Project 2: Reinforcement Learning

John Greaney-Cheng johngc@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

My algorithm is a slight modification of the $Q-$Learning algorithm. The $Q-$Learning algorithm starts with an initial guess action-value function and updates it by iterating through the sample data. The $Q-$Learning update rule is

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]$$

Below is the initial guess action-value function and some hyperparameters we need for the update

Initial Guess Action-Value Function: $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$, $Q(s,a) = 5$
This initial value of 5 encourages exploration of the Action Space
State Space Size $|\mathcal{S}| = 100, 50000, 302020$ for small, medium, large respectively
Action Space Size $|\mathcal{A}| = 4, 7, 9$ for small, medium, large respectively
Discount Factor $\gamma = 0.95, 1.0, 0.95$ for small, medium, large respectively
Learning Rate $\alpha = 0.1$
Number of Epochs = 10

Once we've initialized everything, we start updating by iterating through the data. Every time we go through through the entire dataset, we call that an epoch, so we go through the entire dataset 10 times. Each time we go through the dataset, we start by shuffling the row order of the dataset. This is to help avoid any biases that may come from the original data order which makes the learning process more balanced and random.

We can then iterate through the shuffled rows. Each row contains a (s, a, r, s') tuple. We use this tuple along with our current $Q(s,a)$ and the hyperparameters defined above to update $Q(s,a)$ with the $Q-$Learning update rule. After iterating though all the shuffled rows, we then start a new epoch, reshuffle the rows, and iterate through the rows in that new shuffled order. This continues until we iterate through the entire dataset 10 times, after which we stop updating $Q(s,a)$. We then return the following policy

$$\pi(s) = \arg\max_a Q(s,a)$$

The algorithm we just described works for small and large.csv. However, medium.csv required some extra modifications to the $Q-$Learning algorithm for a policy that performed better than the baseline policy. One detail I noticed about medium.csv was that there were states that never appeared in the s column. This given with the context of the problem, trying to get to a destination, implies those states were terminal. This means if s' is a terminal state, then the $Q-$Learning update rule shouldn't account for discounted rewards, because there won't be any more rewards once s' is reached. I modified the $Q-$Learning

update rule to account for this. If s' is a terminal state, meaning that the state in the s' column is never found in the s column for any row, then

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r - Q(s,a)\Big]$$

If s' isn't a terminal state, then the update rule is the same as before. The other modification I made is modifying the reward value just for medium.csv. This is because the context of medium.csv is choosing accelerations to drive to a destination. If the destination is far away enough, then there's not enough feedback about whether it's moving in the right direction, so the action-value function struggle to learn useful actions for states far away from the goal (terminal states). I took a look at the terminal states to determine the goal and found that the goal had a high position. To fix this, I modified the rewards by adding the position of the state (scaled properly) to encourage accelerations that move towards the goal

$$\text{pos} = (s-1) \bmod 500$$

$$r \leftarrow r + 0.1 \times \frac{\text{pos}}{500}$$

This modified version of $Q-$Learning produced a policy for medium.csv that exceeded the baseline policy. Below are the runtimes for small.csv, medium.csv and large.csv.

Run time for small.csv: 5 seconds
Run time for medium.csv: 6 seconds
Run time for large.csv: 6 seconds

2. Code

```
using CSV, DataFrames, Random, Printf

# Q-learning model
mutable struct QLearning
    S       # state space (assumed to be 1:nstates)
    A       # action space (assumed to be 1:nactions)
    gamma   # discount factor
    Q       # Q(s,a) - action-value function
    alpha   # learning rate
end

# Q-learning update
function update!(model::QLearning, s, a, r, s, alpha)
    gamma, Q = model.gamma, model.Q
    # Q[s,a]  Q[s,a] + alpha * (r + gamma * max_a' Q[s,a]  Q[s,a])
    Q[s, a] += alpha * (r + gamma * maximum(Q[s, :]) - Q[s, a])
    return model
end

# Train Q-learning model to find optimal Q(s,a) function
function train_qlearning(csvfile::String)
    # Read input file and initialize hyperparameters
```

```
df = CSV.read(csvfile, DataFrame)
alpha = 0.1
nepochs = 10

if csvfile == "small.csv"
    nstates = 100
    nactions = 4
    gamma = 0.95
elseif csvfile == "medium.csv"
    nstates = 50000
    nactions = 7
    gamma = 1.0
elseif csvfile == "large.csv"
    nstates = 302020
    nactions = 9
    gamma = 0.95
end

# Initialize Q(s,a) = 5 for all (s,a) to encourage exploration
model = QLearning(1:nstates, 1:nactions, gamma, fill(5.0, nstates,
nactions), alpha)

# Find states that appear in the s column (used to detect states that don
't)
nonterminal_states = Set(df.s)

for epoch in 1:nepochs
    println("Epoch $epoch")

    # Shuffle input data order to avoid sequential bias
    shuffled = df[shuffle(axes(df, 1)), :]

    for row in eachrow(shuffled)
        s = Int(row.s)
        a = Int(row.a)
        r = Float64(row.r)
        sp = Int(row.sp)

        # Modified rewards for medium.csv
        if csvfile == "medium.csv"
            pos = (s - 1) % 500
            r += 0.1 * (pos / 500.0)
        end

        # Non-modified update for non-terminal s'
        if sp in nonterminal_states
            update!(model, s, a, r, sp, alpha)
        else
            # Modified update for terminal s': no next Q
            model.Q[s, a] += alpha * (r - model.Q[s, a])
```

```
            end
        end
    end

    return model
end


# Save policy as text in output file
function save_policy(model::QLearning, file_name::String)
    open(file_name, "w") do f
        for s in model.S
            best_action = argmax(model.Q[s, :])
            write(f, @sprintf("%d\n", best_action))
        end
    end
end


if length(ARGS) != 2
    error("usage: julia project2.jl <infile>.csv <outfile>.policy")
end

inputfilename = ARGS[1]
outputfilename = ARGS[2]
model = train_qlearning(inputfilename)
save_policy(model, outputfilename)
```