# Reinforcement Learning Algorithms used in Tic Tac Toe

July 15, 2019

## 1 TD(0) Learning

**For** episode = 1, M **do**
  Initialize a fresh game
  **For** t = 1, T **do**
    Play move $a_t = \begin{cases} \text{random move} & \text{with probability } \epsilon \\ \text{argmax}_a \tilde{V}(succ(s_t, a), \theta) & \text{for white} \\ \text{argmin}_a \tilde{V}(succ(s_t, a), \theta) & \text{for black} \end{cases}$
    Receive reward $r_t$
    Store the transition $(s_t, s_{t+1}, r_t, a_t)$ in the replay buffer $D$
    Sample a random minibatch transition from $D$
    Set the TD-target: $y_t = \begin{cases} r_t & \text{if game terminates at step } t+1 \\ r_t + \gamma \tilde{V}(s_{t+1}, \theta) & \text{otherwise} \end{cases}$
    Perform a stochastic gradient decent step on $[y_t - V(s_t, \theta)]^2$ with respect to $\theta$
    Every $c$ steps set $\tilde{V} = V$
  **End For**
**End For**

$\gamma$ is the discount factor and $\theta$ the neural network parameters.

## 2 TD($\lambda$)

**procedure** REFRESH($l$)
  **For** transition $(s_t, s_{t+1}, r_t, R_t^\lambda, a_t) \in l$ processing back-to-front **Do**
    **If** terminal($s_{t+1}$) **Then**
      Update $R_t^\lambda \leftarrow r_t$
    **Else**
      Get adjacent transition $(s_{t+1}, s_{t+2}, r_{t+1}, R_{t+1}^\lambda, a_{t+1})$ from $l$
      Update $R_t^\lambda \leftarrow r_t + \gamma[\gamma R_{t+1}^\lambda + (1-\lambda)V(s_{t+1}, \theta)]$
    **End If**
  **End For**
**End procedure**

**For** episode = 1, M **do**
  Initialize a fresh game
  **For** t = 1, T **do**

$$\text{Play move } a_t = \begin{cases} \text{random move} & \text{with probability } \epsilon \\ \text{argmax}_a V(succ(s_t, a), \theta) & \text{for white} \\ \text{argmin}_a V(succ(s_t, a), \theta) & \text{for black} \end{cases}$$

    Receive reward $r_t$
    Append the transition $(s_t, s_{t+1}, r_t, R_t^\lambda, a_t)$ to $L$, where $R_t^\lambda$ is arbitrary
    **If** terminal($s_{t+1}$) **Then**
      REFRESH($L$)
      Store $L$ in D
    **End If**
    Sample a random minibatch transition from $D$
    Perform a stochastic gradient decent step on $[R_t^\lambda - V(s_t, \theta)]^2$ with respect to $\theta$
    Every $c$ steps REFRESH($D$)
  **End For**
**End For**

$\gamma$ is the discount factor and $\theta$ the neural network parameters.

# 3 Q-Learning

**For** episode = 1, M **do**
  Initialize a fresh game
  **For** t = 1, T **do**
    Play move $a_t = \begin{cases} \text{random move} & \text{with probability } \epsilon \\ \text{argmax}_a \tilde{Q}(s_t, a, \theta) & \text{for white} \\ \text{argmin}_a \tilde{Q}(s_t, a, \theta) & \text{for black} \end{cases}$
    Receive reward $r_t$
    Store the transition $(s_t, s_{t+1}, r_t, a_t)$ in the replay buffer $D$
    Sample a random minibatch transition from $D$
    Set the target: $y_t = \begin{cases} r_t & \text{if game terminates at step } t+1 \\ r_t + \gamma \text{max}_{a'} \tilde{Q}(s_{t+1}, a', \theta) & \text{otherwise} \end{cases}$
    Perform a stochastic gradient decent step on $[y_t - Q(s_t, a_t, \theta)]^2$ with respect to $\theta$
    Every $c$ steps set $\tilde{Q} = Q$
  **End For**
**End For**

$\gamma$ is the discount factor and $\theta$ the neural network parameters.

# 4 DQN($\lambda$)

**procedure** REFRESH($l$)
  **For** transition $(s_t, s_{t+1}, r_t, R_t^\lambda, a_t) \in l$ processing back-to-front **Do**
    **If** terminal($s_{t+1}$) **Then**
      Update $R_t^\lambda \leftarrow r_t$
    **Else**
      Get adjacent transition $(s_{t+1}, s_{t+2}, r_{t+1}, R_{t+1}^\lambda, a_{t+1})$ from $l$
      Update $R_t^\lambda \leftarrow r_t + \gamma[\gamma R_{t+1}^\lambda + (1-\lambda)\max_{a'} Q(s_{t+1}, a', \theta)]$
    **End If**
  **End For**
**End procedure**

**For** episode = 1, M **do**
  Initialize a fresh game
  **For** t = 1, T **do**
    Play move $a_t = \begin{cases} \text{random move} & \text{with probability } \epsilon \\ \text{argmax}_a Q(s_t, a, \theta) & \text{for white} \\ \text{argmin}_a Q(s_t, a, \theta) & \text{for black} \end{cases}$
    Receive reward $r_t$
    Append the transition $(s_t, s_{t+1}, r_t, R_t^\lambda, a_t)$ to $L$, where $R_t^\lambda$ is arbitrary
    **If** terminal($s_{t+1}$) **Then**
      REFRESH($L$)
      Store $L$ in D
    **End If**
    Sample a random minibatch transition from $D$
    Perform a stochastic gradient decent step on $[R_t^\lambda - Q(s_t, a_t, \theta)]^2$ with respect to $\theta$
    Every $c$ steps REFRESH($D$)
  **End For**
**End For**

$\gamma$ is the discount factor and $\theta$ the neural network parameters.

# 5 AlphaZero