Tarea 2

Javier Espinoza
October 11, 2017

CC5114-1 Redes Neuronales y Programación Genética
Departamento de Ciencias de la Computación
Facultad de Ciencias Físicas y Matemáticas

1 Cómo ejecutar

Para ejecutar el script necesario, se debe correr el script *Run.py* que contiene 2 partes. El loop while corresponde a la elaboración/construcción del dataset para la red neuronal. Luego, la tabla o Board aprende de estos dos archivos creados por el bloque previo para poder jugar contra la computadora o un humano.

2 Red Neuronal

La construcción de la red neuronal se da por una layer de inputs, 1 hidden layer y 1 ouput layer. La capa de inputs posee 9 inputs que corresponden a los 9 símbolos que pueden haber en la tabla de gato, con los primeros tres inputs siendo la primera fila, y así para los otros trios. Los outputs corresponden a lo mismo, las posiciones del tablero TicTacToe. Si es que una de los outputs tiene un número bastante grande, entonces significa que hay que colocar un símbolo en esa casilla correspondiente. La hidden layer contiene 12 nodos. La visualización de esta red es:

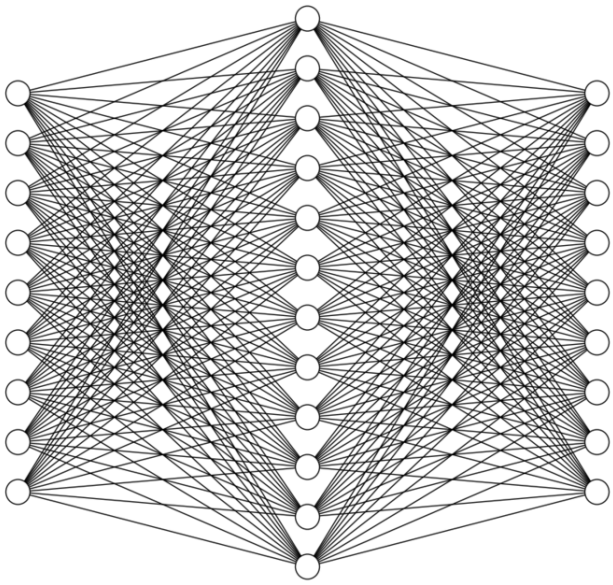

Figure 1: Red neuronal usada

3 Datos para entrenar

Debido a que no existe data de movimientos para ganar en un juego de Tic Tac Toe, se tuvo que generar los datos para entrenar a mano. Es decir, se realizó un script donde 2 computadoras (inputs random) juegan Tic Tac Toe. Mientras se escogen los lugares para poner los símbolos X y O, se tiene guardados en listas los movimientos que hace la computadora X y O, y el estado de la tabla al momento de hacer el movimiento correspondiente.

Si es que una de las computadoras gana el juego, se guardan los movimientos de la computadora ganadora en los archivos *fileX.txt* o *fileO.txt*, dependiendo del símbolo que ganó. Por otro lado, si es que hay un empate, se guardan ambos movimientos en los dos archivos, después de ser normalizados para ser puestos en la red. Un ejemplo del archivo *fileO.txt* es:

1.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
1.0 1.0 1.0 1.0 1.0 0.0 0.5 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 1.0 0.5 1.0 0.0 0.5 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Ejemplo de archivo normalizado

Dentro del archivo, 1.0 representa un espacio en blanco, 0.5 es O y 0.0 representa la X. En este caso, el archivo original es:

0 0 0 0 0 X 0 0 0 6
0 0 0 0 0 X O X 0 3
0 X 0 O 0 X O X 0 0

Ejemplo de archivo no normalizado

La red lee los datos que vienen de *fileO.txt* o *fileX.txt* dependiendo de qué símbolo quiere jugar.

Si bien esta data fue sacada de dos computadoras jugando randommente TicTacToe, es posible hacer que aprenda de inputs humanos reemplazando la parte del código de *Boards.py* donde se obtiene un número cualquiera entre 0 y 8 para ubicar en el tablero. De esta manera, la red aprenderá de inputs humanos y tenderá más a ganar que a hacer un movimiento cualquiera dentro del juego.

4 Resultados

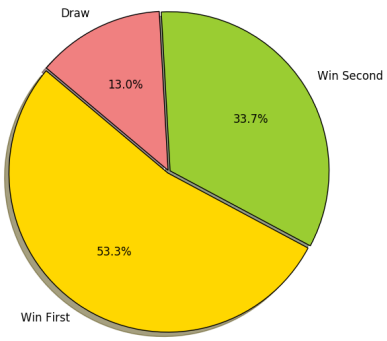

Figure 2: Resultados de jugar TicTacToe con la Red Neuronal cuando la Red Neuronal empieza el juego.

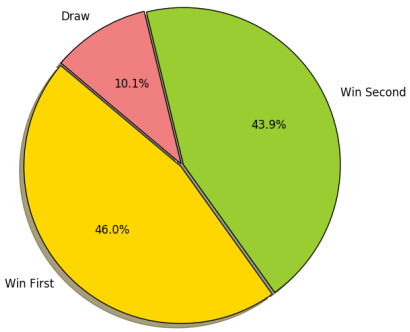

Figure 3: Resultados de jugar TicTacToe con la Red Neuronal cuando la Red Neuronal juega segundo.

En estos resultados, se hizo que la red neuronal jugara contra la computadora, donde ésta última escoge al azar una posición para poner su símbolo. De ahí, la red neuronal obtiene el estado actual de la tabla, usa eso como input, y entrega como resultado la posición donde debería ir su propio símbolo.

Es importante decir que los resultados obtenidos por esta red son de esta manera debido a que el dataset que fue generado es random, en otras palabras, generados por la computadora. Sin embargo, el dataset es posible de extender a inputs humanos, pero requiere de una cantidad mayor de tiempo debido a todas las combinaciones posibles que pueden ocurrir dentro del juego.

5 Tema de Investigación

AlphaGo es un programa informático basado en inteligencia computacional para jugar el juego de mesa Go. Es la primera máquina que derrotó a un jugador de Go sin handicaps en el tablero de 19x19.

El algoritmo de AlphaGo está dado por una combinación de técnicas de Deep Learning y árboles de búsqueda. Utiliza un árbol de búsqueda MonteCarlo, un algoritmo de búsqueda heurístico para la toma de decisiones, especialmente en juegos.

AlphaGo fue entrenada a través de redes neuronales usando la experiencia de un jugador humano, y una base de datos de alrededor de 30 millones de movimientos. Luego era entrenado aún más desempeñando partidos contra instancias de sí mismo, usando aprendizaje por refuerzo.

5.1 Algoritmo de Montecarlo

El enfoque del árbol de búsqueda Monte Carlo se encuentra en el análisis de los movimientos que proveen la mayor probabilidad de ganar, ampliando el árbol de búsqueda basado en un muestreo aleatorio del espacio de búsqueda. La aplicación de búsqueda de árbol de Monte Carlo en los juegos se basa en muchos playoffs. En cada emisión se juega de salida hasta el final mediante la selección de movimientos al azar. El resultado final del juego de cada playout se utiliza para ponderar los nodos en el árbol del juego de manera que los mejores nodos son más propensos a ser elegidos en futuros playoffs.

El modo más básico de usar estos playouts es aplicar el mismo número de playouts después de cada movimiento del jugador real, y luego escogiendo el movimiento que haya obtenido la mayor cantidad de partidas ganadas. La eficiencia de este método incrementa con el tiempo debido a que más playouts son asignados a movimientos que hayan ganado el juego.

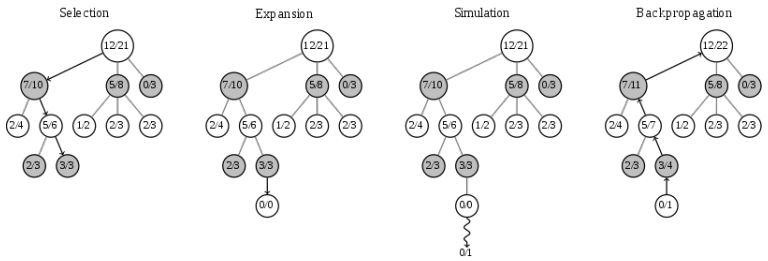

Figure 4: Método de búsqueda en un árbol de Monte Carlo. Existen 4 pasos: Selección, Expansión, Simulación y Propagación hacia Atrás.

- Selección: Se empieza desde la raíz y se selecciona sucesivamente hijos hasta una hoja L.
- Expansión: A menos que L gane o pierda el juego, se crea uno o más nodos hijos y se escoge un nodo C entre ellos.
- Simulación: Se juega un playout random del nodo C. Este proceso se llama rollout.
- Propagación hacia Atrás: Usar la información del playout para actualizar la información del árbol entero hasta la raíz.

5.2 Ventajas y Desventajas

Aunque la evaluación de los movimientos de MonteCarlo convergen a Minimax (algoritmo de decisión para minimizar la causa o probabilidad de pérdida en el peor caso), la versión básica de búsqueda en un árbol de MonteCarlo es muy lenta. Sin embargo, no necesita una función de evaluación. Solamente se deben implementar las reglas del juego (sea Go, TicTacToe, etc) para explorar el espacio de búsqueda. También, el árbol crece de manera asímetrica debido a que el algoritmo busca o se concentra en los subárboles más prometedores.

Una desventaja, sin embargo, es que si el algoritmo se enfrenta con un jugador experto, puede que exista un subárbol o branch que pueda llevar a una partida perdida que no es fácil de obtener pues la búsqueda en un árbol de MonteCarlo no tendrá en cuenta un subárbol 'despreciable'. Se teoriza que esto ocurrió al momento en que AlphaGo fue derrotado por Lee Seedol.

5.3 Links relacionados a AlphaGo

Artículo sobre AlphaGo escrito por DeepMind, lider de la investigación en inteligencia artificial
Artículo sobre el juego entre AlphaGo y Lee Sedol.
Introducción al Algoritmo de búsqueda de Montecarlo.