Aprendizaje de máquina TIC-TAC-TOE

Introducción

El objetivo de este proyecto es diseñar un sistema de aprendizaje que aprenda a jugar **TIC-TAC-TOE**. Para proceder con nuestro diseño primero vamos a especificar el problema de aprendizaje que estamos tratando, escoger la función objetivo que se va a aprender, la representación de dicha función, valores de entrenamiento.

Problema de aprendizaje de TIC-TAC-TOE

- Tarea T : Jugar **TIC-TAC-TOE**.
- Medida de desempeño P : Porcentaje de juegos ganados contra sí mismo.
- Experiencia de entrenamiento E : Jugar contra sí mismo.

Función objetivo

Sea V una función que dado un tablero válido del juego le asigna un valor real y la denotamos por $V : B \rightarrow \mathbb{R}$ donde B es el conjunto de tableros válidos y $\mathbb{R}$ el conjunto de números reales.

Representación de la función objetivo

Para cada tablero dado, la función V va a ser calculada por la combinación lineal

$V(b) = \sum_{i=0}^{6} w_i * x_i$

donde x_0 = 1, $x_i \; \forall i \; 1 \leq i \leq 6$ representa la $i - ésima$ característica del tablero

$x_1 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparece una X y dos casillas vacías.

$x_2 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparece un O y dos casillas vacías.

$x_3 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparecen dos X y una casilla vacía.

$x_4 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparecen dos O y una casilla vacía.

$x_5 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparecen tres X.

$x_6 \rightarrow$ Cantidad de líneas(filas, columnas, diagonales) donde aparecen tres O.

y $w_i \, \forall i \, 0 \leq i \leq 6$ son coeficientes reales que serán escogidos por el algoritmo de aprendizaje.

Valores de entrenamiento

Vamos a definir la función de entrenamiento de la siguiente forma

- Si b es un tablero final donde se ganó entonces $V_{train}(b) = 100$
- Si b es un tablero final donde se empató entonces $V_{train}(b) = 0$
- Si b es un tablero final donde se perdió entonces $V_{train}(b) = -100$
- En otro caso, $V_{train}(b) \leftarrow V_{train}(Sucesor(b))$

donde $Sucesor(b)$ representa el tablero siguiente de realizar un movimiento en el juego.

Ajuste de los pesos(w_i)

El enfoque que vamos a utilizar para actualizar los pesos es minimizar el cuadrado del error entre los valores de entrenamiento y los valores que asigna la función objetivo(LMS).

$E = \sum_{\forall <b,V_{train}(b)> \, \in \, conjunto \ de \ entrenamiento} (V_{train}(b) - V(b))^2$

Regla de ajuste

$\forall < b, V_{train}(b) > \in \ \ conjunto \ de \ entrenamiento$

- Usar los pesos actuales para calcular $V(b)$
- $\forall w_i$, $w_i \leftarrow w_i + \alpha * (V_{train}(b) - V(b)) * x_i$

donde α es una constante pequeña que mide el paso de actualización de los pesos y va a ir disminuyendo a medida que va entrenando el programa.

Implementación

El diseño final del proyecto está compuesto por siete módulos, cuatro de ellos representan los componentes principales del sistema de entrenamiento, mientras que los otros tres representan al juego, el agente que va a aprender y la función objetivo. Primero vamos a explicar el módulo del juego, el agente y mencionar simplemente el módulo de la función objetivo.

1. **TIC-TAC-TOE** $\rightarrow$ Contiene la clase **TIC_TAC_TOE_game**, representa al juego con una matriz de 3x3 en la cual solo pueden haber $1's$ para representar a las X, $0's$ para las casillas vacías y $-1's$ para los O. Esta clase contiene los métodos **get_valid_moves**, que devuelve una lista de tuplas de la forma x, y que indica que la posición x, y de la matriz está vacía, **change_state** cambia la posición x, y de la matriz por algún valor(1 o -1), **game_status** que devuelve un entero para representar la condición actual del juego, $(1, 0, -1)$ para saber si ganó el jugador de las X, si hubo empate o si ganó el jugador de los O respectivamente y 2 para indicar que el juego no ha finalizado, **get_board** devuelve una copia del tablero actual y **print_board** imprime en la consola el tablero actual.
2. **bot** $\rightarrow$ Contiene tres clases que representan distintos tipos de agentes, primero tenemos un agente llamado **RandomBot** que sencillamente todos sus movimientos son aleatorios, luego tenemos a **VBot** que en todas sus jugadas utiliza la función objetivo para evaluar los posibles tableros que puede provocar y toma al que más valor le dio, por último tenemos a

MixedBot un agente que tiene una probabilidad de 0.5 de realizar una jugada aleatoria sino realiza la misma función que la clase vista anteriormente. Este último agente es el que fue usado en el entrenamiento para tener la posibilidad de explorar tablero que si usáramos siempre la función objetivo no íbamos a encontrar.
3. **v_1** $\rightarrow$ Representa con la función **approximate_board_value** la misma función objetivo discutida arriba.

Ahora vamos a pasar a explicar como se implementó el sistema de entrenamiento.

1. **performance_system** $\rightarrow$ Este módulo es el encargado de generar una historia de juego, donde los agente toman los valores actuales de los pesos y juegan una partida, la función **create_game_history** es quien se ocupa de esto y devuelve una lista donde la primera posición son los tableros jugados por el agente de la X y la segunda son los tableros del agente del O. El tablero final cuenta como un tablero jugado por ambos para poder asignar los valores de entrenamiento más tarde.
2. **critic** $\rightarrow$ Contiene a la función **assign_V_train_values** que va a asignar a cada tablero de cada jugador los valores de entrenamiento usando la forma antes descrita.
3. **generalizer** $\rightarrow$ Aquí es donde ocurre la actualización de los pesos, la función **assing_board_approximation** recibe los conjuntos de entrenamiento para aplicar el algoritmo LMS y devolver los nuevos pesos.
4. **main** $\rightarrow$ Es el encargado de conectar los anteriores módulos. Contiene un ciclo **for** para simular los entrenamientos, al ejecutarse se inicializan unos valores aleatorios entre -10 y 10 de los pesos, luego con dichos valores se procede a utilizar el **performance_system**, la salida de este es pasada a **critic** para generar los conjuntos que va a utilizar el **generalizer**, con los pesos actualizados se comienza una nueva iteración. Los valores de los pesos que quedan al final del ciclo son los que el agente aprende luego de una cantidad determinada de entrenamiento. Los últimos valores son escritos en **weights.txt**.

Módulo adicional

El módulo **match** al ejecutase inicia un juego entre una persona y un agente tipo **VBot**. Se elige que jugador se quiere ser y se comienza a jugar hasta que acabe. La forma de realizar un movimiento es introducir x, y separados por espacio, por ejemplo $1\ 1$. Como está el módulo ahora mismo los agentes **VBot**(el que aprendió a jugar X y el que aprendió a jugar O) tiene asignados unos pesos luego de entrenar 500 veces.