Notes on the Tic-Tac-Toe Game

Notations. A *configuration* (also called a *state*) of a tic-tac-toe game is defined as a placement of `X`'s and `O`'s on a 3×3 board. An example configuration is shown below[1]. Given two configurations s_1 and s_2, if we can obtain s_2 via one additional move from s_1, we then call s_2 a *successor* of s_1[2].

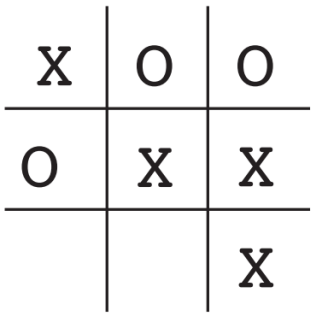

Figure 1: A configuration of a tic-tac-toe game [1]

Initialization. Imagine a scenario where we are playing tic-tac-toe against an opponent. Before the *first* game starts, each configuration is associated with an initially probability of winning for us. Without loss of generality, suppose we always play `X`, then (i) all configurations with three `X`'s in a line have winning probability 1, and (ii) all configurations with three `O`'s in a line have winning probability 0. Further, we initialize the winning probabilities of all other configurations to 0.5. Let $V(s)$ denote our winning probability under a configuration s. As the game proceeds, the winning probabilities of configurations get updated.

Strategy. The game starts at time $t = 1$ from an empty configuration with no `X`'s and `O`'s.[3] Let s_t denote the configuration of the game at time-step $t \geq 1$, and let $\mathcal{S}_t$ denote the set of all successors of s_t. Suppose it is our turn at time t. We make a move, which effectively choose a configuration s_{t+1} from $\mathcal{S}_t$. In particular, the strategy used in the provided source code for choosing a new configuration

[1] As students might notice, many configurations are illegal. Thus, two possible questions for students: *What is the total number of configurations? What is the total number of legal configurations*?

[2] Two possible questions for students: *Can a configuration have more than one successor? Can a configuration be a successor of more than one configuration?*

[3] One possible question for students: *what are the successors of an empty configuration?*

is the following ϵ-*Greedy* strategy:

$$\begin{cases} s_{t+1} \leftarrow \text{a configuration in } \mathcal{S}_t \text{ with the largest winning probability} & w.p.\ 1-\epsilon \\ s_{t+1} \leftarrow \text{a random configuration in } \mathcal{S}_t & w.p.\ \epsilon \end{cases}$$

After choosing a configuration s_{t+1}, we updates our winning probability $V(s_t)$ as follows [1]:

$$V(s_t) = V(s_t) + \alpha(V(s_{t+1}) - V(s_t)) \tag{1}$$

where α is a small positive fraction called a *step-size parameter* that governs the learning rate. Note that the step-size parameter should decrease over time.

A game ends with a configuration where either a player wins, or a draw is yield. If we play multiple games with the opponent, the set of winning probabilities of configurations learned from previous games are carried over to the new game as initial winning probabilities.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.