

## Q Player's Interleaved Tabular Q-Learning Algorithm

Can be found in the file `./tictactoe/players.py`. This contains a class `QPlayer` which contains the functions `train_single_game` and `train` that implement the following pseudo-code.

---

### Algorithm 1 Q Player's interleaved tabular Q-Learning algorithm

---

```

1: Initialize  $Q_1, Q_2$  arbitrarily
2: for a total of  $N$  games do
3:   Initialize  $s$  as the empty board
4:   Take  $\epsilon$ -max-greedy action  $a$  and observe next state  $\tilde{s}$ 
5:   Take  $\epsilon$ -min-greedy action  $\tilde{a}$  and observe next state  $s'$ 
6:   while  $s'$  non-terminal do
7:     if in state  $s$  Player 1 is to move then
8:        $Q_1(s, a) \leftarrow Q_1(s, a) + \alpha [0 + \gamma \max_{a'} Q_1(s', a') - Q_1(s, a)]$ 
9:       Take  $\epsilon$ -max-greedy action  $a'$  and observe next state  $\tilde{s}'$ 
10:    else if in state  $s$  Player 2 is to move then
11:       $Q_2(s, a) \leftarrow Q_2(s, a) + \alpha [0 + \gamma \min_{a'} Q_2(s', a') - Q_2(s, a)]$ 
12:      Take  $\epsilon$ -min-greedy action  $a'$  and observe next state  $\tilde{s}'$ 
13:     $(s, a, \tilde{s}, \tilde{a}, s', a', \tilde{s}') \leftarrow (\tilde{s}, \tilde{a}, s', a', \tilde{s}')$ 
14:    If Player 1 won, do  $r \leftarrow 1$ , else if Player 2 won, do  $r \leftarrow -1$ , else, do  $r \leftarrow 0$ 
15:    if in state  $s$  Player 1 is to move then
16:       $Q_1(s, a) \leftarrow Q_1(s, a) + \alpha [r - Q_1(s, a)]$ 
17:       $Q_2(\tilde{s}, \tilde{a}) \leftarrow Q_2(\tilde{s}, \tilde{a}) + \alpha [r - Q_2(\tilde{s}, \tilde{a})]$ 
18:    else if in state  $s$  Player 2 is to move then
19:       $Q_2(s, a) \leftarrow Q_2(s, a) + \alpha [r - Q_2(s, a)]$ 
20:       $Q_1(\tilde{s}, \tilde{a}) \leftarrow Q_1(\tilde{s}, \tilde{a}) + \alpha [r - Q_1(\tilde{s}, \tilde{a})]$ 
21: return  $Q_1, Q_2$ 

```

---

**Remark.** Given a game state  $s$ , the  $\epsilon$ -max-greedy action is a random action with probability  $\epsilon$  or equal to  $\arg \max_a Q_1(s, a)$  with probability  $1 - \epsilon$ . The  $\epsilon$ -min-greedy action is a random action with probability  $\epsilon$  or equal to  $\arg \min_a Q_1(s, a)$  with probability  $1 - \epsilon$ .

**Parameters.** Have to be chosen at the beginning.

1. Step-size  $\alpha \in (0, 1]$
2. Greedy-factor  $\epsilon \in [0, 1]$
3. Discount factor  $\gamma$
4. Number of games  $N \in \mathbb{N}$ .

## TD Player's Adjusted Tabular TD( $\lambda$ ) Algorithm

Can be found in the file `./tictactoe/players.py`. This contains a class `TDPlayer` which contains the functions `train_single_game` and `train` that implement the following pseudo-code.

---

**Algorithm 2** TD Player's adjusted tabular TD( $\lambda$ ) algorithm

---

```
1: Initialize  $V$  arbitrarily
2: Choose  $\alpha \in (0, 1]$  and  $\epsilon \in [0, 1]$ 
3: for a total of  $N$  games do
4:   Initialize  $s$  as the empty board
5:   Take  $\epsilon$ -max-min-greedy action  $a$  and observe next state  $s'$ 
6:   Initialize empty dictionary  $\mathbf{z}$ 
7:   while  $s'$  non-terminal do
8:     If  $\mathbf{z}(s)$  is empty, do  $\mathbf{z}(s) \leftarrow 1$ , else do  $\mathbf{z}(s) \leftarrow \mathbf{z}(s) + 1$ 
9:     for all  $x$  in  $\mathbf{z}$  do
10:       $V(x) \leftarrow V(x) + \alpha[0 + \gamma V(s') - V(s)]\mathbf{z}(x)$ 
11:       $\mathbf{z}(x) \leftarrow \lambda\gamma\mathbf{z}(x)$ 
12:     Take  $\epsilon$ -max-min-greedy action  $a$  and observe next state  $s''$ 
13:      $(s, s') \leftarrow (s', s'')$ 
14:     If  $\mathbf{z}(s)$  is empty, do  $\mathbf{z}(s) \leftarrow 1$ , else do  $\mathbf{z}(s) \leftarrow \mathbf{z}(s) + 1$ 
15:     If Player 1 won, do  $r \leftarrow 1$ , else if Player 2 won, do  $r \leftarrow -1$ , else, do  $r \leftarrow 0$ 
16:     for all  $x$  in  $\mathbf{z}$  do
17:       $V(x) \leftarrow V(x) + \alpha[r - V(s)]\mathbf{z}(x)$ 
18:      $V(s') \leftarrow r(s')$ 
19: return  $V$ 
```

---

**Remark.** Given a game state  $s$ , the  $\epsilon$ -max-min-greedy action is a random action with probability  $\epsilon$  or with probability  $1 - \epsilon$  it is equal to  $\arg \max_a V(s_a)$  if it is Player 1's turn and  $\arg \min_a V(s_a)$  if it is Player 2's turn. State  $s_a$  is the state that follows after applying action  $a$  to  $s$ .

**Parameters.** Have to be chosen at the beginning.

1. Step-size  $\alpha \in (0, 1]$
2. Greedy-factor  $\epsilon \in [0, 1]$
3. Discount factor  $\gamma$
4. Number of games  $N \in \mathbb{N}$ .

## Deep Player's Adjusted Approximate Monte Carlo Algorithm

Can be found in the file `./tictactoe/players.py` and `./connectfour/players.py`. They contain a class `DeepPlayer` which contain the functions `train_single_game` and `train` that implement the following pseudo-code.

---

**Algorithm 3** Deep Player's adjusted approximate Monte Carlo algorithm

---

```
1: Initialize  $\theta \in \mathbb{R}^d$  arbitrarily
2: for  $i = 1, \dots, N$  do
3:   Initialize  $s_0$  as the empty board
4:   Generate a sequence of states  $s_0, \dots, s_T$  according to  $\epsilon$ -max-min-greedy policy
5:   If Eva won, do  $r \leftarrow 1$ , else if Odin won, do  $r \leftarrow -1$ , else, do  $r \leftarrow 0$ 
6:   for  $t = T, \dots, 0$  do
7:      $\theta \leftarrow \theta - \alpha [v(\theta, s_t) - (\gamma^{T-t}r + 1)/2] \nabla v(\theta, s_t)$ 
8:   if  $i \bmod n \equiv 0$  then
9:      $\alpha \leftarrow \alpha \cdot b$  and  $\epsilon \leftarrow \epsilon \cdot c$ 
10: return  $V$ 
```

---

**Remark.** Given a game state  $s$ , the  $\epsilon$ -max-min-greedy action is a random action with probability  $\epsilon$  or with probability  $1 - \epsilon$  it is equal to  $\arg \max_a V(s_a)$  if it is Player 1's turn and  $\arg \min_a V(s_a)$  if it is Player 2's turn. State  $s_a$  is the state that follows after applying action  $a$  to  $s$ .

**Parameters.** Have to be chosen at the beginning.

1. Step-size  $\alpha \in (0, 1]$
2. Greedy-factor  $\epsilon \in [0, 1]$
3. Discount factor  $\gamma$
4. Number of games  $N \in \mathbb{N}$ .
5. Decrease parameters  $n \in \mathbb{N}$ ,  $b, c \in [0, 1]$
6. Function approximator  $v$  which is based on a Neural network