Universidad Carlos III de Madrid

Aprendizaje Automático

Grado en Ingeniería Informática

Grupo 83

Práctica 3: Aprendizaje por refuerzo

Autores:
Daniel Medina García
Alejandro rodríguez Salamanca

11 de mayo de 2016

Índice

Introducción

Adquirida ya cierta experiencia con el uso de técnias de Aprendizaje Automático tanto supervisado como no supervisado, esta práctica final completa nuestra formación en estas técnicas con la inclusión del aprendizaje por refuerzo. Continuando la línea de las prácticas anteriores, en esta *Práctica 3* nuestro equipo genera un tercer agente automático para el conocido *Comecocos* o *PacMan* haciendo uso de esta potente técnica, con la intención de batir a los anteriores.

1. Diagrama con los diferentes pasos de la tarea de aprendizaje realizada en la práctica

2. Generación del espacio de estados

La elección del espacio de estados es determinante en el funcionamiento de un agente automático guiado por aprendizaje supervisado. En la búsqueda del mejor espacio de estados, el equipo se centró en reducir el número de posibles estados para acelerar el aprendizaje mientras se fomentaba la generalización. Para ésto hay que seleccionar cuidadosamente las variables de los estados para que sean lo más informados posibles.

Primero, probamos un espacio de estados formado por la dirección en la que se encuentra el fantasma más cercano, y si existen muros alrededor de PacMan, esto es, si el fantasma más cercano se encuentra por encima de PacMan y a la izquierda de éste, y hay un muro justo debajo, el estado sería `North,West,False,False,True,False`. Este espacio de estados nos daba un total de 256 estados (cuatro posibles valores de los dos primeros atributos, y dos posibles valores de los cuatro siguientes). Tras ejecutar un agente con este espacio de estados, nos dimos cuenta de que la información relativa a los muros no aportaba información relevante puesto que en nuestro agente sólo elegimos una acción de entre las legales, y aquellas direcciones en las que se encuentra un muro no están incluídas en dicho conjunto.

Después, probamos a eliminar dicha información sobrante y nos quedamos con un conjunto de estados compuesto tan solo por la dirección en la que se encuentra el fantasma más cercano.

Hicimos competir este espacio de estados diseñado por Daniel contra el explicado a continuación, desarrollado por Alejandro. Como este último estaba convencido de su inmediata derrota, chetó a su agente con recompensas intermedias que finalmente le dieron la victoria. Alejandro negará todo esto y dirá que los que se dopan son los ciclistas (reflendo su carencia de constancia en lo deportivo), aunque los reveladores hechos no hacen sino mostrar la carente deportividad del susodicho participante.

Por último, el conjunto de estados elegido fue el formado por la dirección del fantasma más cercano y la dirección del último movimiento realizado por PacMan. Esto nos da un total de 64 posibles estados. Estos atributos fueron elegidos por dos razones:

- Simplifica mucho el conjunto de estados — Al tener en cuenta únicamente tres atributos, el conjunto de estados queda muy reducido, lo que era uno de los objetivos que tratábamos de cumplir, puesto que un conjunto de estados muy grande puede contener estados que no se lleguen a alcanzar nunca.

- Aporta información necesaria y genérica — Otro de los requisitos que hemos intentado cumplir ha sido obtener un agente genérico que no esté *atado* a ningún mapa, y que sea capaz de desenvolverse con soltura en diversos escenarios. Por ello, los atributos escogidos son independientes del mapa, pero aportan la información que necesita PacMan para determinar hacia qué dirección debe moverse.

3. Generación de la *tabla Q*

Para generar la *tabla Q* hemos hecho uso de la técnica conocida como *online learning*. Se trata de un método mediante el cual los datos se encuentran disponibles en orden secuencial (se generan mientras PacMan está jugando partidas) y son usados para actualizar la mejor predicción hasta el momento de la *tabla Q* dado un estado y una acción.

En cuanto a los refuerzos, hemos decidido dividirlos en tres tipos:

- **PacMan se come un fantasma**. Cuando PacMan se come un fantasma, recibe un refuerzo positivo con un valor de 199, coincidiendo con la diferencia de puntuación entre el estado actual y el próximo estado al que se transiciona.
- **PacMan se acerca al fantasma más cercano**. En este caso, entendemos que PacMan está realizando un movimiento beneficioso, por el cual recibe un refuerzo positivo con valor 10.
- **PacMan se aleja del fantasma más cercano**. Si PacMan está *persiguiendo* a un fantasma y su distancia a éste aumenta, se penaliza el movimiento con un refuerzo negativo con valor -1.

Tras probar diversas maneras de dar refuerzo a nuestro agente, la descrita previamente es la que mejor funcionó y la que se adaptaba mejor cuando variábamos el tipo de mapa. Pese ello, en un principio los refuerzos intermedios no dieron un resultado tan bueno como esperábamos; si se está jugando en un mapa con muros que no permiten a PacMan tomar el camino más directo hacia su presa, el acercarse al fantasma no siempre indica avanzar en la dirección correcta pero siempre genera refuerzo positivo.

Además, se ha usado la técnica ϵ-greedy, por la cual nuestro agente toma la acción devuelta por la política obtenida hasta el momento un $\epsilon\,\%$ de las veces, y $(1-\epsilon)\,\%$ una acción aleatoria, para que el mapa sea explorado y se prueben diversas acciones para un mismo estado, con el objetivo de quedarnos con la que se considere mejor.

4. Construcción del gente automático

Nuestro agente ha sido implementado sobre la clase `PacmanQAgent`, la cual extiende de `QLearningAgent`, por lo que trae cierto comportamiento predefinido para que resulte más sencillo implementar un agente utilizando la técnica de Q-Learning.

Para ello, primero se inicializan los valores de `alpha, epsilon` y `discount` en el constructor de la clase, para ser usados posteriormente en la actualización del valor en la *tabla Q*. Además, se carga la *tabla Q*, que se encuentra en un fichero llamado `qtable.txt`. Esta *tabla Q* es almacenada en un objeto de tipo `Counter`, que se encuentra en `utils.py`. Dicho objeto actúa como un diccionario, en el que las claves serán tuplas estado-acción, y el valor, el correspondiente al que retorne la función Q.

Nuestra política es retornada por el método `getPolicy`. Este método recibe un estado, e itera sobre todas las acciones legales, devolviendo aquella acción cuyo valor en la tabla es el máximo para el estado dado.

Si una partida supera los 800 turnos, ésta acaba, pues entendemos que nuestra tabla no se está actualizando con valores que ayuden a nuestro agente a jugar mejor.

Finalmente, al final de cada turno, el método `update` es llamado. Este método recibe como parámetros el estado actual, el estado al que se transiciona, la acción realizada y el refuerzo obtenido con dicha acción. Aquí es donde se encuentra implementada la función de actualización:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma max_a(s,a)]$$

5. Evaluación

Para nuestro agente, hemos decidido probar dos configuraciones distintas a partir de las cuales hemos generados dos *tablas* Q que nos dispondremos a evaluar.

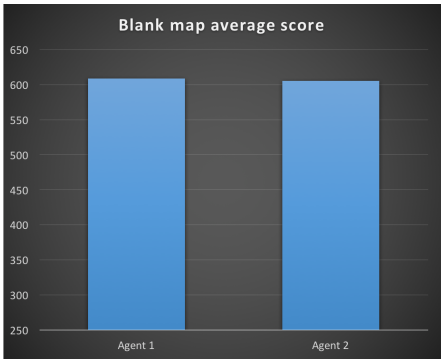
α	0.3
γ	0.8
ϵ	0.8

(a) Agente 1

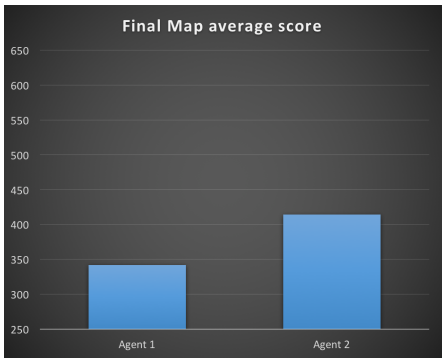
α	0.6
γ	0.4
ϵ	0.7

(b) Agente 2

Para realizar la evaluación de ambas configuraciones hemos jugado 10 partidas en el mapa por defecto y en `finalMap` para comprobar cómo se comportaba el agente en dos escenarios completamente diferentes. Primero, evaluamos los datos obtenidos con los parámetros previamente mostrados, obteniendo los resultados que muestra la figura a continuación.



(a) Default map



(b) Final map

Figura 2: Rendimiento por agentes durante el aprendizaje

Más tarde, utilizaremos siempre la acción que indique nuestra política, sin actualizar los valores de la *tabla* Q durante las distintas partidas para observar las diferencias. La siguiente figura muestra estos resultados para el mapa por defecto.

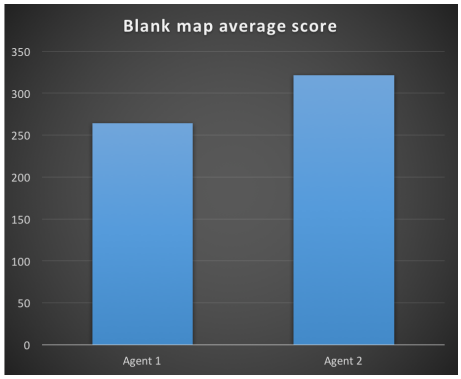


Figura 3: Rendimiento por agentes siguiendo únicamente la política

Pese a la mayor media de puntuación observada aquí en el segundo agente, cabe destacar la ausencia de *timeouts* en el primer agente, mientras que el agente 2 abandonó un 30 % de las partidas jugadas. Un *timeout* se produce cuando la partida sobrepasa los 2000 turnos, habitualmente causado por la entrada en un bucle. También indicar que la ausencia de datos para el *Final map* se debe que todas las partidas acabaron en *timeout* para ambos agentes.

Como podemos observar, en aquellas partidas jugadas con un porcentaje de movimientos aleatorios y en las cuales la *tabla Q* se actualiza, obtenemos unos resultados mucho mejores que cuando seguimos siempre la acción que devuelve nuestra política y mantenemos la *tabla Q* invariable. Esto puede ser debido a:

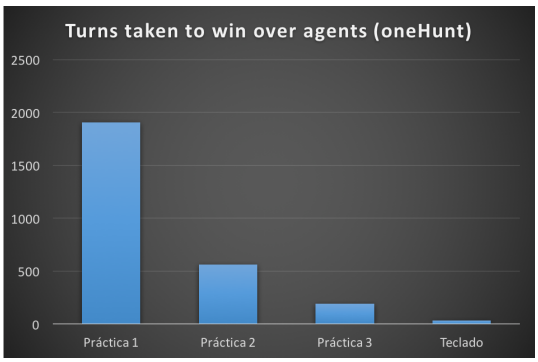
- **Nuestro agente no ha aprendido aún la política óptima** - Pese a haber jugado un elevado número de partidas con nuestro agente para conseguir una *tabla Q* fiable cuyos valores se correspondiesen con la acción más adecuada para cada instante, no hemos conseguido que el algoritmo obtenga los valores adecuados, o la forma en la que expresamos los estados no es la más adecuada.
- **La definición de estado que usamos es muy general** - Al intentar usar una definición de estado lo más genérica posible y haber entrenado en diversos mapas, existe la posibilidad de que nuestro agente no sepa desenvolverse con soltura en ninguno.

Tras valorar los resultados, hemos llegado a la conclusión de que la gran diferencia obtenida es debido al segundo caso, puesto que cuando el agente realiza ciertas acciones aleatorias que le ayudan a explorar un mapa y salir de aquellas situaciones en las que no tiene claro qué debe hacer, y es capaz de actualizar la *tabla Q* para `recordar` esas decisiones en futuras partidas, es capaz de obtener unas puntuaciones muy buenas en las partidas jugadas, y más teniendo en cuenta los datos con los que cuenta para tomar las decisiones.

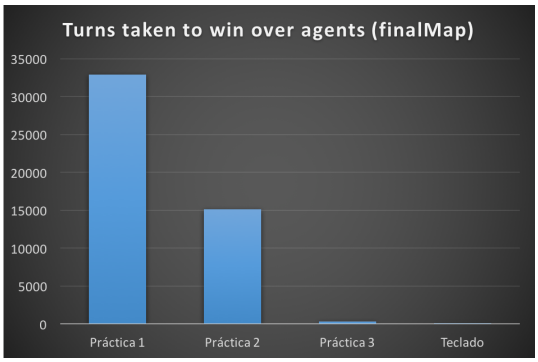
Dada la inestabilidad del segundo agente y el parecido rendimiento de ambos cuando los dos funcionan con normalidad, elegimos el primer agente como el final para nuestro equipo.

6. Análisis del agente final

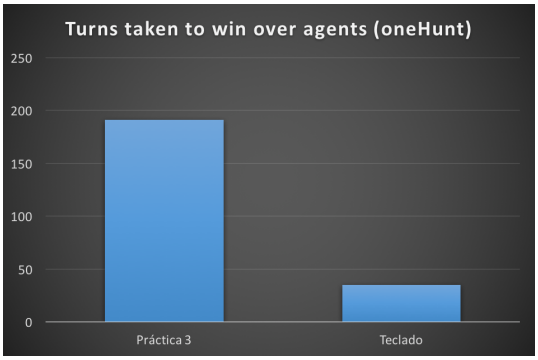
Para evaluar el agente seleccionado, lo comparamos con los generados con otras técnicas en las diferentes prácticas anteriores. De esta forma, jugamos con cada uno de los agentes para recopilar los datos necesarios para comparar los distintos rendimientos. La siguiente figura muestra los resultados obtenidos:



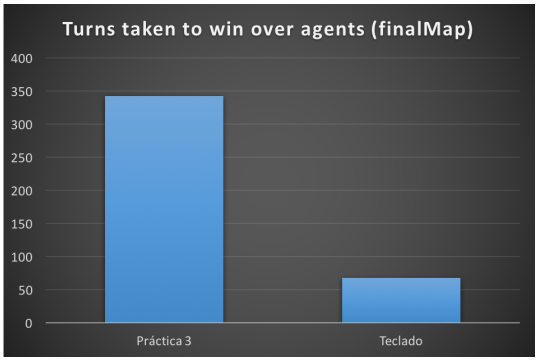
(a) Mapa por defecto - todos los agentes



(b) Final map - todos los agentes



(c) Mapa por defecto - P3 vs. óptimo



(d) Final map - P3 vs. óptimo

Figura 4: Comparativa entre los distintos agentes

Los resultados son demoledores. El nuevo agente está muy lejos del rendimiento obtenido por los agentes de las anteriores prácticas, situándose mucho más cerca del agente humano incluso disponiendo de menos datos. Podemos demostrar así que el agente cumple su función habiendo sido capaz, sin conocer la posición exacta de los fantasmas como hace el agente humano, lograr un rendimiento que se acerca bastante al que conseguiría una persona jugando con el teclado. De esta forma, hacemos una evaluación positiva tanto del espacio de estados como del algoritmo usado por el agente para tomar las decisiones.

El principal problema de este agente es la ausencia de memoria. La carencia de información global sobre su posición causa que PacMan pueda quedar atrapado en el laberinto cuando se requiera backtracking para salir de una calle sin salida, siendo imposible que salga por sí solo y teniendo que recurrir a acciones aleatorias para escapar de bucles en ocasiones. La implementación de ésta, así como un mayor entrenamiento del agente, podrían optimizar el comportamiento del agente.

7. Conclusiones

Esta última práctica ha potenciado bastante nuestra "fe" en el aprendizaje automático. Con el aprendizaje por refuerzo hemos visto que es posible crear un agente viable para ser usado como contrincante para un humano haciendo tan solo uso de una de las técnicas utilizadas, por lo que tenemos motivos para creer que la combinación de varias de ellas podría dar resultados aún mejores.

Problemas encontrados

La dificultad de esta *Práctica 3* ha residido no tanto en la propia implementación como las prácticas anteriores sino en el razonamiento y la toma de decisiones en cuanto al diseño del agente, tanto por el espacio de estados, como por el sistema de aprendizaje (recompensas y parámetros de aprendizaje) empleados.

Comentarios personales, opinión sobre la práctica

A través de las distintas prácticas nos hemos empapado de las distintas posibilidades que ofrece el aprendizaje automático. Hemos observado que, para el problema concreto de PacMan, el aprendizaje supervisado es la técnica más exitosa, lo que nos causa curiosidad por saber en qué entornos el aprendizaje supervisado y no supervisado puede ser más aplicable.

En nuestra opinión, la orientación de la práctica es muy interesante y tan sólo ponemos como aspecto a mejorar la inexperiencia de los profesores con el entorno de trabajo en el que se desarrollaron debido a las novedades introducidas en este año. Creemos y esperamos que los alumnos de próximos años podrán desarrollar con mayor facilidad sus agentes y que éstos tendrán mejores resultados al facilitar esa primera fase de toma de contacto, investigación y prueba y error con las plataformas utilizadas.