Inteligencia Artificial

Agustina Disiot - 221025
Santiago Diaz - 240926

Q-Learning

Estimación de la estrategia óptima pi* = arg max Q en TD

Control de TD (temporal difference) fuera de política. Encuentra la política óptima (greedy) mientras mejora siguiendo una política épsilon greedy. Los métodos fuera de la política evalúan o mejoran una política diferente de la utilizada para generar los datos.

Método de diferencias temporales 0 porque tiene un look ahead de 1

Off policy: método de mejora de la policy, para poder mejorarla hay que salirse de la policy para poder descubrir mejoras.

Control: método de estimación de la policy óptima

Lo primero que tenemos que hacer es convertir el espacio continuo en uno discreto, para eso usamos bins.

Las observaciones se componen de cuatro cosas: la posición del cartel, la velocidad, el ángulo del pole y la velocidad 'at tip' del pole. Utilizaremos estos para definir los bins.

Las acciones posibles son solo 1 o 0, derecha o izquierda respectivamente.

Creamos una función get state que pasándole la observación devuelve el estado actual discreto, que permitirá encontrar la acción según la policy. Misma idea que la presentada en clase con linspace, donde se utiliza el rango de cada atributo de la observación para obtener el estado. Luego Q se inicializa con estos bins en valores arbitrarios.

Epsilon Greedy Policy

Crea una política (policy) avara (greedy) de épsilon basada en una función Q dada y épsilon.

Para el valor de epsilon utilizamos una función llamada exploration rate. Esto nos permite que a comienzos del episodio se explore con más probabilidad.

Se devuelve una función que toma el estado como entrada y devuelve las probabilidades para cada acción en forma de matriz de longitud del espacio de acción (conjunto de acciones posibles).

Para el valor de alpha también utilizamos una función learning rate o tasa de actualización

Tener estos por separado permite intercambiar dinámicamente su valor a medida que transcurre un episodio: existe un trade off entre explore y exploit donde debemos utilizar un epsilon mas grande al principio para poder explorar más y luego utilizar más el recurso de explotar lo conocido y explorar cada vez menos. No podemos no explorar ya que quedaría en una policy subóptima.

gamma es el descuento, alfa tasa de actualización o learning rate, epsilon greedy → trade off entre explorar y explotar. Si siempre uso exploit entonces no es off policy, siempre juego con la policy sub óptima. De vez en cuando debe explorar para ver si hay estados que no conoce que podrían mejorar su valor.

El factor de descuento, alpha y epsilon tienen un default value en Q-Learning

El valor de gamma, es decir el factor de descuento no puede ser tan chico ya que si es asi, la funcion tiende demasiado a greedy, no permitiendo hallar la policy mas optima, ya que se basaria demasiado en las recompensas de la acción, mientras que si es muy grande puede tener problemas de convergencia.

- Función de valor de acción
- Un diccionario anidado que mapea estado -> (acción -> valor-acción)

Se crea la policy epsilon greedy apropiada para el espacio de acción

Luego para cada uno de los episodios:
- se resetea el estado a el reset del ambiente y elige la primera acción
- se loopea la iteración de obtener la probabilidad de todas las acciones del estado actual
- se elige la acción según la distribución de probabilidad, en este caso uniforme, utilizando epsilon dinámico
- se toma la acción y se obtiene la recompensa
- transiciona al siguiente estado
- se actualizan las estadísticas antes mencionadas: largo de episodio y recompensas acumuladas
- se actualiza el valor de Q usando Q anterior, gamma, y las recompensas nuevas
- si termina el episodio se acaba (se usa el done de step para confirmar esto)

Diferencia temporal para actualizar el valor de Q en un par de estado-acción
Para esto se inicializa Q en estados arbitrarios de tamaño de los bins y además las posibles acciones.

Entrenamiento:
- Discretizar el estado en bins
- explotar la acción de la policy
- explorar acción random
- incrementar el ambiente
- actualizar Q

Para la simulación: Considerando que ya se entrenó el algoritmo de Q learning ahora podemos utilizar directamente la policy óptima, en lugar de greedy que es la que se utilizaba al momento de entrenar y explorar.

Expectimax vs Minimax

Decidimos implementar ambos agentes para nuestro trabajo obligatorio. Esto nos permite probar las suposiciones que teníamos sobre el uso de cada una y mostrar el ratio de partidas ganadas con cada agente.

Utilizamos la misma función de evaluación para ambos agentes.
La función de evaluación o heurística que programamos es una mezcla de las tres heurísticas que se recomendaban

Smooth:
- Calcular el \"smoothness\" del tablero. Para ello debemos:
    - Aplicar la raíz cuadrada al tablero
    - Sumar la diferencia entre cada casilla y la de abajo
    - Sumar la diferencia entre cada casilla y la de la derecha
    - Elevar este resultado a un smoothness_weight a determinar
    - Multiplicar por -1

- valorT (valor tablero): Calcular el valor del tablero. Para ello debemos
    - Elevar el tablero al cuadrado
    - Sumar todos los valores que se encuentran en el tablero

Empty: Calcular la cantidad de espacios vacíos. Para ello debemos:
    - Obtener la cantidad de celdas vacías
    - Multiplicar por un empty_weight (recomendable en el orden de las decenas de miles)

Heurísticas Defensivas:
Smoothness: Esto es porque cuanto más "smooth" el tablero, más fácil es juntar fichas. No quiero tener fichas de bajo valor al lado de fichas de alto valor, ya que estas no me permitirán el acceso cuando quiera juntarlas en una sola casilla.

Heurísticas Ofensivas:
Valor del Tablero: Esto es porque cuanto más fichas grandes tengo, más cerca de ganar estoy.

Vacíos: Esto es porque cuanto más espacios vacíos tenes, menos chance de tener un mal estado.

Utilizamos profundidad 3 para los árboles, que es como un look ahead de 3 niveles (2 jugadas de player y 1 de board). Además el play comienza con una jugada de board, por lo que podría implementarse con una profundidad de 4 niveles y que en play solo se decidiera el movimiento a ejecutar con la heurística.

Una profundidad mayor aumenta considerablemente el tiempo exponencialmente de una partida. En este momento una partida suele durar entre 11-14 minutos. Con una profundidad

extra demora 1 hora 45 minutos. Con una profundidad menor no gana la suficiente cantidad de veces que consideramos apropiada.

Estadísticas:

Runs individuales:

Minimax: Podemos ver que demora 11:49 minutos en llegar a una celda 2048 en 986 movimientos

Expectimax: Podemos ver que demora 15:12 minutos en llegar a una celda 2048 en 981 movimientos

Viendo solamente una partida de cada uno no es suficiente para darnos cuenta cual es mejor, ya que si viéramos solo estos, podríamos concluir que expectimax es peor porque demora mas y ambos agentes consiguieron ganar en aproximadamente la misma cantidad de jugadas.

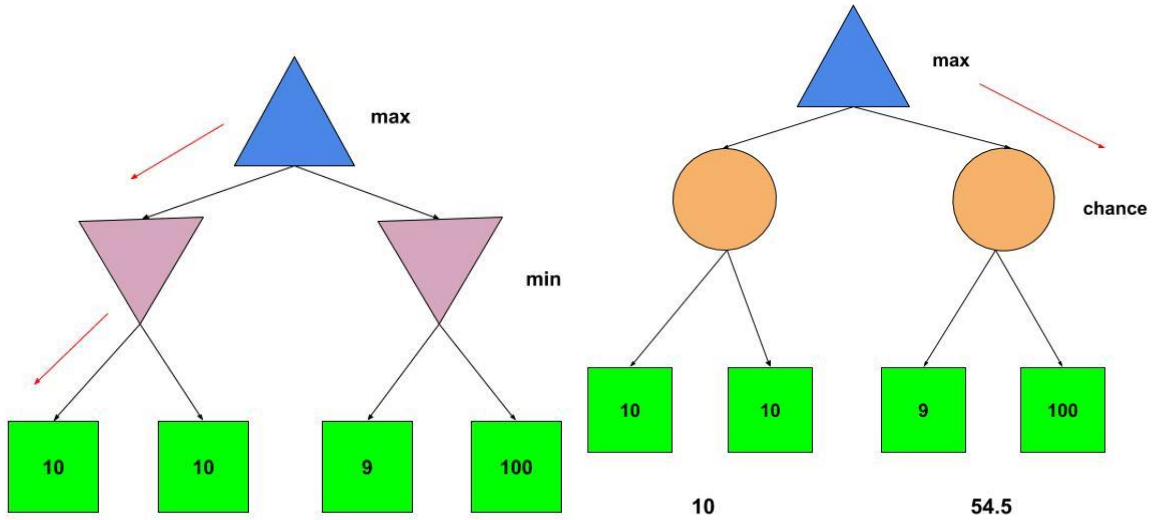
Hicimos luego un ratio de 20 partidas por agente de los cuales se obtuvieron los siguientes resultados:

Minimax: en 260 minutos se completaron 20 juegos: 9 ganados y 11 perdidos por el jugador

Expectimax: en 256 minutos se completaron 20 juegos: 16 ganados y 4 perdidos
Si vamos a la teoría de juegos suma 0 esto tiene sentido, ya que el jugador juega en contra de un tablero randomico, con chances diferentes de introducir una ficha 2 o 4. El tablero no juega óptimamente en contra del jugador.

El algoritmo de búsqueda Expectimax es un algoritmo de teoría de juegos utilizado para maximizar la utilidad esperada. Es una variación del algoritmo Minimax. Mientras que Minimax asume que el adversario (el minimizador) juega de manera óptima, Expectimax no lo hace. Esto es útil para modelar entornos donde los agentes adversarios no son óptimos o sus acciones se basan en el azar.

Sabemos que el tablero juega al azar con diferentes chances de agregar una celda valor 2 o 4.



Los nodos Chance toman el promedio de todas las utilidades disponibles y nos dan la "utilidad esperada". El nodo maximizador elige el subárbol correcto para maximizar las utilidades esperadas.

Ventajas de Expectimax sobre Minimax:
El algoritmo Expectimax ayuda a aprovechar los oponentes no óptimos.
A diferencia de Minimax, Expectimax 'puede correr un riesgo' y terminar en un estado con una mayor utilidad ya que los oponentes son aleatorios (no óptimos).

Desventajas:
Expectimax no es óptimo. Puede llevar a que el agente pierda (terminando en un estado con menor utilidad)
Expectimax requiere que se explore el árbol de búsqueda completo. No se puede hacer ningún tipo de poda, ya que el valor de una única utilidad inexplorada puede cambiar drásticamente el valor de expectimax. Por lo tanto, puede ser lento.
Investigando un poco más sobre este dato y tipos de poda, se mencionó en clase un tipo alpha beta prunning, que consiste en permitir buscar mucho más rápido e incluso adentrarnos en niveles más profundos en el árbol del juego. Corta ramas en el árbol del juego que no necesitan ser buscadas porque ya existe un mejor movimiento disponible. Se llama poda alfa-beta porque pasa 2 parámetros adicionales en la función minimax, a saber, alfa y beta.

Es sensible a las transformaciones monótonas en los valores de utilidad.
Para minimax, si tenemos dos estados S1 y S2, si S1 es mejor que S2, las magnitudes de los valores de la función de evaluación f (S1) y f (S2) no importan siempre que f (S1) > f (S2).
Para expectimax, las magnitudes de los valores de la función de evaluación son importantes.