# Reinforcement Learning HW3

Hadi Askari

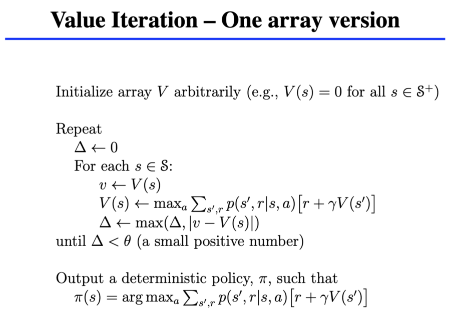
27th April 2023

## 1 Implementation

The code runs in approximately 5 seconds. It has two parts:

### 1.1 Value Iteration

We write the code to implement the following algorithm:



**Value Iteration – One array version**

```
Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

Figure 1: Algorithm for Value Iteration

First we initialized the Value Function and policy to zero. Then until the delta between the value function is greater than theta (we set to 0.001) we run the algorithm. For each set of valid states (1 to 9) we first save the old V value and then create an array of all possible actions (betting from 1 to 9). We then initialize our q_values to zero and update them for each action depending on the win and loss probabilities. We then update the value functions with the max of q_values and the policy with the action correlating to the argmax of the q_values.

### 1.2 Policy Iteration

This is a combination of policy evaluation and policy improvement. We implement the following algorithm.

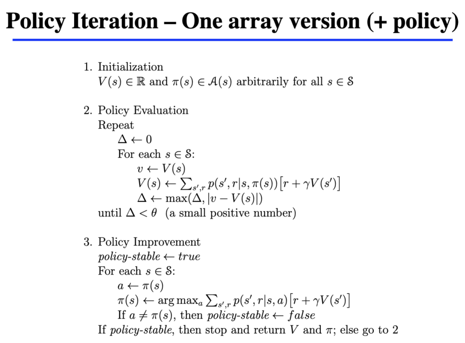
We initialize V with zero and pick a random policy (the aggressive policy from last assignment). After that we perform steps 2 (policy evaluation that we implemented in the last assignment) and 3 (policy improvement) until our optimal policy becomes stable (this occurs when the policy values stop changing).

In policy improvement we act greedily with respect to the policy for each state. We do this by taking the argmax of the actions for each state.

## 2 Policy and Value Function Values

The following are our final values:

**Optimal Policy for Value Iteration:**
1 2 3 4 5 6 7 2 1
**Optimal Value Function for Value Iteration:**
-0.9979 -1.9792 -2.8800 -3.7920 -4.0000 -4.8000 -5.6000 -5.9200 -6.1280



**Policy Iteration – One array version (+ policy)**

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
       $\Delta \leftarrow 0$
       For each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$  (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ true
   For each $s \in \mathcal{S}$:
       $a \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
       If $a \neq \pi(s)$, then policy-stable $\leftarrow$ $false$
   If policy-stable, then stop and return $V$ and $\pi$; else go to 2
```

Figure 2: Algorithm for Policy Iteration

**Optimal Policy for Policy Iteration:**
1 1 1 1 1 1 1 1 8
**Optimal Value Function for Policy Iteration:**
13.9268 14.5854 13.7696 12.7901 11.7924 10.7927 9.7927 8.7927 7.7927