DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Machine Learning

Trabajo práctico 2

Qlearning

Resumen

Integrante	LU	Correo electrónico
Negri, Franco	893/13	franconegri2004@hotmail.com
Podavini Rey, Martín Gastón	483/12	marto.rey2006@gmail.com

Palabras claves:

TP, 4 en linea, qlearning

Índice

1. Introduccion

En este trabajo practico buscamos modelar el juego de 4 en linea, modelar jugadores que puedan jugar sobre ese modelo y por ultimo, implementar una clase jugador que utilice tecnicas de q-learning para observar su comportamiento al variar parametros o entrenandolo bajo ciertas circunstancias particulares que concideremos interesantes.

Algo interesante para notar aqui es que, en este juego se contará con la participación de dos ajentes que compiten entre ellos, por lo que deberá prestarse particular atención en donde se modelará la etapa de recompensa del algoritmo.

Otra idea que intentaremos explorar aqui es que politica de exploración utilizar en nuestro algoritmo y veremos que impacto tiene cada una de ellas en los resultados obtenidos.

2. Desarrollo

2.1. Modelado

2.2. Modelado Del Juego

El juego en si consistirá en una lista de listas donde cada lista representará una columna del tablero. En cada turno un jugador elejirá una columna numerada del 0 al n, siendo n el numero de columnas totales y el juego se encargará. Ambos jugadores contarán en el momento de la elección con el estado actual del tablero.

Luego de cada jugada, el juego chequará si el jugador ganó o si ya no hay mas movimientos posibles, terminando el juego e informando que jugador gano o si fue empate.

El pseudocodigo del juego será, entonces el siguiente:

```
1: While True
2:     columna = player.move(tablero)
3:     jugar_ficha(columna, tablero)
4:     if jugador_gano(tablero)
5:         return player
6:     if tablero_lleno(tablero)
7:         return empate
8:     player = otro_jugador(player)
```

Algorithm 1: jugar()

Como ya adelantamos en la introducción, es necesario definir en que etapa del algoritmo se le asignará la recompensa al algoritmo de q-learning. Una opción bastante sencilla e intuitiva es la de asignar recompensas en el momento en que algun jugador gana la partida. Por ejemplo al ganar la partida uno de los jugadores, se le asigna una recompensa de 1 a el y una recompensa de -1 al contrincante. Siguiendo con el lineamiento anterior, tambien sería posible asignarles recompensas en el momento en que se empata, por ejemplo, asignandoles a ambos jugadores una recompensa de 0,5

En caso de que no se haya llegado a un estado final, una posible propuesta es asignarle al otro jugador una recompenza por ejemplo de 0.

El pseudocodigo entonces, pasaría a verse de la siguiente manera:

```
While True
    columna = player.move(tablero)
    jugar_ficha(columna, tablero)
    if jugador_gano(tablero)
        player.recompensa(1)
        otro_jugador(player).recompensa(-1)
        return player
    if tablero_lleno(tablero)
        player.recompensa(0.5)
        otro_jugador(player).recompensa(0.5)
        return empate
    otro_jugador(player).recompensa(0)
    player = otro_jugador(player)
```

Algorithm 2: jugar()

2.3. Modelado De Jugadores

Un jugador estará compuesto por dos metodos basicos, uno que llamaremos *move*() que, dado un estado del tablero devuleve una acción valida (tirar ficha en la primera columna, en la segunda, etc) y una función *reward*() que nos dirá en que estado quedo el juego luego de movernos y de que el oponente moviera y una recompensa que utilizaremos para actualizar la matriz Q en caso de que el jugador lo requiera.

En particular, para la clase jugador que implemente q-learning el algoritmo para decidir que acción realizar vendrá dado de la siguiente manera:

```
acciones_validas = elegir_acciones_validas(tablero)
Para cada acción en acciones_validas
    obtener q para accion
    guardarlo en lista de qs_validos
accion_elegida = elegir_acción(acciones_validas,qs_validos)
retornar accion_elegida
```

Algorithm 3: move(tablero)

Y la función learn, que basicamente consiste en adaptar la función de aprendisaje vista en la teorica:

```
prev_q = getQ(estado_previo, accion_elegida)
maxqnew = tomar maximo q de tomar una acción en el tablero resultante
q[(estado_previo, accion_elegida)] = prev + α * ((reward + γ*maxqnew) - prev)
```

Algorithm 4: learn(tablero,recomensa)

Aqui quedan por definir varios parametros, entre ellos α, γ y los valores iniciales para la matriz Q. En primera instancia decidimos tomar valores que concideramos apropiados de a cuerdo a datos empiricos de la implementación y que concideramos razonables. Tomaremos $\alpha = 0{,}4$, $\gamma = 0{,}9$ y todas los valores de Q iniciarán con 1.

En la siguiente sección propondremos distintos algoritmos para elejir una acción tomado del jugador y teorizar como estas impactan en el aprendisaje.

2.4. Planteo De Distintas Estrategias de Movimiento

En esta sección plantearemos distintas estrategias que utilizará nuestro jugador al momento de elegir si explorar nuevas posibilidades o utilizar el mejor camino conocido.

En particular plantearemos estas tres estrategias:

- Estrategia Greedy: toma un camino random con probabilidad $\epsilon\,\%$ y en caso contrario utilice el mejor brazo conocido.
- Estrategia ϵ-first: toma un camino random en las primeras ϵ iteraciones y luego toma el mejor camino conocido.
- Estrategia Softmax: basada en una función probabilistica que desarrollaremos a continuación.

La estrategia Softmax se basa en darle probabilidades distintas a cada acción posible dependiendo de la recomensa esperada de cada una de ellas.

En particular la probabilidad de cada una de las acciones vendrá dada por:

La variable τ se denomina parametro de temperatura, para temperaturas cercanas a infinito todas las acciones tienen aproximadamente la misma probabilidad de ser elegidas, mientras que para temperaturas bajas (cercanas a cero) la probabilidad de la accion de mayor recompensa tenderá a 1.

En particular nuestra estrategia consistirá en comenzar con una temperatura alta para favorecer la exploración e ir gradualemente disminuyendola para favorecer mas a aquellas con mayor recopensa. Faltará definir de manera experimental cual será el valor inicial de τ y de que manera reduciremos su valor (algunas posiblidades son de manera linea, logaritmica, etc).

2.5. Analisis Espacio De Estados

Segun el modelo planteado para la resolución del problema, la matriz Q tendrá como como columnas los diferentes estados en los que se puede encontrar el tablero y como filas las acciones que se pueden realizar sobre ese determinado tablero. Dado que la cantidad de estados diferentes en los que se puede encontrar el tablero tiene complegidad PSPACE y que la cantidad la acciones es acotada (a lo sumo m), podemos determinar que la complegidad espacial de nuestro algoritmo esta en PSPACE... not cool.

Si bien esto hace que la complegidad de explorar cada una de las posibilidades sea extremadamente costosa, consideramos que para la mayoría de los casos no es necesario concer todo el arbol sino los estados mas generales.

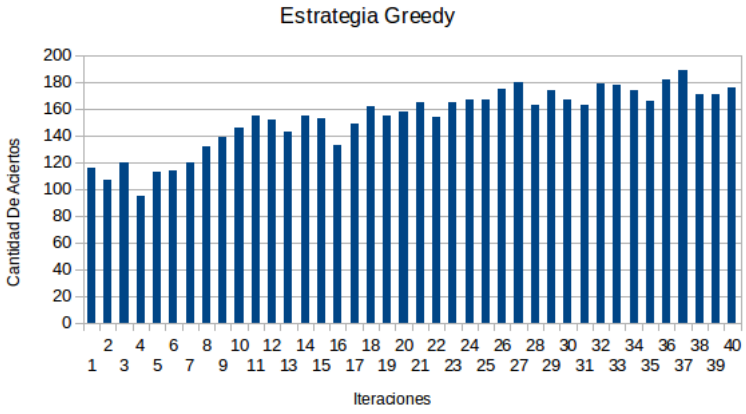
3. Experimentación

4. Estrategias vs Jugador Random

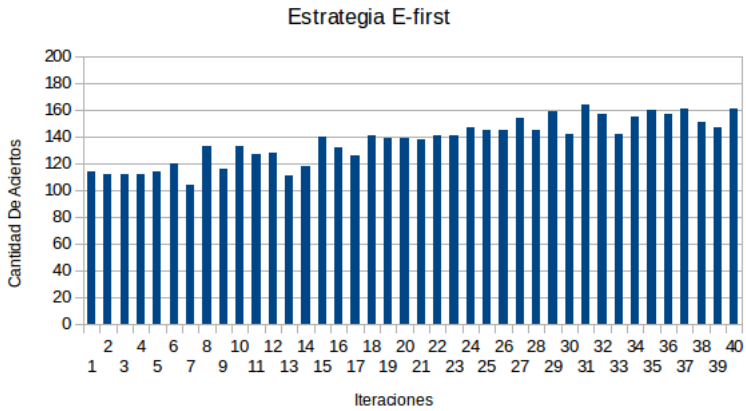
Como primera instancia en la etapa de experimentación, haremos competir a un jugador q-learner contra un jugador que elige entre los posibles movimientos con probabilidad uniforme. Utilizaremos las tres estrategias enunciadas previamente para ver como se comporta cada una.

La metodología utilizada para observar como evoluciona el algoritmo de q-learning será la de hacer jugar a ambos jugadores 10000 veces randomizando cual es el que comienza primero. Luego tomaremos estos resultados y cada 200 juegos, graficaremos cuantas veces ganó el algoritmo q-learning.

Para la estrategia greedy, eligiendo un $\epsilon = 0{,}1$ obtuvimos los siguientes resultados:

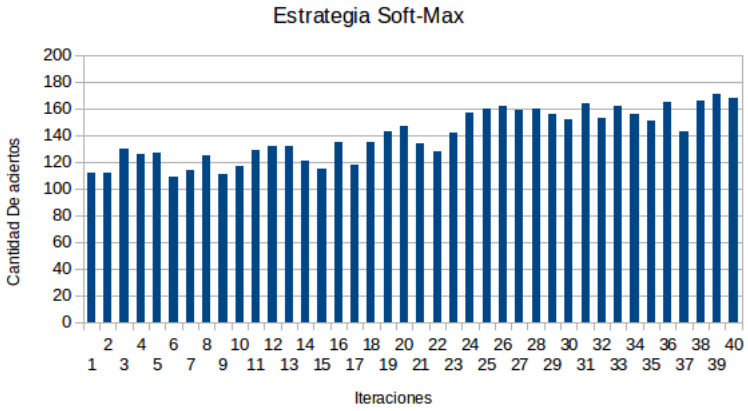


Para la estrategia ϵ-first eligiendo un $\epsilon = 10\,\%$, y los resultados fueron los siguientes:



Puede verse como en el primer 10 % de las iteraciones el algorimo gana la mitad de las veces y prierde la otra mitad, lo que era de esperarse para una politica completamente random.

Con la tercera politica Soft-Max, decidimos utilizar una temperatura incial igual a 1 y una función de calor que decrese de manera logaritmica con cada iteración:



4.1. Performance de dos q-learners compitiendo

4.2. Performance al cambiar la inicialización de Q

5. Concluciones