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

Initialize Q_1, Q_2 arbitrarily
for a total of N games **do**
    Initialize s as the empty board
    Take ϵ-max-greedy action a and observe next state $\tilde{s}$
    Take ϵ-min-greedy action $\tilde{a}$ and observe next state s'
    while s' non-terminal **do**
        if in state s Player 1 is to move **then**
            $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\big[0 + \gamma \max_{a'} Q_1(s',a') - Q_1(s,a)\big]$
            Take ϵ-max-greedy action a' and observe next state $\tilde{s}'$
        else if in state s Player 2 is to move **then**
            $Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\big[0 + \gamma \min_{a'} Q_2(s',a') - Q_2(s,a)\big]$
            Take ϵ-min-greedy action a' and observe next state $\tilde{s}'$
        $(s, a, \tilde{s}, \tilde{a}, s') \leftarrow (\tilde{s}, \tilde{a}, s', a', \tilde{s}')$
    If Player 1 won, do $r \leftarrow 1$, else if Player 2 won, do $r \leftarrow -1$, else, do $r \leftarrow 0$
    if in state s Player 1 is to move **then**
        $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\big[r - Q_1(s,a)\big]$
        $Q_2(\tilde{s},\tilde{a}) \leftarrow Q_2(\tilde{s},\tilde{a}) + \alpha\big[r - Q_2(\tilde{s},\tilde{a})\big]$
    else if in state s Player 2 is to move **then**
        $Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\big[r - Q_2(s,a)\big]$
        $Q_1(\tilde{s},\tilde{a}) \leftarrow Q_1(\tilde{s},\tilde{a}) + \alpha\big[r - Q_1(\tilde{s},\tilde{a})\big]$
return Q_1, Q_2

Remark. Given a game state s, the ϵ-max-greedy action is a random action with probability ϵ or equal to $\arg\max_a Q_1(s,a)$ with probability $1-\epsilon$. The ϵ-min-greedy action is a random action with probability ϵ 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 γ
4. Number of games $N \in \mathbb{N}$.

TD Player's Adjusted Tabular TD(λ) 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(λ) algorithm

Initialize V arbitrarily
Choose $\alpha \in (0, 1]$ and $\epsilon \in [0, 1]$
for a total of N games **do**
  Initialize s as the empty board
  Take ϵ-max-min-greedy action a and observe next state s'
  Initialize empty dictionary $\mathbf{z}$
  while s' non-terminal **do**
    If $\mathbf{z}(s)$ is empty, do $\mathbf{z}(s) \leftarrow 1$, else do $\mathbf{z}(s) \leftarrow \mathbf{z}(s) + 1$
    for all x in $\mathbf{z}$ **do**
      $V(x) \leftarrow V(x) + \alpha[0 + \gamma V(s') - V(s)]\mathbf{z}(x)$
      $\mathbf{z}(x) \leftarrow \lambda\gamma\mathbf{z}(x)$
    Take ϵ-max-min-greedy action a and observe next state s''
    $(s, s') \leftarrow (s', s'')$
  If $\mathbf{z}(s)$ is empty, do $\mathbf{z}(s) \leftarrow 1$, else do $\mathbf{z}(s) \leftarrow \mathbf{z}(s) + 1$
  If Player 1 won, do $r \leftarrow 1$, else if Player 2 won, do $r \leftarrow -1$, else, do $r \leftarrow 0$
  for all x in $\mathbf{z}$ **do**
    $V(x) \leftarrow V(x) + \alpha[r - V(s)]\mathbf{z}(x)$
  $V(s') \leftarrow r(s')$
return V

Remark. Given a game state s, the ϵ-max-min-greedy action is a random action with probability ϵ 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 γ
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

Initialize $\theta \in \mathbb{R}^d$ arbitrarily
for $i = 1, ..., N$ **do**
  Initialize s_0 as the empty board
  Generate a sequence of states $s_0, ..., s_T$ according to ϵ-max-min-greedy policy
  If Eva won, do $r \leftarrow 1$, else if Odin won, do $r \leftarrow -1$, else, do $r \leftarrow 0$
  for $t = T, ..., 0$ **do**
    $\theta \leftarrow \theta - \alpha\big[v(\theta, s_t) - (\gamma^{T-t} r + 1)/2\big]\nabla v(\theta, s_t)$
  if $i \mod n \equiv 0$ **then**
    $\alpha \leftarrow \alpha \cdot b$ and $\epsilon \leftarrow \epsilon \cdot c$
return θ

Remark. Given a game state s, the ϵ-max-min-greedy action is a random action with probability ϵ 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 γ
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