Grado en Ingeniería Informática
3º Curso
Grupo 83

José Luis Pérez Torres (100383385)
Miguel Ángel Torrijos González (100383559)

PRACTICA 2:
Aprendizaje por Refuerzo

Memoria de la práctica

uc3m | Universidad Carlos III de Madrid

Índice

Introducción

Esta memoria se corresponde con la práctica 2 de la asignatura Aprendizaje automático de la Universidad Carlos III de Madrid. Este documento está relacionado con el programa Pac-man proporcionado por los profesores. En este documento se detallarán los procesos seguidos y los resultados obtenidos a la hora crear un agente mediante aprendizaje por refuerzo.

Fase 1: Selección de la información del estado y función de refuerzo

Para realizar esta primera fase, ha sido necesario evaluar la situación del problema inicial del Pac-Man que teníamos en el tutorial 1 y entender los conocimientos del aprendizaje por refuerzo vistos en la asignatura. Así pues, lo primero que hacemos es tener en cuenta que para el aprendizaje del agente se van a emplear tuplas de experiencia siguiendo la siguiente forma:

(estado_tick, acción, estado_tick_siguiente, refuerzo)

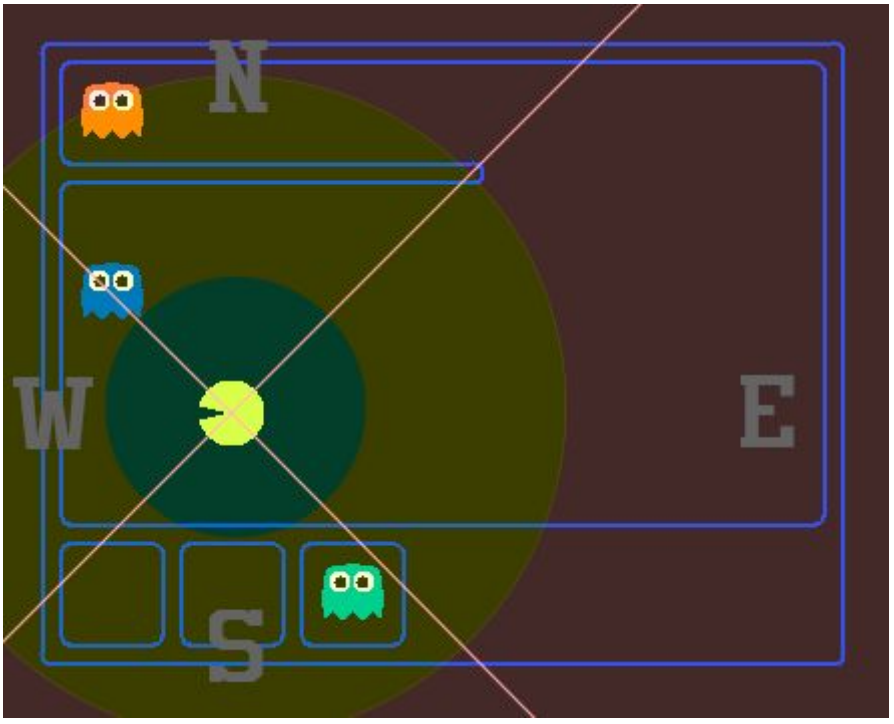
En esta tupla, estado_tick contiene toda la información del estado en un tick determinado, la acción es aquella que el agente ha tomado en ese tick. El estado_siguiente_tick es el estado al que ha llegado el agente después de tomar la acción definida anteriormente y por último, el refuerzo representa el valor devuelto al agente dependiendo de la acción que haya escogido al hacer la transición de un estado a otro y que le indicará si esa decisión fue correcta o no.

El objetivo que tenemos es generar un MDP y para ello necesitamos definir los estados y la función de refuerzo. Comenzaremos explicando el estado definido y sus componentes. Nuestro estado cuenta con 6 elementos. El primero de ellos es la distancia al fantasma más cercano. Debido a que usando valores enteros acabariamos con muchas filas para la q-table, decidimos representar la distancia de forma cualitativa por lo que así, este atributo puede tomar tres valores: cerca, medio o lejos representados en el código como 0, 1 y 2 para mayor comodidad. Ahora vamos a comentar los siguientes 4 elementos añadidos que ya habíamos usado en la práctica 1. Estos son las variables “IsPared” que representan si el pacman tiene una pared en una dirección concreta. Por esto, estos elementos pueden tomar dos valores, o 0 o 1 si no hay o hay una pared. El último atributo es la dirección en la que se encuentra el fantasma más cercano o, si este se encuentra en una zona del mapa con una sola entrada a la que llamaremos “hueco”, la dirección al “hueco” más cercano. Este atributo, por lo tanto tendrá 4 valores; “Norte”, “Este”, “Sur” y “Oeste.

Juntando estos 6 elementos podemos calcular la cantidad de filas que tendrá nuestra q-table usando el siguiente cálculo:

$$3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 4 = 192$$

Por lo tanto, tendremos definidos 192 estados distintos. Creemos que el cambio que hemos hecho para que el primer elemento solo tenga 3 elementos en vez de un número indefinido de ellos ha sido vital para conseguir ajustarnos a la limitación de 200 filas en la q-table. Esto es debido a que si hubiésemos colocado la distancia como un entero tendríamos tantos valores para ese atributo como la distancia máxima de un mapa fuera, cosa que no podemos definir sin poner límites a la q-table. Ahora vamos a poner un ejemplo de una ejecución y vamos a ver cuales son los valores que toma estado en este caso.



Lo primero que comentaremos será la distancia al fantasma más cercano que como se puede apreciar sería el azul. Como podemos ver por las zonas marcadas de forma radial, este fantasma se encuentra en una distancia media. Lo siguiente sería los atributas que comprueban si el pacman está en contacto con alguna pared. En este caso no se cumple por lo que los 4 atributos tomarían el valor 0. Por último, la dirección en la que se encuentra el fantasma. En este caso concreto, el fantasma está justo en la línea que divide la sección norte y la oeste, pero tal y como funciona nuestro código se le asignará la dirección oeste debido a que el eje X tiene más prioridad que el eje Y. Este ejemplo también nos vale para enseñar lo que consideramos “hueco”. En el caso del fantasma naranja está metido un un pasillo, pues nosotros consideramos “hueco” a la entrada a dicho pasillo. De esta forma

evitamos que el pacman se pegue al muro de debajo del fantasma naranja y se quede ahí de forma infinita.

Una vez definidos los estados se va a pasar a detallar la función de refuerzo. La función de refuerzo empleada es la función de actualización no determinista del algoritmo Q-Learning la cual ya se empleo en el tutorial 4.

- Función de actualización no determinista (caso general)

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$$

Esta función creemos que es idónea para resolver problemas de aprendizaje por refuerzo.

En cuanto al refuerzo que recibe el pacman en cada tick de ejecución se ha decidido diseñarlo de tal forma que se priorice cómo de cerca se encuentra el pacman de un fantasma. Para detallar las diferentes posibilidades en el valor de refuerzo se va a tomar el caso de que el fantasma más cercano se encuentra cerca según el primer atributo, está hacia el norte y por lo tanto el pacman debe ir hacia allá.

En primer lugar si el fantasma más cercano se encuentra cerca los refuerzos que se van a obtener van a ser altos en caso de que la acción ejecutada sea la idónea y muy negativos en caso de que la acción ejecutada sea errónea para así penalizar esa acción.

Si el fantasma o hueco más cercano se encuentra en el norte y el pacman ejecuta la acción “Norte” y no hay ningún muro en esa dirección el refuerzo que se obtendrá será de 50, el máximo posible. En caso de ejecutar otra acción que no sea norte se obtendrá un refuerzo de -50. Otro posible caso es que el hueco o fantasma más cercano se encuentre al norte pero entre el pacman y el objetivo haya un muro. En este caso el refuerzo si se ejecuta la acción “Norte” será de -15 ya que no va del todo encaminado pese a haber un muro en el camino. Si en esta situación se ejecutan las acciones “Este” u “Oeste” el refuerzo será de 15 ya que ir en una de esas dos direcciones parece la mejor solución para sortear un muro que hay en el norte. Por último si se ejecuta la acción “Sur” el refuerzo será -50 para penalizar que se está yendo en dirección contraria.Esto se generaliza para todas las direcciones

Si el valor del primer atributo es 1 (medio) los refuerzos varían siendo 25 la máxima recompensa, -25 la máxima penalización y 10 y -10 las recompensas y penalizaciones medias. En el caso de que el primer atributo sea 2 (lejos) los

refuerzos varían siendo 15 la máxima recompensa, -15 la máxima penalización y 5 y -5 las recompensas y penalizaciones medias. Así conseguimos que se recompense o se penalice en función de la cercanía al objetivo.

Fase 2: Construcción del agente

Para esta segunda fase tendremos que construir nuestro agente en la clase BustersAgents. A este nuevo agente le he llamado QLearningAgent igual que en el tutorial 4 ya que vamos a usar algunas funciones que ya venían desarrolladas. Este agente lo hemos ido construyendo de forma incremental según hemos ido haciendo pruebas y comprobando su funcionamiento en los mapas. En primer lugar, implementamos la funcionalidad del agente del tutorial 4 en nuestra práctica para poder trabajar con la Q-table siguiendo el algoritmo Q-Learning. En la nueva clase QLearning Agents se mantienen los mismas funciones que ya poseían otros agentes como, choose action, getAction, init, etc. Además de estas funciones tambien agregamos las necesarias para poder usar la Q-table con nuestro agente, writeQtable, readQtable, getQvalue, etc. Estas funciones las hemos obtenido del tutorial 4. Por último, vamos a comentar las funciones que hemos desarrollado para completar el agente.

- **generateState**: Como su nombre indica, esta función es la encargada de crear el estado actual del pacman con los atributos definidos anteriormente. Para ello obtiene la distancia al fantasma más cercano y la pasa por varias comprobaciones para transformarla a una variable cualitativa. Después obtiene las acciones legales del Pacman para comprobar si tiene paredes alrededor y por último, obtiene la posición relativa del fantasma o hueco más cercano y calcula en q direccion se encuentra.
- **goodActions:** Esta función es auxiliar para la generación del estado. Dada una posición del pacman y una posición objetivo, la función devuelve una lista de las acciones recomendadas que debe realizar el pacman para llegar al objetivo.
- **buscaHueco:** Esta función se encarga de buscar huecos por los que el pacman pueda sortear los muros. Dada una posición y una acción a ejecutar se evalúa que en la componente x o y a la que se va a llegar con dicha acción existe un hueco por el que el pacman pueda pasar.
- **getReward**: Esta función es la encargada de otorgar una recompensa segun el estado y la acción que obtiene por parámetro. Los valores de recompensa que devuelve esta explicados más en detalle en la fase 1 de este documento.

A la mayoría de métodos del tutorial 4 se les llama desde la función choose Action, donde también se realiza la actualización de la tabla Q. El método

readQtable se realiza en el init de la clase QLearningAgent y el writeQtable se realiza al final de cada partida en la función runGames de busters.py.

Durante la fase de aprendizaje de nuestro agente hemos ido variando los valores de α, ε y γ para encontrar los que nos ofrezcan unos resultados mejores. Al final, hemos usado 0.3 , 0.4 y 0.75 para estas variables ya que después de varias pruebas nos otorgaban los mejores resultados en la mayor cantidad de mapas. En nuestro proceso de aprendizaje hemos dado más importancia a los 5 mapas aportados por los profesores de esta asignatura, pero también hemos entrenado a nuestro Pacman en otros mapas que ya teníamos de anteriores prácticas o tutoriales y algunos creados por nosotros. Debido a la variedad de mapas en los que hemos puesto nuestro agente a practicar, creemos que hemos llegado a tener una q table bastante robusta que se puede adaptar a varias situaciones. Como extra, para que nuestro agente no se estropease con una sola ejecución mala en algún mapa, las primeras ejecuciones usábamos un alpha muya reducido para que los posibles errores no destacaron mucho y según el pacman iba mejorando subíamos esta variable para que los datos más relevantes fueran los positivos.

A continuación vamos a pasar a comentar el proceso de creación del estado y los pasos por lo que hemos pasado hasta llegar a nuestro estado final detallado en la fase 1 de este documento.

Nuestra primera idea de estado fue una simple para comenzar con algo facil. En esta solo almacenamos la distancia al fantasma más cercano y la dirección a este. En esta primera prueba nos dimos cuenta rápido de que no era muy óptima ya que la distancia la estábamos guardando como un número entero asiq el tamaño de la tabla-Q dependería de la distancia máxima que pudiese haber entre un fantasma y el pacman. Por esto, esta primera idea fue descartada.

Para solucionar el problema que nos apareció en la anterior prueba, decidimos cambiar la distancia al fantasma más cercano a una variable con una cantidad de valores estables. Para esto hicimos que la distancia pudiese tomar los valores cerca, media y lejos. De esta forma, la tabla-Q tendrá un número de estados finito y estable. En este caso, nos quedamos con (3·4) **12 estados**. Como este número es bastante pequeño en la siguiente idea intentamos añadir más atributos para hacer el estado más completo.

En esta tercera prueba decidimos añadir las variables isPared para las 4 direcciones. Uniendo esto pensamos que ayuda a dar mejores refuerzos ya que tienes un mayor control sobre en qué situación exacta se encuentra el Pacman. usando estos atributos llegamos a tener (3·2·2·2·2·4) **192 estados** lo que

consideramos un número mucho más completo que el anterior y que da más consistencia al agente.

Por último, realizamos un último cambio ya que, en este punto nuestro agente podía pasarse los dos primeros mapas sin problemas pero no conseguía avanzar en los mapas con muros. Por esto decidimos cambiar el atributo que representa la dirección en la que se encuentra el fantasma más cercano para que si el fantasma se encontraba encerrado en una sala con un solo hueco de entrada, este atributo devolviese la dirección al hueco y no al fantasma. Esto nos ayudó mucho ya que de esta forma el agente puede entrar en la sala donde se encuentra el fantasma y después centrarse en ir a por el. Como no hemos cambiado los valores de los atributos ni añadido ninguno nuevo en este último caso seguimos teniendo **192 estados**.

Este conjunto final de atributos es el estado final con el que decidimos quedarnos y que describimos más detalladamente en la fase 1.

Fase 3: Evaluación del agente

En este apartado se va a evaluar el agente desarrollado en los 5 mapas de prueba proporcionados y se le va a comparar con los dos anteriores realizados en el curso. Para obtener la puntuación final media y la media de ticks por ejecución se van a realizar 5 partidas seguidas.

Todas las pruebas se han realizado con los valores α = 0.4, ε = 0.3 y γ = 0.75

Laberinto: labAA1.lay -k 1

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	1	1	1
Puntuación final media	183	46	174.2
Media de ticks por ejecución	17	165	25.4

Podemos observar que QLearningAgent resulve el primer laberinto sin problemas por lo que pasamos a testearlo en el siguiente laberinto.

Laberinto: labAA2.lay -k 2

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	2	No lo resuelve en menos de 2000 ticks	2
Puntuación final media	383		374.75
Media de ticks por ejecución	17		25.8

De nuevo los resultados obtenidos en el agente desarrollado por aprendizaje por refuerzo son muy buenos por lo que se considera este laberinto cómo superado.

Laberinto: labAA3.lay -k 3

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	3	No lo resuelve en menos de 2000 ticks	3
Puntuación final media	569		561
Media de ticks por ejecución	31		40.2

En este laberinto ya se añaden muros que impiden el fácil acceso a los fantasmas. En las primeras aproximaciones de nuestro agente este laberinto era imposible de resolver generandose un bucle infinito cuando pacman intentaba comerse al último fantasma. Con la consideración de los huecos en el estado obtenemos resultados de finalización muy buenos por lo que consideramos este laberinto como superado.

Laberinto: labAA4.lay -k 3

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	3	No lo resuelve en menos de 2000 ticks	3
Puntuación final media	566		657
Media de ticks por ejecución	35		140

Para este laberinto la ejecución empeora debido a la dificultad del mismo. Aún así consideramos que se obtienen buenos resultados ya que el laberinto se resuelve en menos de 150 ticks de media y no se perciben grandes bucles y estancamientos en la ejecución por lo tanto, se considera este mapa cómo superado.

Laberinto: labAA5.lay -k 3

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	No lo resuelve en menos de 2000 ticks	No lo resuelve en menos de 2000 ticks	3
Puntuación final media			714.8
Media de ticks por ejecución			295.2

Este es el último laberinto de prueba proporcionado por los profesores. El pacman en ocasiones encuentra dificultades para salir del primer obstáculo pero lo acaba superando. Consideramos que pese a que los resultados sean peores

respecto a mapas anteriores se mantiene la buena tendencia de resultados por lo que consideramos este laberinto como superado.

Testeado y entrenado el pacman en los 5 laberintos de prueba se va a proceder a probar el agente en otros mapas.

Laberinto: oneHunt.lay -k 4

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	4	No lo resuelve en menos de 2000 ticks	4
Puntuación final media	775		758.3
Media de ticks por ejecución	25		39.8

Laberinto: openHunt.lay -k 3

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	3	No lo resuelve en menos de 2000 ticks	3
Puntuación final media	573		559.8
Media de ticks por ejecución	27		60.2

Laberinto: smallHunt.lay -k 4

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	4	No lo resuelve en menos de 2000 ticks	4
Puntuación final media	769		732.5
Media de ticks por ejecución	31		75

Laberinto: bigHunt.lay -k 4

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	4	No lo resuelve en menos de 2000 ticks	4
Media de ticks por ejecución	139		751.8

En este mapa en ocasiones no termina de superarlo porque se atasca con el último fantasma, para calcular la media se han considerado ejecuciones que superen el laberinto.

Laberinto: trickyClassic.lay -k 4

Resultados/Agente	BasicAgentAA (Tutorial 1 mejorado)	WekaAgent (Práctica 1)	QLearningAgent (Práctica 2)
Número de fantasmas comidos	No lo resuelve en menos de 2000 ticks	No lo resuelve en menos de 2000 ticks	Error de ejecución
Media de ticks por ejecución			

El pacman llega correctamente a la fila de los 4 fantasmas pero se ejecuta un stop que no se debería producir y se produce una excepción. El equipo no ha sido capaz de resolverlo.

Análisis del rendimiento

Con todas las pruebas realizadas vamos a realizar un análisis del rendimiento y vamos a estudiar que se podría mejorar para obtener mejores resultados.

En líneas generales el agente funciona correctamente ya que supera los 5 mapas propuestos en el enunciado. De todas formas, aún se podría mejorar mucho más sobre todo el tratamiento y la decisión de acciones frente a los muros reconsiderando la función de refuerzo o los estados.

Conclusión

Esta práctica nos ha servido de gran ayuda para afianzar nuestros conocimientos en la asignatura de Aprendizaje por Refuerzo y más concretamente en el temario de aprendizaje por refuerzo y la creación de un agente que use el metodo Q-Learning. Esto se suma al hecho de que creemos que aprender mediante algo tan visual como un pacman que va aprendiendo a jugar es un buen método para estimular el aprendizaje. En nuestro caso teníamos algunas dudas sobre algunos conceptos conceptos puntuales de la práctica pero haciendo pruebas hemos entendido mejor estos conceptos.

Con respecto a las dificultades encontradas en esta práctica ha estado englobadas en torno a la implementación de la clase QLearningAgent en bustersAgents.py. Hemos tenido alguno problemas como por ejemplo que el método getAction no podía llamarse así ya que creaba un conflicto. Errores como este hemos tenido algunos pocos y nos ha llevado relativamente bastante tiempo arreglarlos para lo que realmente era ya que, en muchas ocasiones, como el ejemplo antes mencionado, los errores que aparecen en la consola no eran los que exactamente estaban dando problemas, por lo que solucionar dichos errores fue una tarea de prueba y error. Creemos que vendría bien un pequeño apartado en el enunciado con problemas recurrentes para que los alumnos no tengan que andar dando muchas vueltas en un error que no es importante en la práctica. En general creemos que la práctica está bien planteada.