PRÁCTICA 4 INTELIGENCIA ARTIFICIAL: TORNEO DE MANCALA

Grupo 2301, Pareja 03

Alejandro Cabana Suárez y Aitor Arnaiz del Val

PREGUNTA C1

Explique los fundamentos, el razonamiento seguido, la bibliografía utilizada y las pruebas realizadas para la entrega de la/s función/es de evaluación entregada/s en el torneo

En un primer momento intentamos razonar las funciones de evaluación. Pensamos por ejemplo en premiar jugadas que saltaran turno al contrario, pero la función '(estado-debe-pasar-turno estado)' no funciona. Intentamos sustituirla contando el número de hollos con las semillas justas para caer en el kalaha y repetir turno, pero de esta manera premiábamos que se mantuvieran en esa posición, en lugar de que se jugaran. Intentamos algo parecido con las posiciones desde las que podíamos comer semillas del oponente, con el mismo resultado. Luego, probamos heurísticas simples no muy pensadas, como devolver el número de semillas en los tres hollos más cercanos al kalaha (que son más difíciles de comer), o el número de hollos vacíos en nuestro lado (más oportunidades de comer); ninguna funcionó. Curiosamente, nos confundimos al implementar la segunda y en lugar de hollos vacíos contamos hollos con una única semilla. Esta fue la primera que consiguió ganar al jugador regular.

Como veíamos que razonar no servía de mucho y ya habíamos leído algo sobre redes neuronales y algoritmos genéticos, decidimos crear uno para producir funciones de evaluación. Está implementado en python como una red con 14 inputs (las semillas en cada hollo y los dos kalahas), una capa oculta con 10 neuronas, y un solo output, que es el valor de la función de evaluación. Los pesos entre una capa de neuronas y la siguiente están codificados como una matriz de números entre -1 y 1 que multiplicamos por el vector de neuronas de la primera capa, para conseguir el vector de neuronas de la siguiente. Entre la primera capa y la segunda hay una matriz 10x14, y entre la segunda y la última, una 10x1. Después de aplicar los pesos, a cada resultado le sumamos otro número al que llamamos sesgo para obtener, ahora sí, el valor de la neurona siguiente.

Desde python lo único que hacemos es generar todos estos números de acuerdo al algoritmo, imprimir en un fichero el código lisp que multiplica las matrices y suma los vectores, ejecutar el juego y analizar la salida para ver quién es el ganador.

Para que los jugadores vayan mejorando utilizamos un algoritmo genético. En resumen esto significa comenzar con jugadores aleatorios, probarlos todos, y combinar los mejores para producir la siguiente generación de jugadores. Aunque parece simple a primera vista, esta parte ha sido a la más tiempo hemos dedicado.

En primer lugar, para combinar dos jugadores lo que hacemos es pasar por cada uno de los pesos y sesgos que tenemos que generar, y con la misma probabilidad del 47,5% elegir entre escoger el del primero o el del segundo. El 5% restante de las veces lo rellenamos con un número aleatorio, a lo que llamamos mutaciones. Además, en cada generación introducimos un jugador totalmente aleatorio para que el algoritmo no se centre únicamente en una estrategia y pierda capacidad de mejora.

Para decidir qué jugador es mejor que otro hemos pasado por varias fases:

- Primero creamos generaciones de 10 jugadores e hicimos que compitieran todos contra todos. Los dos que más partidas hubieran ganado pasarían a la siguiente generación y se reproducirían. Esta estrategia no funcionó muy bien; los jugadores que obtuvimos sólo eran buenos contra sus "hermanos", pero cualquier jugador aleatorio conseguía ganarles.

- Después decidimos generar otro conjunto de jugadores aleatorios a los que llamamos jueces, y toda la generación competía contra cada uno de ellos. En un principio había 20 jueces, y se renovaban a cada generación. Con esto arreglamos el problema de la fase anterior, pero seguíamos queriendo mejorar el algoritmo.

- Como en cada generación dos de los jugadores eran de la anterior y no era raro que estos fueran de los mejores de la siguiente, pensamos que tenía sentido que conservaran las puntuaciones de las partidas contra los jueces anteriores. Así que las mantuvimos y pasamos a contar el porcentaje de partidas ganadas. El problema con esto fue que nuevos jugadores que tuvieran la suerte de ganar a todos los jueces de su generación les quitaban el puesto a los "veteranos" para después perder más partidas y bajar su porcentaje, con lo que perdíamos a los veteranos que eran buenos por nuevos jugadores que no eran necesariamente mejores.

- Para solucionar el problema, inspirados por el tema de incertidumbre de la clase de teoría, decidimos que cada jugador comenzaría por defecto con una partida ganada, otra perdida y otra empatada. De esta manera, ninguno podría obtener un 100% de victorias en su primer "torneo". El resultado no fue el que esperamos, ahora premiábamos demasiado a los jugadores antiguos, que ocupaban todas las primeras posiciones y no dejaban que el algoritmo evolucionara. Ampliamos a generaciones de 20 jugadores contra 30 jueces, entre los que pasaban de generación los 5 mejores, pero tampoco sirvió.

- El último intento fue algo que nos recomendaron nuestros profesores:

  Por un lado, Alberto nos dijo que necesitábamos generaciones mucho más grandes, así que las ampliamos a 100 jugadores contra 50 jueces. Para que diera tiempo a ejecutar suficientes generaciones en una noche para entregar jugadores cada día, paralelizamos las partidas contra los jueces.

  Por otra parte Santini nos dijo que existía la estrategia del pool. Por cada generación creamos un pool de jugadores en el que aparecen varias veces de acuerdo a su posición en el ranking. Esto lo conseguimos escogiendo una pareja aleatoria e introduciendo al pool el mejor de esa pareja, y repitiendo el proceso 100 veces. Ahora de este pool de nuevo elegimos parejas aleatorias y las combinamos para formar los jugadores de la siguiente generación. En todo caso, los dos mejores jugadores siguen pasando automáticamente a la siguiente generación. Esta es la estrategia que hemos seguido en los últimos días del torneo.

Con respecto a las fuentes, son bastante diversas y no las consultamos durante la realización de la práctica sino bastante antes. Aún así, sin duda la que más ayudo a comprender el funcionamiento de las redes neuronales fue la serie de vídeos sobre el tema del canal de YouTube 3Blue1Brown

PREGUNTA C2

El jugador llamado bueno sólo se distingue del regular en que expande un nivel más. ¿Porqué motivo no es capaz de ganar al regular? Sugiera y pruebe alguna solución que remedie este problema.

Esto puede ocurrir porque el jugador bueno solamente escoge la opción que maximiza la diferencia entre el número del lado de fichas del contrario y el del jugador en el segundo nivel de profundidad, sin tener en cuenta la opción tomada en el primer nivel de profundidad; por otra parte, el jugador regular calcula la diferencia contraria: la resta entre el numero de fichas en el lado del jugador con turno menos las del enemigo, optando por la ganancia a corto plazo para sí mismo, que es lo que más le conviene explorando solo en el primer nivel de profundidad, y no la mínima para el otro, como hace el jugador bueno.

Esto puede ocasionar que haya una muy mala jugada para el jugador contrario en la siguiente profundidad, que es lo que intentamos conseguir con el jugador bueno, la jugada mas incómoda para el jugador contrario, pero puede ser que con esta decisión tengamos que tomar un camino aún menos recomendable para nosotros en nuestro turno, lo cual no nos conviene y es lo que intenta evitar el jugador regular, maximizando sus ganancias a corto plazo.

Una forma de evitar que el jugador bueno no gane al regular debido a esta diferencia en la elección de la heurística seria aunar ambas comprobaciones, minimizando la ganancia a largo plazo del jugador enemigo y maximizando la nuestra al mayor corto plazo posible, con lo que exploraríamos tanto a profundidad 1 como a profundidad 2, tomando la mejor decisión posible hasta el máximo de profundidad 2.

EFICIENCIA

En cuanto a la eficiencia y la rapidez de nuestros jugadores, todos nuestros jugadores son igual de eficientes y la ejecucion de todos ellos tarda lo mismo pues todos realizan el mismo numero de operaciones y las mismas operaciones, una serie de sumas y multiplicaciones de matrices.