0.0 0.0  
## [2,] 0.1 0.2 0.7 0.0  
## [3,] 0.0 0.0 0.9 0.1  
## [4,] 0.0 0.0 0.1 0.9

#T = Transistion probability  
  
  
  
#Create matrix with rewards:  
  
R=matrix(c( -1, -1, -1, -1,  
 -1, -1, -1, -1,  
 -1, -1, -1, -1,  
 10, 10, 10, 10),  
 nrow=4,ncol=4,byrow=TRUE)  
  
#R = Reward matrix  
  
  
  
#Check if this provides a well-defined MDP  
mdp\_check(T, R) # empty string ==> ok

## [1] ""

#Policy iteration with discount factor g = 0.9  
m <- mdp\_policy\_iteration(P=T,  
 R=R,  
 discount=0.9)  
  
  
#Display optimal policy p  
m$policy

## [1] 3 4 1 1

names(T)[m$policy]

## [1] "down" "right" "up" "up"

#Display value funtion vp  
m$V

## [1] 58.25663 69.09102 83.19292 100.00000

#The value shows the movement of following this policy as I move from state to state.  
  
  
#REINFORCEMENT LEARNING  
  
# Viewing the pre-built function for each state, action and reward  
  
env <- gridworldEnvironment  
print(env)

## function (state, action)   
## {  
## next\_state <- state  
## if (state == state("s1") && action == "down")   
## next\_state <- state("s2")  
## if (state == state("s2") && action == "up")   
## next\_state <- state("s1")  
## if (state == state("s2") && action == "right")   
## next\_state <- state("s3")  
## if (state == state("s3") && action == "left")   
## next\_state <- state("s2")  
## if (state == state("s3") && action == "up")   
## next\_state <- state("s4")  
## if (next\_state == state("s4") && state != state("s4")) {  
## reward <- 10  
## }  
## else {  
## reward <- -1  
## }  
## out <- list(NextState = next\_state, Reward = reward)  
## return(out)  
## }  
## <bytecode: 0x000000001c896468>  
## <environment: namespace:ReinforcementLearning>

states <- c("S1", "S2", "S3", "S4")  
states

## [1] "S1" "S2" "S3" "S4"

actions <- c("up", "down", "left", "right")  
actions

## [1] "up" "down" "left" "right"

# Sample N = 1000 random sequences from the environment  
  
#Data format must be (s, a, r,s\_new) tuples  
#as row in a dataframe structure.  
  
data <- sampleExperience(N = 1000,  
 env = env,  
 states = states,  
 actions = actions)  
  
  
head(data)

## State Action Reward NextState  
## 1 S4 up -1 S4  
## 2 S3 down -1 S3  
## 3 S3 down -1 S3  
## 4 S2 down -1 S2  
## 5 S2 left -1 S2  
## 6 S4 right -1 S4

# Define reinforcement learning parameter  
control <- list(alpha = 0.1, #low learning rate  
 gamma = 0.5, #middle discount factor  
 epsilon = 0.1) #low exploration factor  
  
control

## $alpha  
## [1] 0.1  
##   
## $gamma  
## [1] 0.5  
##   
## $epsilon  
## [1] 0.1

#MODEL  
  
model <- ReinforcementLearning(data,   
 s = "State",  
 a = "Action",  
 r = "Reward",  
 s\_new = "NextState",  
 control = control)  
  
#Print result  
print(model)

## State-Action function Q  
## right up down left  
## S1 -1.919198 -1.919095 -1.924957 -1.922008  
## S2 -1.886576 -1.875725 -1.839550 -1.871030  
## S3 -1.865349 -1.871967 -1.824223 -1.852898  
## S4 -1.870010 -1.843684 -1.846507 -1.876602  
##   
## Policy  
## S1 S2 S3 S4   
## "up" "down" "down" "up"   
##   
## Reward (last iteration)  
## [1] -1000

#From the table above, the reinforcement learning program infers the best policy. The best policy is if you're in S1 go up, if you're in S2 go left, if you're in S3 down, if you,re in S4 go right. This is the most important output from the reinforment learning function.

# CONCLUSION

At present, machines are adept at performing repetitive tasks and solve complex problems easily but cannot solve easy tasks without getting into complexity. This is why, making machines perform simple tasks such as walking, moving hands or even playing tic-tac-toe is very difficult though we, as humans, perform this every day without much effort. With reinforcement learning, these tasks can be trained with an order of complexity.