Inteligencia Artificial

Práctica 4

Pareja 02

Oscar Gomez Borzdynski, José Ignacio Gómez García

3-5-2018

Pregunta C1: Explicación del código entregado

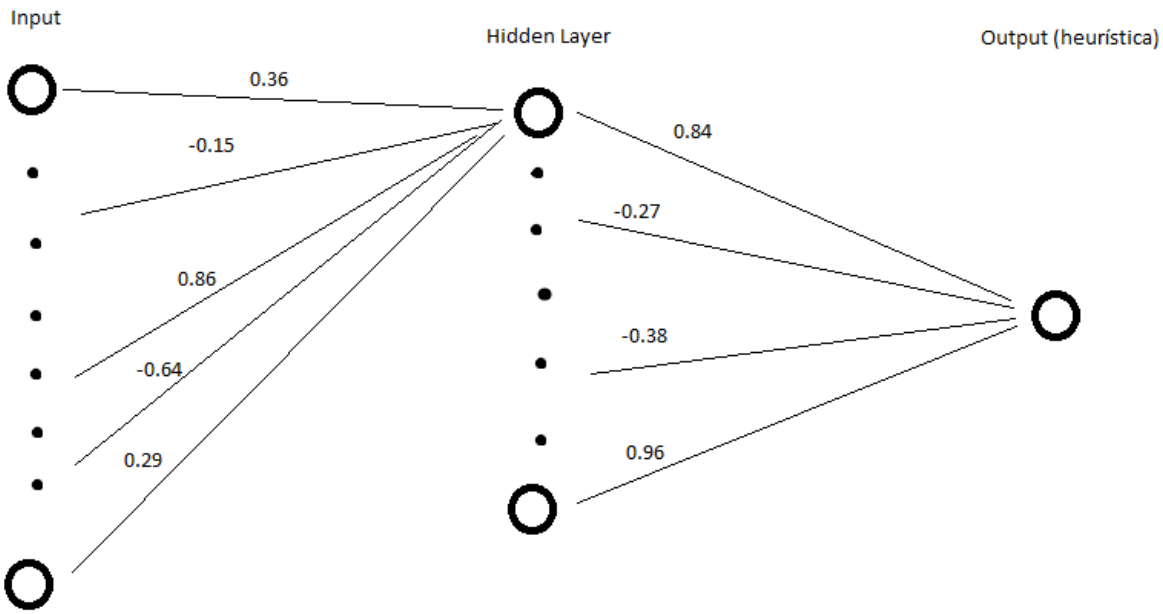
En un primer momento comenzamos pensando heurísticas específicas para el juego. Para ello aprendimos los fundamentos del juego y desarrollamos ciertas estrategias a base de jugar. Entre estas estrategias se encontraban las siguientes:

- Contar el número de fichas que hay en los hoyos
- Contar el número de fichas en los kalahas
- Evaluar las posiciones en las que se podía comer alguna ficha
- Priorizar que la última ficha del movimiento acabase en el kalaha para repetir turno
- Priorizar que nuestra puntuación sea mayor que la del rival

Sin embargo, tras numerosas pruebas, llegamos a la conclusión de que la forma más eficiente de mejorar en el juego era automatizando el proceso empleando un algoritmo genético en colaboración con otra pareja del grupo de Simone Santini (Alejandro Cabana y Aitor Arnaiz). Es importante destacar que, aunque hubo colaboración en el desarrollo del código, cada pareja ha seguido un itinerario distinto a la hora de trabajar con las generaciones y las estrategias de evolución.

Nuestro algoritmo genético (Python + Lisp-SBCL)

Comenzamos elaborando una "red neuronal" en la que se recibían tantos inputs como hoyos había en el tablero, y cuya única salida era un valor entre -1 y 1 que equivalía a la heurística obtenida. También contaba con una *hidden layer* de 10 neuronas que otorgaban pesos aleatorios entre -1 y 1 a cada uno de los inputs de la capa previa. Finalmente, la neurona de salida otorgaba un peso aleatorio a cada una de las neuronas de la *hidden layer*, obteniendo así una ponderación del estado del tablero.



En cuanto a la ingeniería de nuestra red evolutiva, comenzamos con la idea de producir numerosas generaciones (del orden de miles), de pequeño tamaño. Por cada generación tendría lugar un torneo de Mancala en el que se enfrentarían todos contra todos, de forma que sólo los 2 mejores sobrevivieran de cara a la siguiente generación. El resto de la generación sería fruto de combinaciones de los dos supervivientes (intercambiando neuronas y generando algunas nuevas a modo de *mutaciones*). Adicionalmente, incluíamos algunos jugadores totalmente aleatorios que aportaran variabilidad.

Sin embargo, pudimos observar que al hacerlo de esta forma, los jugadores se especializaban en ciertas estrategias, de modo que era muy probable que fueran derrotados por un jugador con una estrategia diferente. Por ello, decidimos aumentar el número de jugadores por generación, e incluso aumentar la variabilidad introduciendo más jugadores aleatorios. Además, decidimos generar series de 150 *jueces* aleatorios, de forma que, en vez de hacer torneos entre los miembros de la nueva generación, éstos se enfrentaran a los jueces.

De esta forma obtuvimos resultados mucho más satisfactorios e, incluso, nos colocamos en cabeza del ranking durante varios días.

Más adelante pensamos en "favorecer" a los jugadores que habían sobrevivido varias generaciones, estableciendo un sistema de porcentajes de victorias que se mantendría entre generaciones. Para evitar que un jugador nuevo obtenga un 100% de victorias nada más ser creado, introdujimos un *hándicap* que sumaba una victoria, una derrota y un empate a los jugadores nuevos, y calculamos que el máximo porcentaje obtenido por un nuevo jugador sería del 95%.
Finalmente, para acelerar la ejecución de nuestro programa, decidimos paralelizar las partidas de cada generación, obteniendo mejores resultados.

Por otro lado, creemos que, como nuestra heurística consiste en una mera operación matricial (producto vectorial), va a ser muy rápida y eficiente computacionalmente.

Pregunta C2:

El jugador bueno sigue la misma estrategia que el regular, pero a un nivel mayor de profundidad. De esta forma, va a elegir la mejor jugada de cara a lo que prevé que pasará dos turnos después. En concreto, el jugador bueno toma el máximo de los resultados, pero no tiene en cuenta los mínimos, por lo que es posible que elija una jugada que le lleve a una situación en la que dependa del movimiento del rival, y éste pueda jugar ese mínimo, perjudicando al bueno.

La solución que proponemos consiste en mirar los mínimos en vez de los máximos, de forma que el jugador bueno tome la jugada con la que menos arriesga, de forma que, aunque el rival juegue la mejor de sus opciones, siga siendo la menos mala para bueno.

```
(defun f-eval-Bueno (estado)
  (if (juego-terminado-p estado)
      -50                                ;; Condicion especial de juego terminado
    ;; Devuelve el maximo del numero de fichas del lado enemigo menos el numero de propias
    (max-list (mapcar #'(lambda(x)
                          (- (suma-fila (estado-tablero x) (lado-contrario (estado-lado-sgte-jugador x)))
                             (suma-fila (estado-tablero x) (estado-lado-sgte-jugador x))))
                      (generar-sucesores estado))
              :test #'Min))
```

Tras probar nuestra solución, hemos comprobado que nuestro bueno gana, tanto moviendo primero como segundo, al regular.