MBIT School
BUSINESS DATA INTELLIGENCE
We know how

Proyecto “Programa Experto de Deep Learning”

Aprendizaje Reforzado sobre el entorno ConnectX de Kaggle

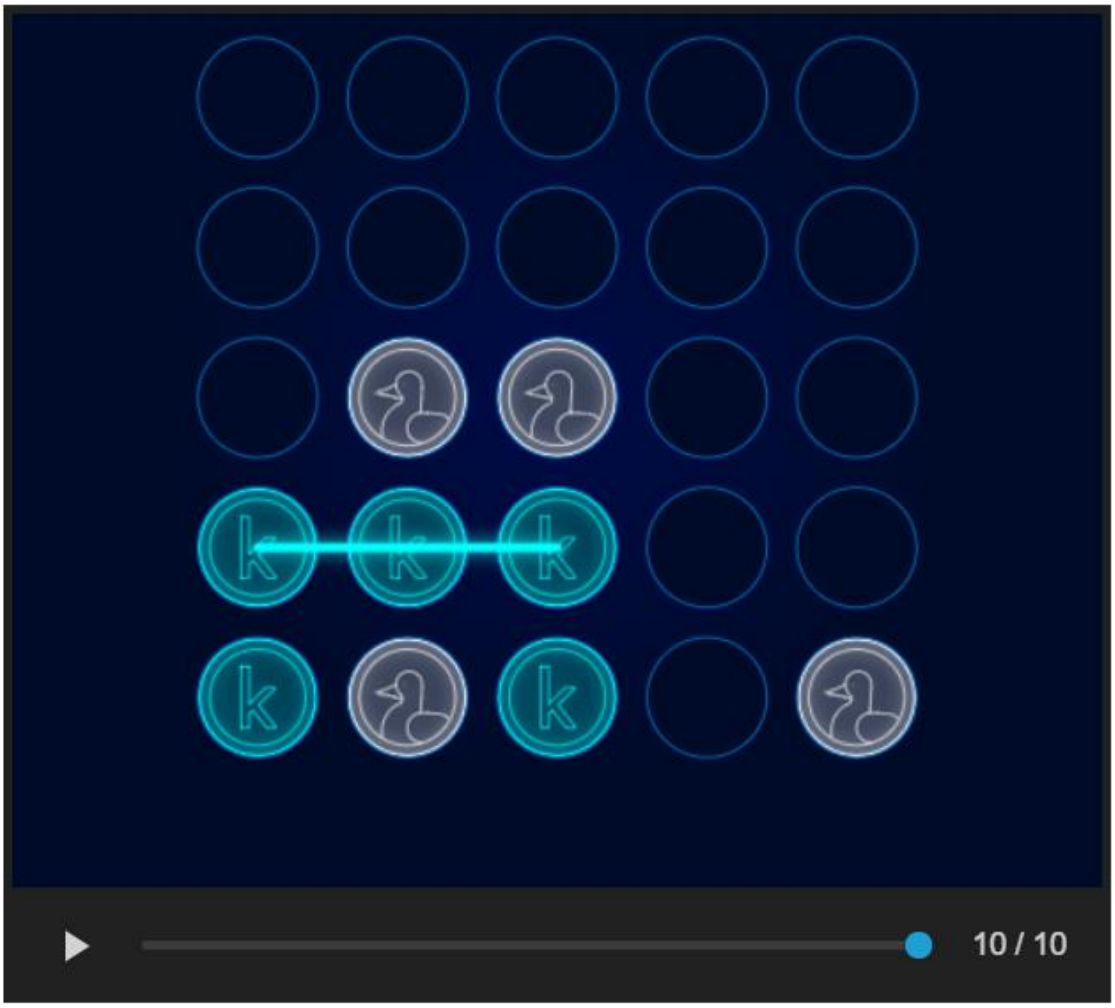
1. OBJETIVO

El objetivo de este proyecto es aplicar los diferentes métodos de Aprendizaje Reforzado sobre un entorno sencillo. En este caso hemos seleccionado el entorno ConnectX de Kaggle (https://www.kaggle.com/c/connectx)

Este entorno simula el juego Conecta 4 en el que tienes un tablero con p filas y q columnas y participan dos jugadores. Cada jugador deposita una ficha en una de las columnas, y esta ocupa la primera posición libre empezando a contar desde abajo. Una vez depositada la ficha, le cede el turno al otro jugador.

El primer jugador que consiga colocar x fichas en horizontal, vertical o en diagonal, gana la partida.

En nuestro caso, para simplificar el aprendizaje, hemos definido un tablero de 5 filas x 5 columnas, en el que hay que conseguir colocar 3 fichas de manera consecutiva.



En este entorno contamos con dos agentes (los dos jugadores) y no hay recompensas intermedias (solamente al ganar o perder la partida). Si queremos tener esas recompensas intermedias tenemos que generarlas nosotros, no vienen por defecto en el entorno.

Este es un problema estacionario (no varía en el tiempo) y discreto (conjunto discreto de posibles acciones)

2. MÉTODOS

2.1. Monte Carlo Control

Utilizamos Monte Carlo para aproximar la política óptima de la siguiente manera:

- Generamos N episodios aleatorios (200.000)
- En cada episodio recorremos los pasos y evaluamos el valor de cada estado.
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mayor valor (greedy).
- El valor de cada estado será el descuento elevado al número de pasos que quedan hasta el final, y multiplicado por la recompensa del último paso: γ^{pasos} * R(final)

$$V_{k+1}(s) = max_a \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma \cdot V_k(s')]$$

- La actualización del valor de los estados se hace de manera incremental (cada vez más pequeña)

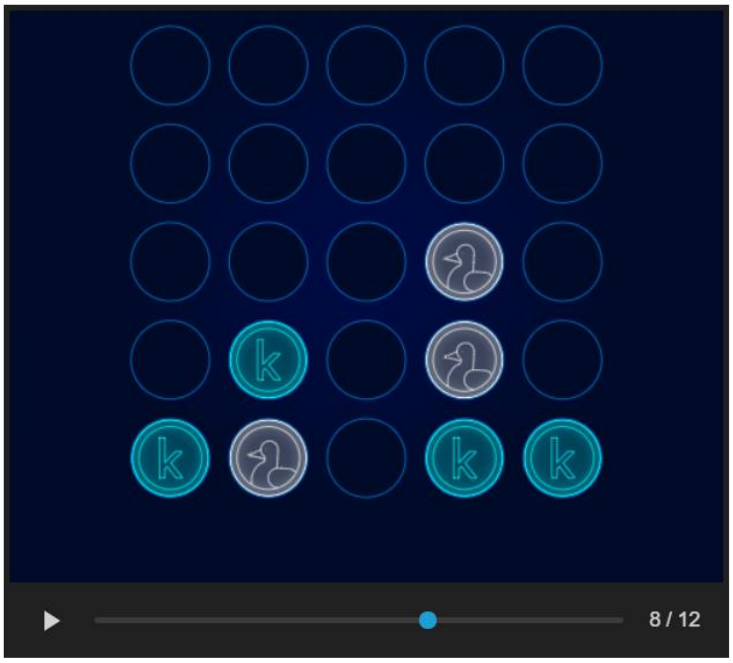
Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. Debido a esto, es igual utilizar first-visit o every-visit.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que seleccionamos aleatoriamente las acciones.

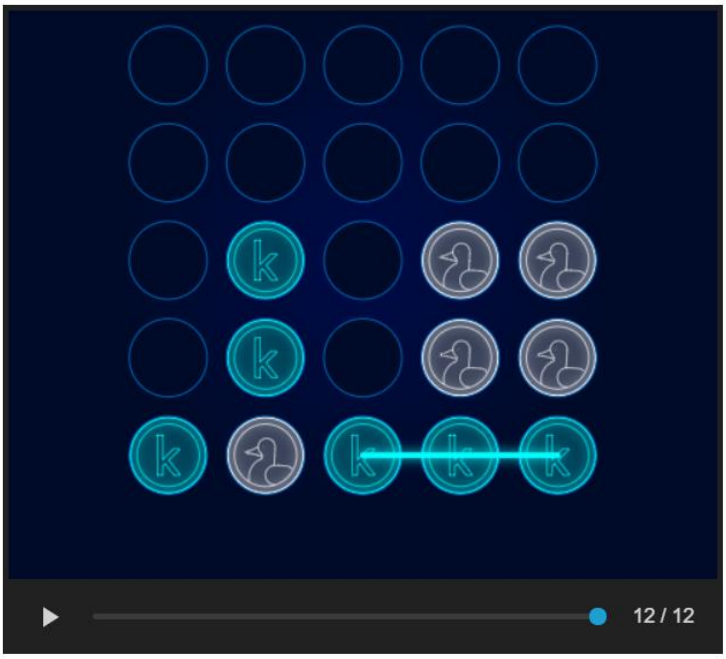
La generación de episodios se hace contra el agente preconfigurado "negamax". Contra el agente "random" el entrenamiento es más rápido, pero no es muy bueno. Por ejemplo, el agente "random" no aprovecha situaciones en las que podría ganar la partida en el siguiente movimiento, lo que hace que esos tableros (estados) no se evalúen correctamente.

Ejemplo:

En este caso, el jugador 2 podría ganar la partida colocando una ficha en la 4ª columna

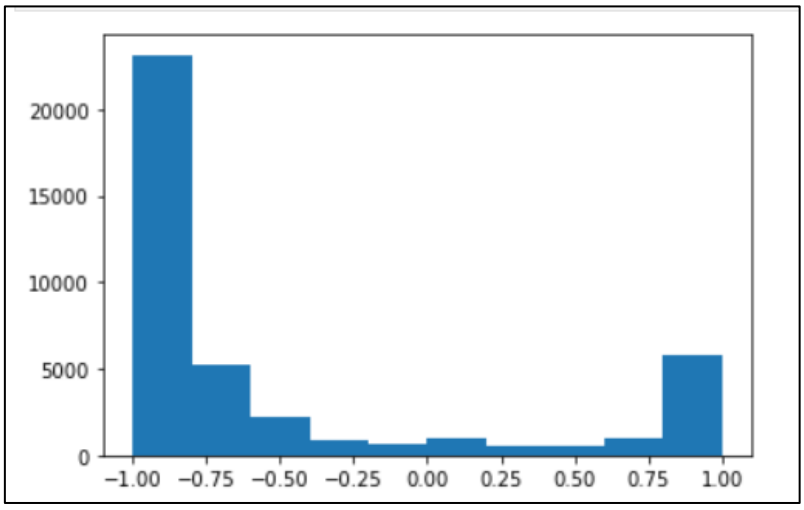


Sin embargo, la coloca aleatoriamente en la quinta columna, y el jugador 1 termina ganando la partida



El primer tablero se evalúa positivamente para el jugador 1, pero es un tablero muy negativo, ya que el jugador 2 podría ganar la partida en el siguiente movimiento.

Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los valores de los estados.



Al ser episodios cortos, y tener un descuento cercano al 1, la mayoría de los valores se agrupan en torno al -1 y 1. Se puede ver que hay una mayoría de valores negativos, ya que el jugador 1 juega aleatoriamente, y el jugador 2 juega ejecutando la acción que más le conviene. Eso hace que el jugador 2 gane muchas más partidas que el 1.

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado
random	negamax	-0,80
random	MonteCarlo Control	-0,80
negamax	random	0,80
negamax	MonteCarlo Control	0,80
MonteCarlo Control	random	1,00
MonteCarlo Control	negamax	1,00

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	MonteCarlo Control	2,00
2	negamax	1,40
3	random	-3,40

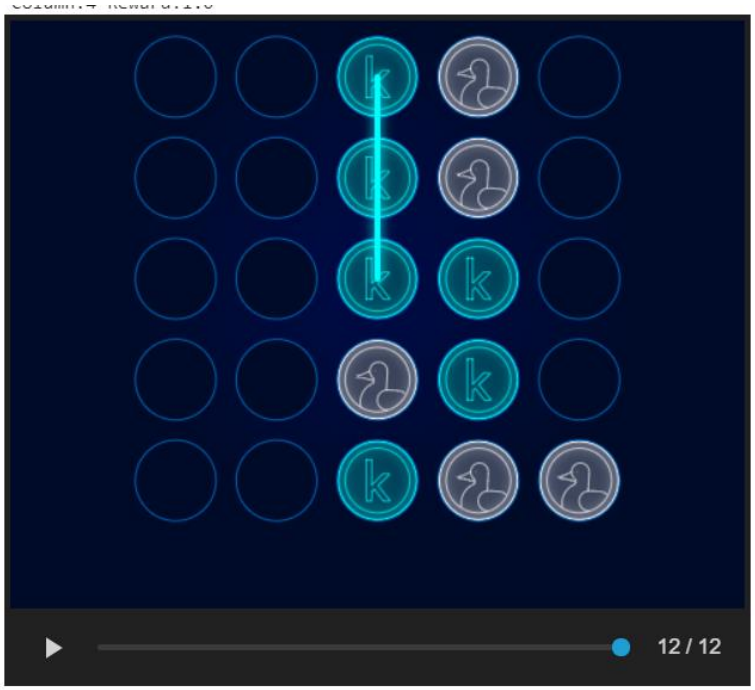
Conclusiones

Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que en ocasiones se llega a estados (tableros) para los que no tiene ninguna información. En las partidas de entrenamiento el jugador 1 seleccionaba la columna aleatoriamente, pero en estas partidas de evaluación el jugador 1 selecciona la mejor jugada.

Ejemplos

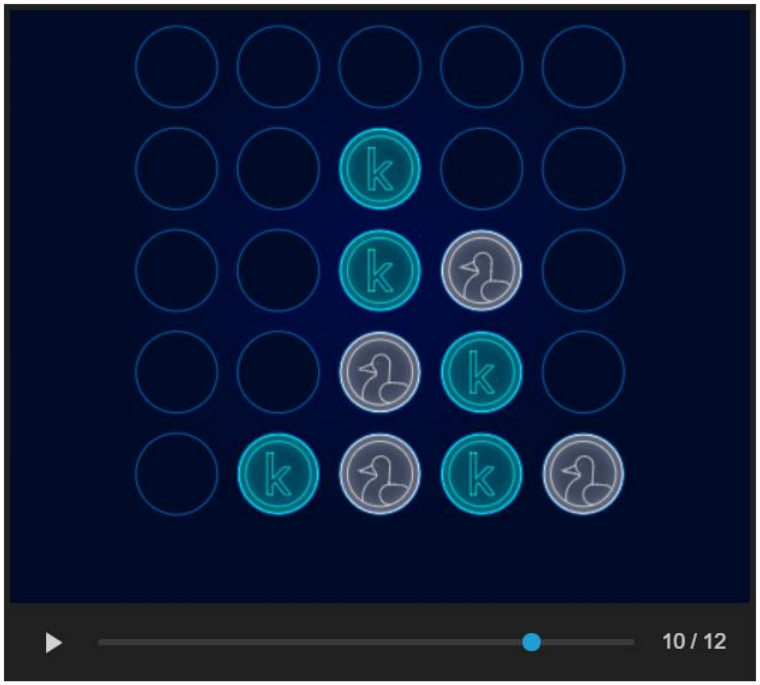
Ejemplo: MonteCarlo Control (azul) vs negamax (gris)

Nuestro agente va agrupando fichas y genera una situación en la que tenemos dos opciones para ganar (columnas 3 y 5)

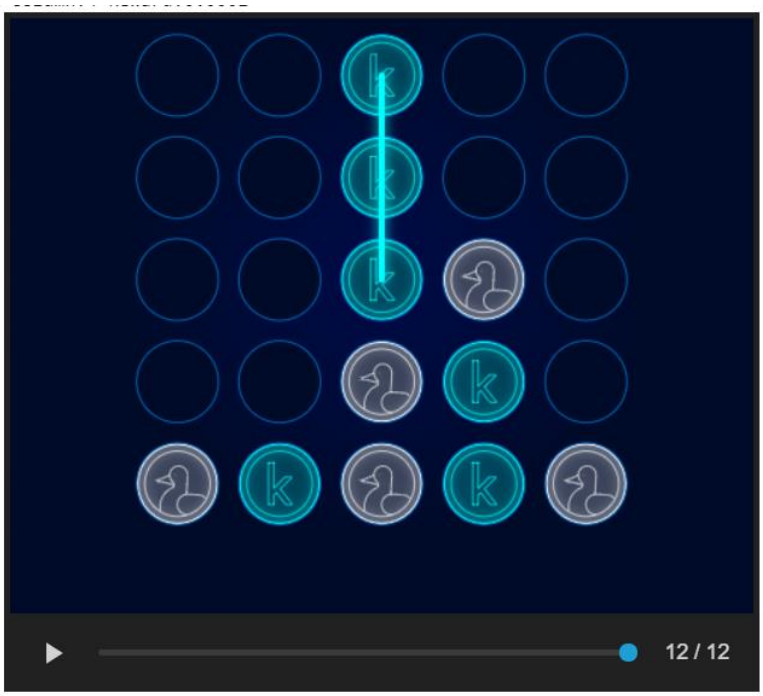


Ejemplo: negamax (azul) vs MonteCarlo Control (gris)

Llegamos a una situación en la que el rival puede ganar con el siguiente movimiento (columna 3).



En vez de colocar la ficha en la columna 3 para evitar perder la partida, la coloca en la columna 1, y el jugador 1 gana la partida.



2.2. Temporal Difference Q-learning

Utilizamos Q-learning para aproximar la función Q-valor, y con ella la política óptima de la siguiente manera:

- Generamos N episodios (200.000)
- En cada paso de cada episodio evaluamos el Q-valor de cada par (estado, acción). Hay que tener en cuenta que cada paso (cada llamada a la función step()) en realidad incluye dos pasos (nuestro movimiento y el del oponente), así que hay evaluar el Q-valor de ambos ya que nuestro agente puede ejercer tanto de jugador 1 como de jugador 2
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mayor valor (greedy).
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:

$$New\ Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma\ maxQ'(s',a') - Q(s,a)]$$

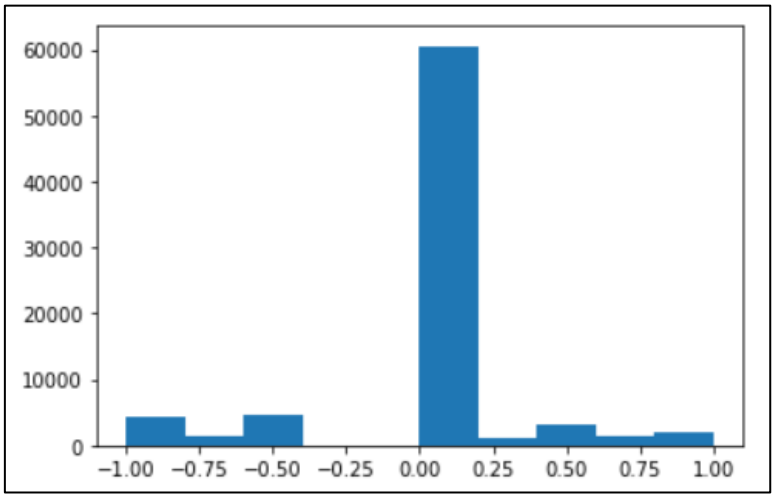
- La actualización del valor de los estados se hace utilizando un parámetro α (learning rate).

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que, dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

La generación de episodios se hace contra el agente preconfigurado "negamax".

Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los Q-valores de los estados.



Como vemos, en este caso, la mayoría de los valores se agrupan en torno al cero. Al no tener este entorno recompensas intermedias, inicialmente sólo tienen Q-valor distinto de cero los estados-acciones finales. Ese valor se va propagando hacia atrás a medida que se generan episodios, pero es más lento que MonteCarlo, donde se evalúan los episodios completos.

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado
random	negamax	-1,00
random	MonteCarlo Control	-1,00
random	TD Q-learning	-0,60
negamax	random	1,00
negamax	MonteCarlo Control	0,80
negamax	TD Q-learning	1,00
MonteCarlo Control	random	1,00
MonteCarlo Control	negamax	1,00
MonteCarlo Control	TD Q-learning	0,80
TD Q-learning	random	0,40

TD Q-learning	negamax	1,00
TD Q-learning	MonteCarlo Control	1,00

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	MonteCarlo Control	2,00
2	negamax	1,80
3	TD Q-learning	1,20
4	random	-5,00

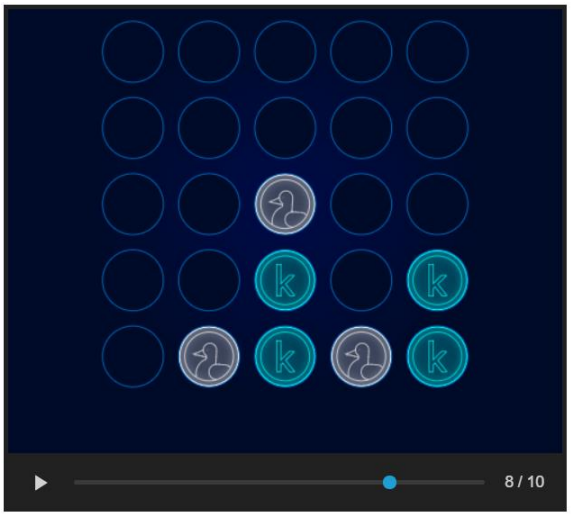
Conclusiones

Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que casi siempre se llega a estados (tableros) para los que no tiene ninguna información. En las partidas de entrenamiento el jugador 1 seleccionaba la columna aleatoriamente, pero en estas partidas de evaluación el jugador 1 selecciona la mejor jugada.

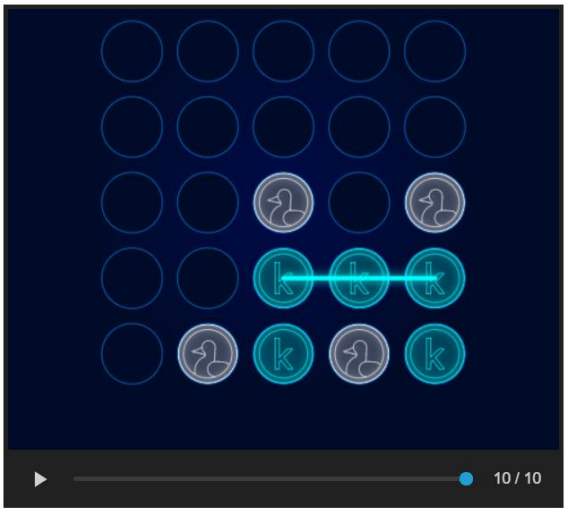
Ejemplos

Ejemplo: TD Q-learning (azul) vs negamax (gris)

Nuestro agente va agrupando fichas y genera una situación en la que tenemos dos opciones para ganar (columnas 4 y 5)



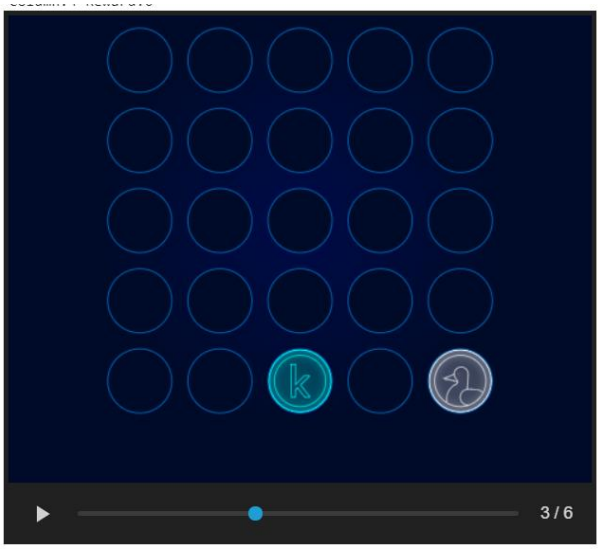
En el siguiente movimiento gana la partida



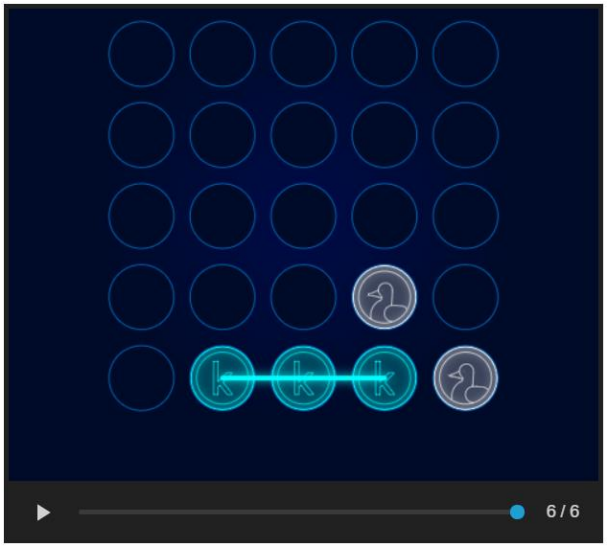
Ejemplo: negamax (azul) vs TD Q-learning (gris)

El jugador 1 (negamax) empieza la partida colocando la ficha en el centro del tablero. Nuestro agente tiene q-valores positivos para las fichas colocadas en las columnas 2 y 4, pero al actuar como jugador 2 busca el mínimo Q-valor, que en este caso es el de las columnas para las que no hay datos (1, 3 y 5). Así que selecciona aleatoriamente entre las 3 columnas y coloca la ficha en la 5.

Como en el entrenamiento el jugador 2 colocaba las fichas con cierto sentido, este no es un estado por el que hayamos pasado, así que a partir de aquí no hay ninguna información en nuestra política (todos los Q-valores están a cero) y se seleccionan las columnas aleatoriamente.



El jugador 1 gana la partida fácilmente



2.3. Temporal Difference SARSA

Utilizamos SARSA para aproximar la función Q-valor, y con ella la política óptima. La diferencia con Q-learning es que en este caso es un método on-policy. La política con la que aprendemos y la política objetivo son la misma (ε-greedy). Esto nos sirve para tener una política más "conservadora", que puede ser útil en problemas en los que la penalización por fallar sea muy grande. Al tomar a veces acciones aleatorias, la política aprende a intentar no fallar incluso en esos casos.

Aproximamos la política objetivo de la siguiente manera:

- Generamos N episodios (200.000)
- En cada paso de cada episodio evaluamos el Q-valor de cada par (estado, acción). Hay que tener en cuenta que cada paso (cada llamada a la función step()) en realidad incluye dos pasos (nuestro movimiento y el del oponente), así que hay evaluar el Q-valor de ambos, ya que nuestro agente puede ejercer tanto de jugador 1 como de jugador 2.
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- Tanto en la política objetivo como en la de exploración se selecciona a veces la acción que nos lleve al estado con mayor valor, y a veces una acción aleatoria (ε-greedy).
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:

$$NewQ(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma Q(s',a') - Q(s,a))$$

- La actualización del valor de los estados se hace utilizando un parámetro α (learning rate).

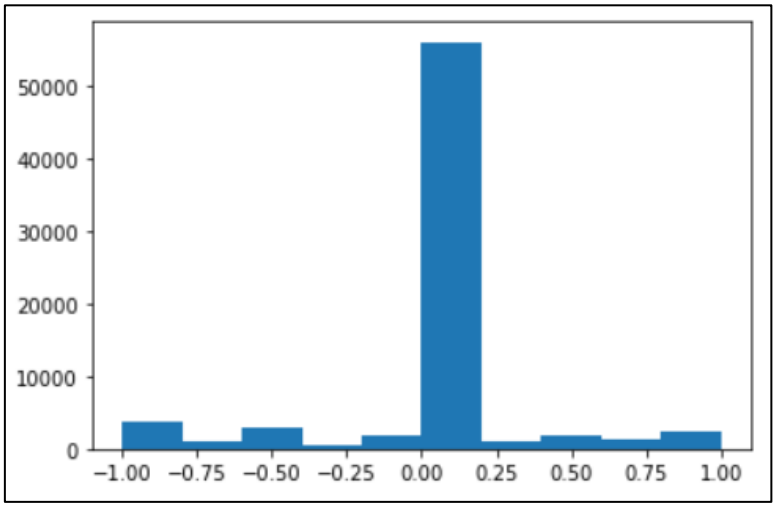
Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método on-policy, ya que la política con la que aprendemos y la política objetivo π son la misma. Dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al

principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

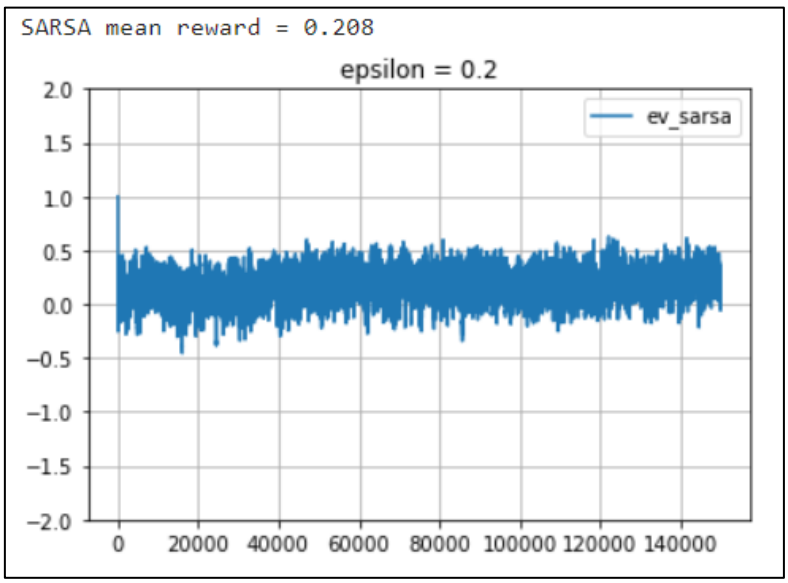
La generación de episodios se hace contra el agente preconfigurado "negamax".

Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los Q-valores de los estados.



Como vemos, en este caso, la mayoría de los valores se agrupan en torno al cero. Al no tener este entorno recompensas intermedias, inicialmente sólo tienen Q-valor distinto de cero los estados-acciones finales. Ese valor se va propagando hacia atrás a medida que se generan episodios, pero es más lento que MonteCarlo, donde se evalúan los episodios completos.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. No es útil verlas directamente, ya que solo hay 1s y -1s. Así que obtenemos la media ponderada de la lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual.



Se puede ver un ligero ascenso, y al final del entrenamiento prácticamente todos los valores medios están por encima de cero

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado
random	negamax	-1,00	TD Q-learning	random	0,80
random	MonteCarlo Control	-1,00	TD Q-learning	negamax	1,00
random	TD Q-learning	0,20	TD Q-learning	MonteCarlo Control	1,00
random	TD Sarsa	-0,20	TD Q-learning	TD Sarsa	0,40
negamax	random	1,00	TD Sarsa	random	0,80
negamax	MonteCarlo Control	1,00	TD Sarsa	negamax	0,80
negamax	TD Q-learning	1,00	TD Sarsa	MonteCarlo Control	0,60
negamax	TD Sarsa	1,00	TD Sarsa	TD Q-learning	0,80
MonteCarlo Control	random	0,60			
MonteCarlo Control	negamax	1,00			
MonteCarlo Control	TD Q-learning	1,00			
MonteCarlo Control	TD Sarsa	0,80			

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	negamax	2,20
2	MonteCarlo Control	1,80
3	TD Sarsa	1,00
4	TD Q-learning	0,20
5	random	-5,20

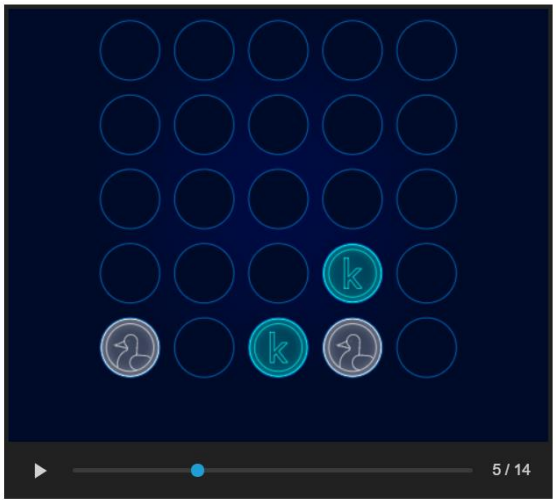
Conclusiones

Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que casi siempre se llega a estados (tableros) para los que no tiene ninguna información. En las partidas de entrenamiento el jugador 1 seleccionaba la mejor columna (pero con poca información) o directamente de manera aleatoria. Sin embargo, en estas partidas de evaluación el jugador 1 selecciona la mejor jugada para la mayoría de los agentes.

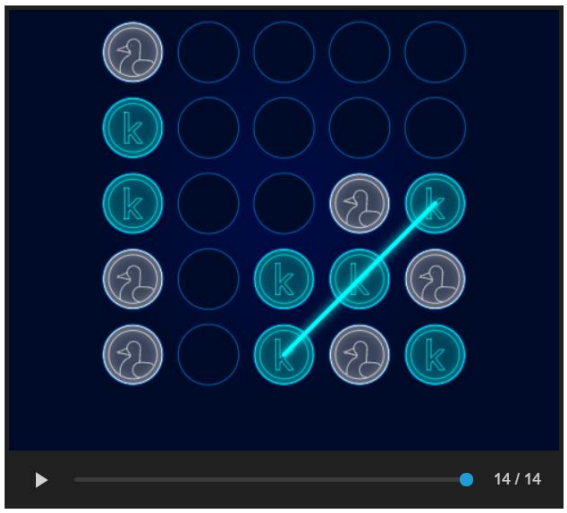
Ejemplos

Ejemplo: TD Sarsa (azul) vs negamax (gris)

Nuestro agente toma algunas decisiones aleatorias. Por ejemplo, en este caso el mejor movimiento es colocar la ficha en la columna 3, pero la coloca en la 5.



Finalmente acaba ganando la partida

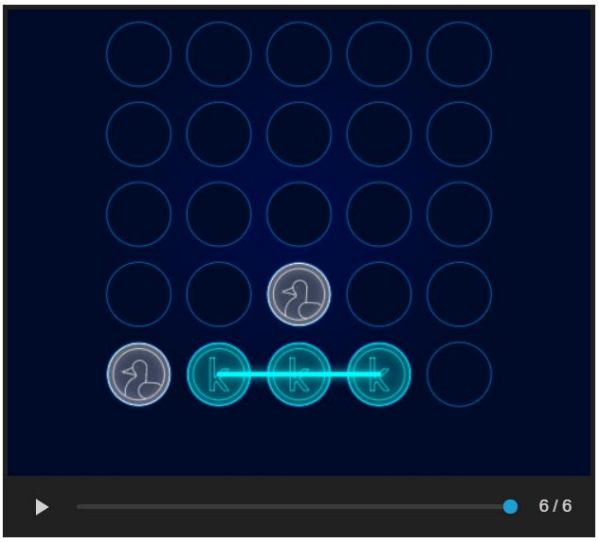


Ejemplo: negamax (azul) vs TD Sarsa (gris)

El jugador 1 (negamax) empieza la partida colocando la ficha en el centro del tablero. Nuestro agente tiene q-valores positivos para las fichas colocadas en las columnas 2 y 4, pero al actuar como jugador 2 busca el mínimo Q-valor, que en este caso es el de las columnas para las que no hay datos (1, 3 y 5). Así que selecciona aleatoriamente entre las 3 columnas y coloca la ficha en la 3.

Como en el entrenamiento el jugador 2 colocaba las fichas con cierto sentido, este no es un estado por el que hayamos pasado, así que a partir de aquí no hay ninguna información en nuestra política (todos los Q-valores están a cero) y se seleccionan las columnas aleatoriamente.

El jugador 1 gana la partida fácilmente



2.4. Temporal Difference Expected SARSA

Utilizamos Expected SARSA para aproximar la función Q-valor, y con ella la política óptima. La diferencia con Sarsa es que, a la hora de estimar el retorno, en vez de usar la política objetivo, se utiliza la esperanza de todas las posibles acciones. En este caso, al tener todas las acciones la misma probabilidad, se obtiene la media en vez de una media ponderada por la probabilidad.

De esta manera se elimina la aleatoriedad a la hora de estimar el retorno, aunque aumenta el coste computacional.

Aproximamos la política objetivo de la siguiente manera:

- Generamos N episodios (200.000)

- En cada paso de cada episodio evaluamos el Q-valor de cada par (estado, acción). Hay que tener en cuenta que cada paso (cada llamada a la función step()) en realidad incluye dos pasos (nuestro movimiento y el del oponente), así que hay evaluar el Q-valor de ambos, ya que nuestro agente puede ejercer tanto de jugador 1 como de jugador 2.
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- Tanto en la política objetivo como en la de exploración se selecciona a veces la acción que nos lleve al estado con mayor valor, y a veces una acción aleatoria (ε-greedy).
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:

$$NewQ(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma \sum_a \pi(a'|s')Q(s',a') - Q(s,a))$$

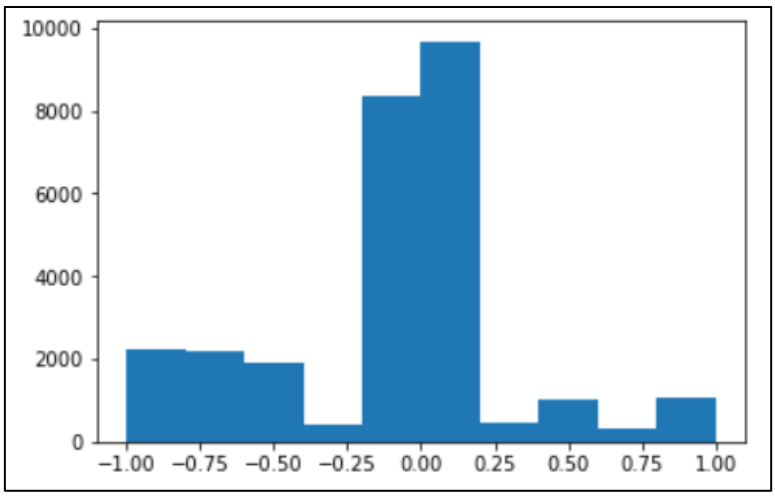
- La actualización del valor de los estados se hace utilizando un parámetro α (learning rate).

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método on-policy, ya que la política con la que aprendemos y la política objetivo π son la misma. Dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

La generación de episodios se hace contra el agente preconfigurado "negamax".

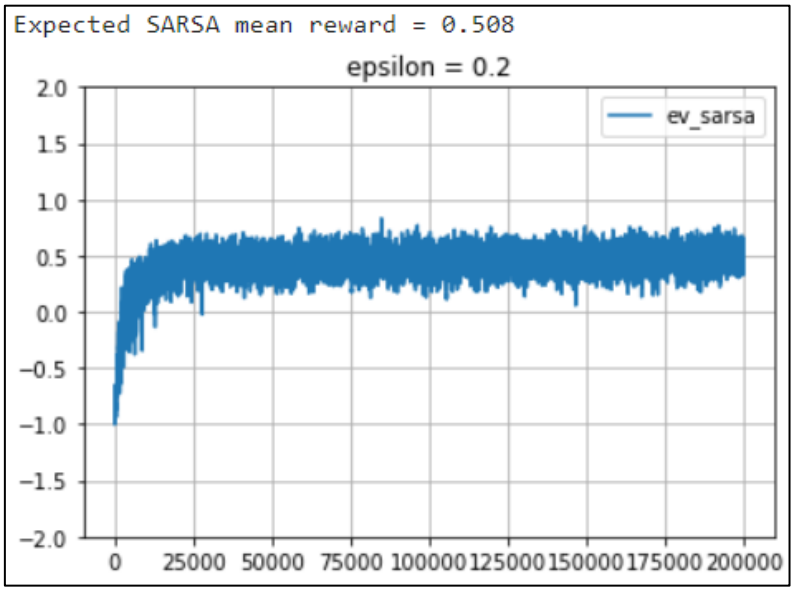
Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los Q-valores de los estados.



Como vemos, en este caso, la mayoría de los valores se agrupan en torno al cero. Al no tener este entorno recompensas intermedias, inicialmente sólo tienen Q-valor distinto de cero los estados-acciones finales. Ese valor se va propagando hacia atrás a medida que se generan episodios, pero es más lento que MonteCarlo, donde se evalúan los episodios completos.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. No es útil verlas directamente, ya que solo hay 1s y -1s. Así que obtenemos la media ponderada de la

lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual.



Se puede ver un claro ascenso en los primeros 25.000 episodios que se ralentiza después. Al final del entrenamiento hay una recompensa media de 0,5.

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado
random	negamax	-1,00	TD Q-learning	random	0,40
random	MonteCarlo Control	-1,00	TD Q-learning	negamax	1,00
random	TD Q-learning	0,40	TD Q-learning	MonteCarlo Control	1,00
random	TD Sarsa	0,40	TD Q-learning	TD Sarsa	0,00
random	TD Expected Sarsa	-0,40	TD Q-learning	TD Expected Sarsa	0,60
negamax	random	1,00	TD Sarsa	random	0,20
negamax	MonteCarlo Control	0,80	TD Sarsa	negamax	0,80
negamax	TD Q-learning	1,00	TD Sarsa	MonteCarlo Control	0,80
negamax	TD Sarsa	1,00	TD Sarsa	TD Q-learning	0,60
negamax	TD Expected Sarsa	1,00	TD Sarsa	TD Expected Sarsa	0,40
MonteCarlo Control	random	0,80	TD Expected Sarsa	random	0,60
MonteCarlo Control	negamax	1,00	TD Expected Sarsa	negamax	0,40
MonteCarlo Control	TD Q-learning	1,00	TD Expected Sarsa	MonteCarlo Control	0,00
MonteCarlo Control	TD Sarsa	0,60	TD Expected Sarsa	TD Q-learning	0,40

MonteCarlo Control	TD Expected Sarsa	0,80	TD Expected Sarsa	TD Sarsa	0,40

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	MonteCarlo Control	2,60
2	negamax	2,60
3	TD Sarsa	0,40
4	TD Q-learning	-0,40
5	TD Expected Sarsa	-0,60
6	random	-4,60

Conclusiones

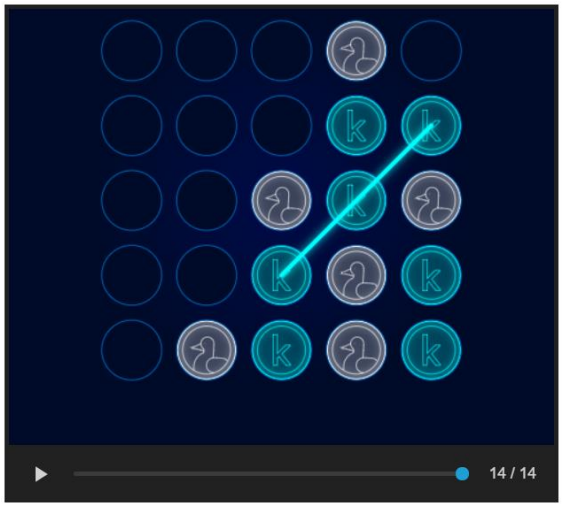
Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que casi siempre se llega a estados (tableros) para los que no tiene ninguna información. En las partidas de entrenamiento el jugador 1 seleccionaba la mejor columna (pero con poca información) o directamente de manera aleatoria. Sin embargo, en estas partidas de evaluación el jugador 1 selecciona la mejor jugada para la mayoría de los agentes.

Ejemplos

Ejemplo: TD Expected Sarsa (azul) vs negamax (gris)

Nuestro agente juega de manera inteligente a pesar de tomar algunas decisiones aleatorias. Por ejemplo, en este caso el mejor genera una situación en la que el agente puede ganar la partida independientemente de lo que haga el adversario. Si el jugador 2 no coloca la ficha en la columna 5, nuestro agente puede ganar en el siguiente movimiento, y si la coloca, también puede ganar (como acaba pasando).

Finalmente acaba ganando la partida a pesar de que el adversario ha intentado evitarlo

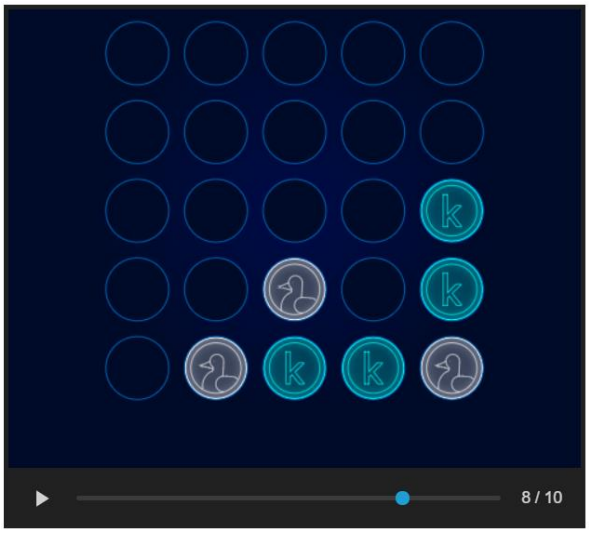


Ejemplo: negamax (azul) vs TD Expected Sarsa (gris)

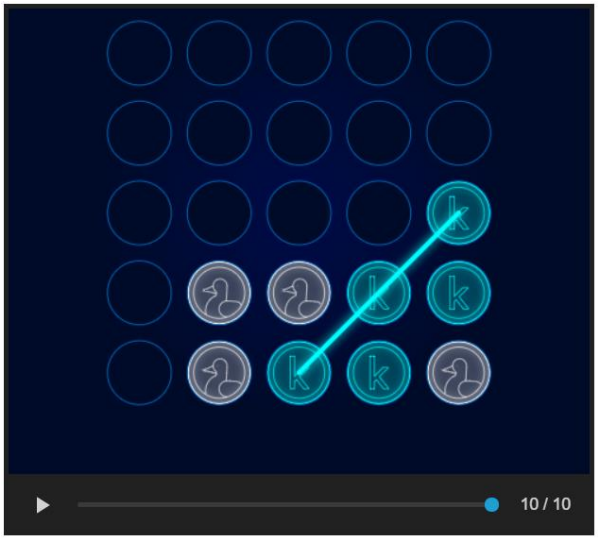
Como en el resto de algoritmos, cuando nuestro agente actúa como jugador 2 suele llegar a una situación en el tablero en la que tiene pocos datos o directamente no tiene ninguno y toma una decisión aleatoria (no se ha dado en el entrenamiento y todos los Q-valores están a cero). Como este es el mayor problema, no hay mucha diferencia de comportamiento con Sarsa.

Se produce porque el agente toma una decisión aleatoria, y en el entrenamiento el jugador 2 colocaba las fichas con cierto sentido y de manera determinista.

En este punto no tiene ningún q-valor para evaluar las posibles acciones.



El jugador 1 gana la partida fácilmente



2.5. Temporal Difference Minimax Q-learning

Utilizamos Minimax Q-learning para aproximar la función Q-valor, y con ella la política óptima. La diferencia con el Q-learning clásico es que en este caso no siempre se utiliza el máximo Q-valor del siguiente (estado, acción) para actualizar el Q-valor actual. Al devolver recompensas positivas cuando gana el jugador 1 y recompensas negativas cuando gana el jugador 2, el mejor q-valor no siempre será el máximo. Siempre suponemos que cada jugador va a hacer el "mejor" movimiento.

Si estamos actualizando el valor de Q(s, a) donde a es una acción realizada por el jugador 1, buscaremos el mínimo Q(s', a'), ya que será el mejor movimiento para el jugador 2, que es el que realiza la acción a'.

Del mismo modo si actualizamos Q(s, a) donde a es una acción realizada por el jugador 2, buscaremos el máximo Q(s', a'), ya que será el mejor movimiento para el jugador 1, que es el que realiza la acción a'.

Con esto intentamos mejorar el comportamiento del agente cuando actúa como jugador 2.

Este algoritmo está basado en el paper de Littman:
https://courses.cs.duke.edu//spring07/cps296.3/littman94markov.pdf

Aproximamos la política objetivo de la siguiente manera:

- Generamos N episodios (200.000)
- En cada paso de cada episodio evaluamos el Q-valor de cada par (estado, acción). Hay que tener en cuenta que cada paso (cada llamada a la función step()) en realidad incluye dos pasos (nuestro movimiento y el del oponente), así que hay evaluar el Q-valor de ambos ya que nuestro agente puede ejercer tanto de jugador 1 como de jugador 2

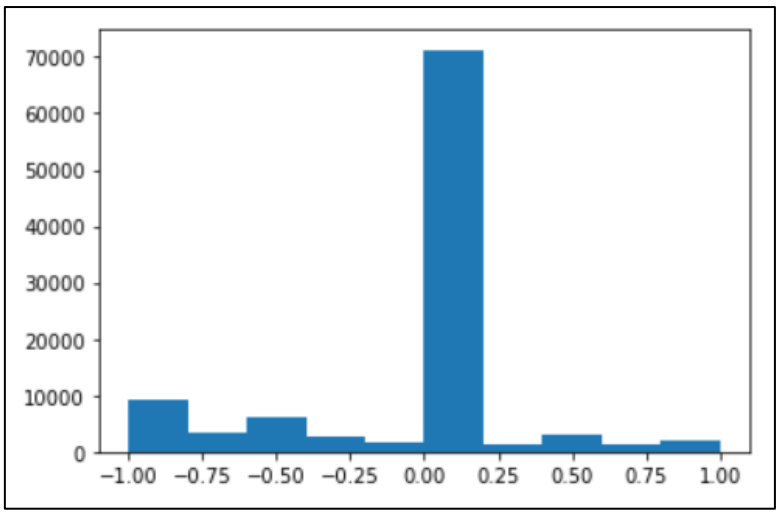
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mejor valor (greedy). Esto es, el mayor para el jugador 1 y el menor para el 2.
  Para el jugador 2 evitamos los q-valores iguales a cero, ya que implican estados que no se han visto durante el entrenamiento y el agente actúa aleatoriamente en esos casos.
  También evitamos q-valores casi iguales a uno, ya que es preferible jugar aleatoriamente que hacerlo sabiendo que vas a perder la partida.
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:
  - Paso 1 (jugador 1)
    $$\text{New } Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \min Q'(s', a') - Q(s, a)]$$
  - Paso 2 (jugador 2)
    $$\text{New } Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \max Q'(s', a') - Q(s, a)]$$
- La actualización del valor de los estados se hace utilizando un parámetro α (learning rate).

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que, dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

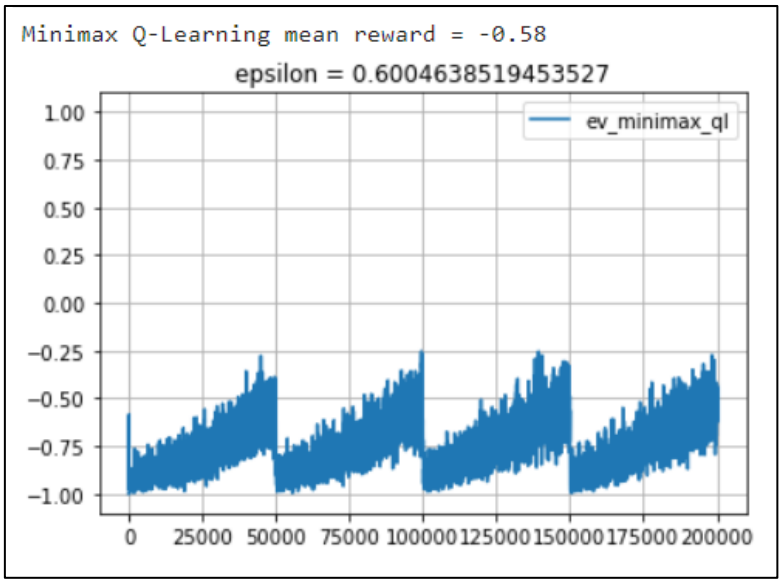
La generación de episodios se hace contra el agente preconfigurado "negamax".

Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los Q-valores de los estados.



Como vemos, en este caso, la mayoría de los valores se agrupan en torno al cero. Al no tener este entorno recompensas intermedias, inicialmente sólo tienen Q-valor distinto de cero los estados-acciones finales. Ese valor se va propagando hacia atrás a medida que se generan episodios, pero es más lento que MonteCarlo, donde se evalúan los episodios completos.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. No es útil verlas directamente, ya que solo hay 1s y -1s. Así que obtenemos la media ponderada de la lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual.



Se puede ver un claro descenso cada 50.000 episodios, ya que se inicializa el valor épsilon para volver a explorar. Durante esos 50.000 episodios se ve que el agente va mejorando.

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado
random	negamax	-0,40	TD Sarsa	random	0,80
random	MonteCarlo Control	-1,00	TD Sarsa	negamax	0,40
random	TD Q-learning	-0,20	TD Sarsa	MonteCarlo Control	0,60
random	TD Sarsa	0,00	TD Sarsa	TD Q-learning	0,80
random	TD Expected Sarsa	-0,20	TD Sarsa	TD Expected Sarsa	0,80
random	TD Minimax QL	-1,00	TD Sarsa	TD Minimax QL	0,20
negamax	random	1,00	TD Expected Sarsa	random	0,60
negamax	MonteCarlo Control	1,00	TD Expected Sarsa	negamax	0,60
negamax	TD Q-learning	1,00	TD Expected Sarsa	MonteCarlo Control	-0,40
negamax	TD Sarsa	0,80	TD Expected Sarsa	TD Q-learning	0,80
negamax	TD Expected Sarsa	1,00	TD Expected Sarsa	TD Sarsa	0,20
negamax	TD Minimax QL	0,60	TD Expected Sarsa	TD Minimax QL	0,60

MonteCarlo Control	random	0,60	TD Minimax QL	random	0,60
MonteCarlo Control	negamax	1,00	TD Minimax QL	negamax	1,00
MonteCarlo Control	TD Q-learning	1,00	TD Minimax QL	MonteCarlo Control	1,00
MonteCarlo Control	TD Sarsa	0,40	TD Minimax QL	TD Q-learning	1,00
MonteCarlo Control	TD Expected Sarsa	1,00	TD Minimax QL	TD Sarsa	0,60
MonteCarlo Control	TD Minimax QL	1,00	TD Minimax QL	TD Expected Sarsa	1,00
TD Q-learning	random	0,40			
TD Q-learning	negamax	1,00			
TD Q-learning	MonteCarlo Control	1,00			
TD Q-learning	TD Sarsa	1,00			
TD Q-learning	TD Expected Sarsa	1,00			
TD Q-learning	TD Minimax QL	1,00			

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	TD Minimax QL	2,80
2	MonteCarlo Control	2,80
3	negamax	1,80
4	TD Q-learning	1,00
5	TD Sarsa	0,60
6	TD Expected Sarsa	-2,20
7	random	-6,80

Conclusiones

Vemos que el agente con la estrategia minimax ha mejorado mucho con respecto a los anteriores, llegando a igualar al agente MonteCarlo, que era de los que mejor se comportaba. El cambio en la estrategia al ser jugador 2 (evitar las acciones con q-valor 0 o casi 1), también ha hecho que mejore.

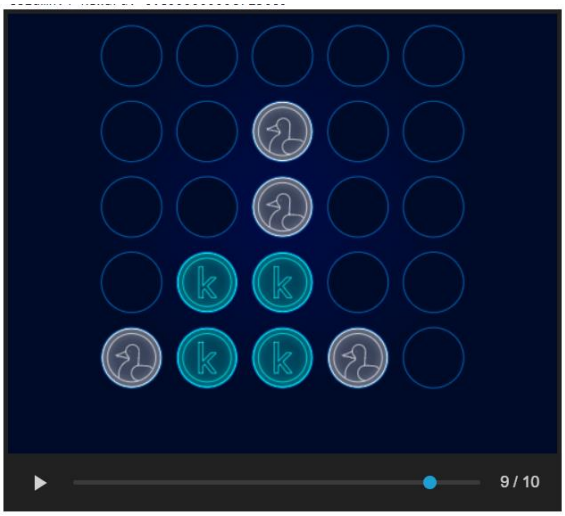
Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que casi siempre se llega a estados (tableros) para los que tiene poca información. En las partidas de entrenamiento el jugador 2 no selecciona las acciones aleatoriamente (no hay mucha exploración).

A pesar de esto, el comportamiento como jugador 2 ha mejorado mucho. Antes, en seguida caía en un estado del que no tenía información, y ahora las partidas duran mucho más y es más difícil de batir.

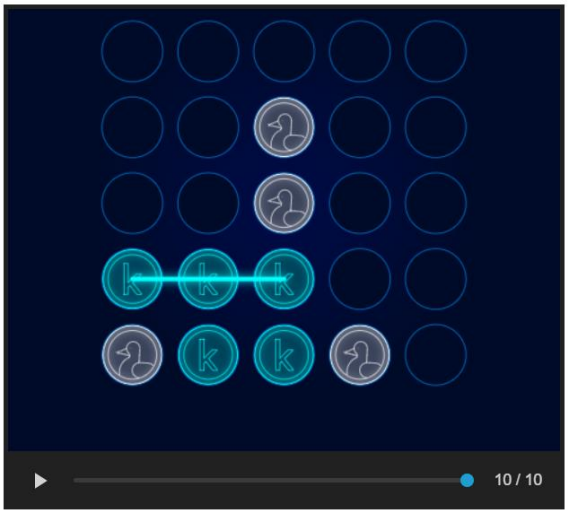
Ejemplos

Ejemplo: TD Minimax QL (azul) vs negamax (gris)

Nuestro agente va agrupando fichas y genera una situación en la que tenemos dos opciones para ganar (columnas 1 y 4)

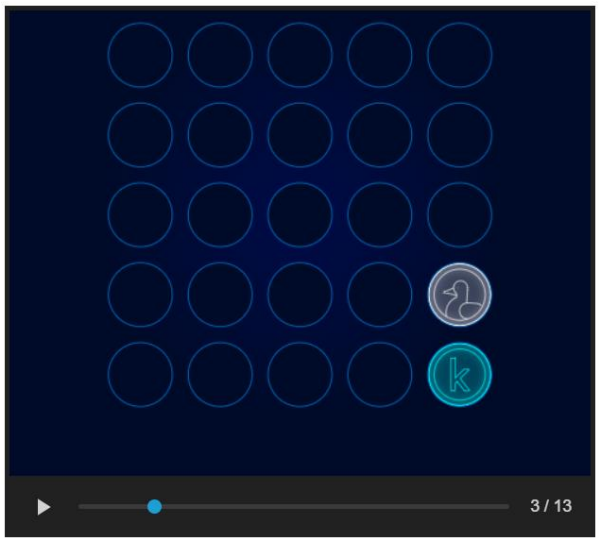


En el siguiente movimiento gana la partida

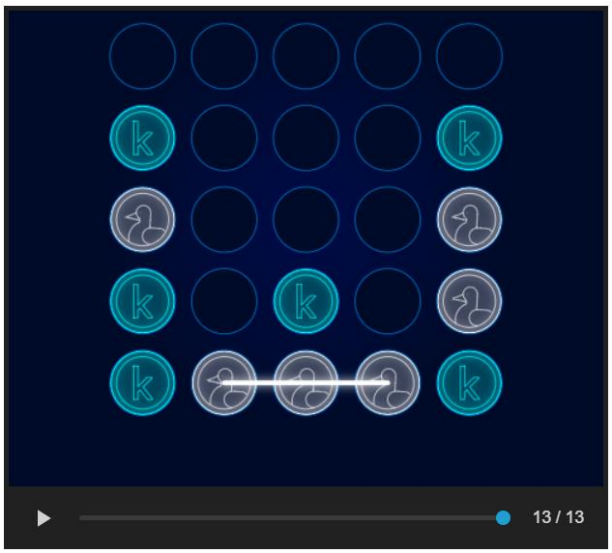


Ejemplo: negamax (azul) vs TD Minimax QL (gris)

El jugador 1 (negamax) empieza la partida colocando la ficha en la columna 5. Nuestro agente tiene q-valores positivos para las fichas colocadas en las columnas 4 y 5, y el resto son ceros. Ahora, al actuar como jugador 2, busca el mínimo Q-valor distinto de 0, por lo que sitúa la ficha en una de esas dos columnas, a pesar de que tengan q-valores positivos.



Al evitar las columnas con q-valor cero, evitamos tableros desconocidos, así que la partida se alarga y nuestro agente acaba venciendo como jugador 2.



2.6. Temporal Difference Double Q-learning

Utilizamos Double Q-learning para aproximar la función Q-valor, y con ella la política óptima. La diferencia con el Q-learning clásico es que en este caso tenemos 2 tablas de Q en vez de una. Durante la fase de entrenamiento se utilizan (aleatoriamente) una para seleccionar la próxima acción y la otra para actualizar el Q-valor. En el Q-learning clásico, al utilizar la misma tabla para esas dos tareas, se tiende a seleccionar y evaluar las acciones con mayor Q-valor, lo que produce una sobreestimación de esas acciones. A su vez, hay menos tendencia a seleccionar y evaluar acciones con Q-valor pequeño, lo que hace que se subestimen.

Esto se intenta solucionar con las dos tablas, para que selección y evaluación se realicen con distintos Q-valores.

Para la política objetivo, lo que se hace es sumar el Q-valor de las dos tablas.

Aproximamos la política objetivo de la siguiente manera:

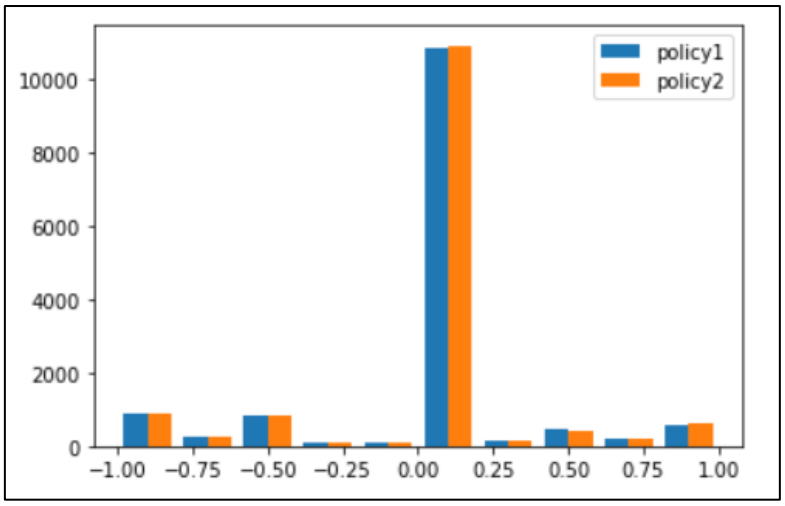
- Generamos N episodios (200.000)
- En cada paso de cada episodio evaluamos el Q-valor de cada par (estado, acción). Hay que tener en cuenta que cada paso (cada llamada a la función step()) en realidad incluye dos pasos (nuestro movimiento y el del oponente), así que hay evaluar el Q-valor de ambos ya que nuestro agente puede ejercer tanto de jugador 1 como de jugador 2
- La recompensa sólo está al final de la partida, así que las recompensas inmediatas son cero hasta el último paso.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mejor valor (greedy) sumando los Q-valores de ambas tablas.
  Para el jugador 2 evitamos los q-valores iguales a cero, ya que implican estados que no se han visto durante el entrenamiento y el agente actúa aleatoriamente en esos casos. También evitamos q-valores casi iguales a uno, ya que es preferible jugar aleatoriamente que hacerlo sabiendo que vas a perder la partida.
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:
  - Usamos la tabla Q para seleccionar y Q' para actualizar
    $$\text{New } Q(s, a) = (1 - \alpha)\, Q(s, a) + \alpha\, [\, R(s, a) + \gamma\, Q'(s', \max_{a'} Q(s', a'))\,]$$
  - Usamos la tabla Q' para seleccionar y Q para actualizar
    $$\text{New } Q'(s, a) = (1 - \alpha)\, Q'(s, a) + \alpha\, [\, R(s, a) + \gamma\, Q(s', \max_{a'} Q'(s', a'))\,]$$
- La actualización del valor de los estados se hace utilizando un parámetro α (learning rate).

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que, dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

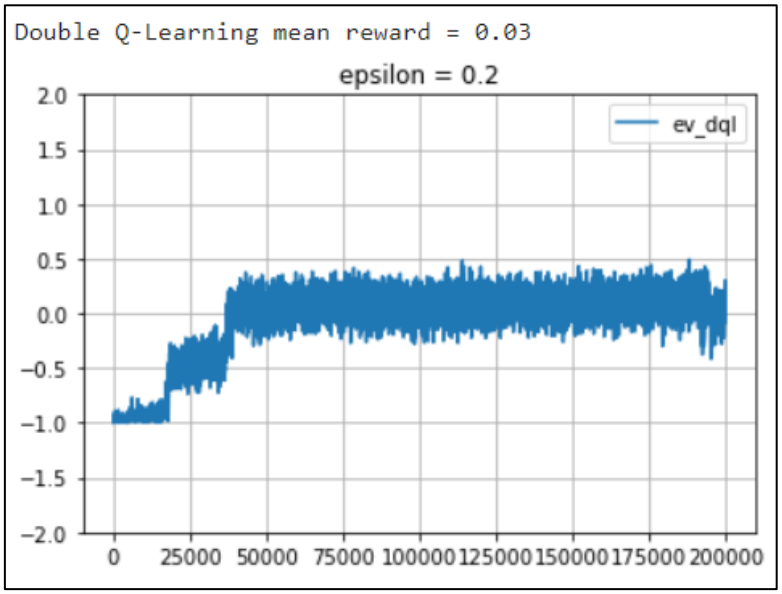
La generación de episodios se hace contra el agente preconfigurado "negamax".

Una vez aproximada la política óptima con la generación de suficientes episodios (200.000) esta es la distribución que obtenemos de los Q-valores de los estados.



Como vemos, en este caso, la mayoría de los valores se agrupan en torno al cero. Al no tener este entorno recompensas intermedias, inicialmente sólo tienen Q-valor distinto de cero los estados-acciones finales. Ese valor se va propagando hacia atrás a medida que se generan episodios, pero es más lento que MonteCarlo, donde se evalúan los episodios completos.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. No es útil verlas directamente, ya que solo hay 1s y -1s. Así que obtenemos la media ponderada de la lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual.



Se puede ver un claro ascenso en las recompensas de los primeros 30.000 episodios, y luego se mantienen estables en torno al cero. Esto puede deberse al bajo valor de épsilon (0,2) que da poco lugar a la exploración.

Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado
random	negamax	-0,80	TD Sarsa	random	0,80
random	MonteCarlo Control	-1,00	TD Sarsa	negamax	0,20
random	TD Q-learning	-0,40	TD Sarsa	MonteCarlo Control	0,80
random	TD Sarsa	-0,60	TD Sarsa	TD Q-learning	0,80
random	TD Expected Sarsa	-0,60	TD Sarsa	TD Expected Sarsa	0,40
random	TD Minimax QL	-0,80	TD Sarsa	TD Minimax QL	0,20
random	TD Double QL	0,40	TD Sarsa	TD Double QL	0,20
negamax	random	1,00	TD Expected Sarsa	random	0,80
negamax	MonteCarlo Control	1,00	TD Expected Sarsa	negamax	0,40
negamax	TD Q-learning	1,00	TD Expected Sarsa	MonteCarlo Control	0,60
negamax	TD Sarsa	1,00	TD Expected Sarsa	TD Q-learning	0,60
negamax	TD Expected Sarsa	1,00	TD Expected Sarsa	TD Sarsa	0,20
negamax	TD Minimax QL	0,40	TD Expected Sarsa	TD Minimax QL	0,40
negamax	TD Double QL	1,00	TD Expected Sarsa	TD Double QL	0,40
MonteCarlo Control	random	1,00	TD Minimax QL	random	1,00
MonteCarlo Control	negamax	1,00	TD Minimax QL	negamax	1,00
MonteCarlo Control	TD Q-learning	1,00	TD Minimax QL	MonteCarlo Control	1,00
MonteCarlo Control	TD Sarsa	1,00	TD Minimax QL	TD Q-learning	1,00
MonteCarlo Control	TD Expected Sarsa	1,00	TD Minimax QL	TD Sarsa	0,80
MonteCarlo Control	TD Minimax QL	0,60	TD Minimax QL	TD Expected Sarsa	0,80
MonteCarlo Control	TD Double QL	1,00	TD Minimax QL	TD Double QL	-0,20
TD Q-learning	random	0,20	TD Double QL	random	0,80
TD Q-learning	negamax	1,00	TD Double QL	negamax	1,00
TD Q-learning	MonteCarlo Control	1,00	TD Double QL	MonteCarlo Control	-0,60
TD Q-learning	TD Sarsa	0,80	TD Double QL	TD Q-learning	0,20
TD Q-learning	TD Expected Sarsa	1,00	TD Double QL	TD Sarsa	0,60
TD Q-learning	TD Minimax QL	1,00	TD Double QL	TD Expected Sarsa	0,80
TD Q-learning	TD Double QL	0,40	TD Double QL	TD Minimax QL	1,00

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	MonteCarlo Control	3,80
2	negamax	2,60
3	TD Minimax QL	1,80
4	TD Q-learning	1,20

5	TD Double QL	1,00
6	TD Sarsa	-0,40
7	TD Expected Sarsa	-1,00
8	random	-9,00

Conclusiones

Vemos que el agente con la estrategia Double Q-learning no ha mejorado al propio Q-learning, así que no parece que el mal funcionamiento del Q-learning se debiera a que los Q-valores estuvieran estimados por encima o por debajo de su valor "real". De hecho, ambas tablas de Q-valores acaban siendo casi iguales.

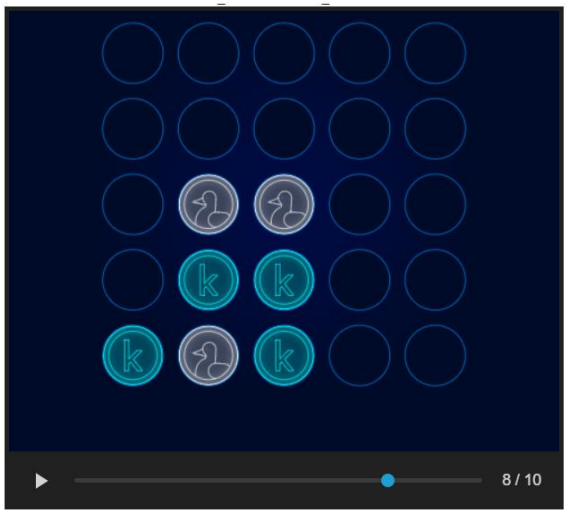
Al haber generado todos los episodios como jugador 1, la política se comporta mejor en esos casos. En las partidas en las que actúa como jugador 2 pierde bastante más, debido a la desventaja que supone esto en el juego, y a que casi siempre se llega a estados (tableros) para los que tiene poca información. En las partidas de entrenamiento el jugador 2 no selecciona las acciones aleatoriamente (no hay mucha exploración).

A pesar de esto, el comportamiento como jugador 2 ha mejorado mucho. Antes, en seguida caía en un estado del que no tenía información, y ahora las partidas duran mucho más y es más difícil de batir.

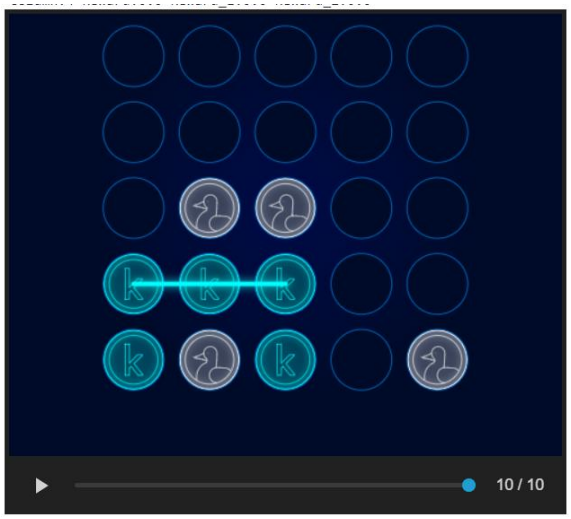
Ejemplos

Ejemplo: TD Double QL (azul) vs negamax (gris)

Nuestro agente va agrupando fichas y genera una situación en la que tenemos dos opciones para ganar (columna 1)

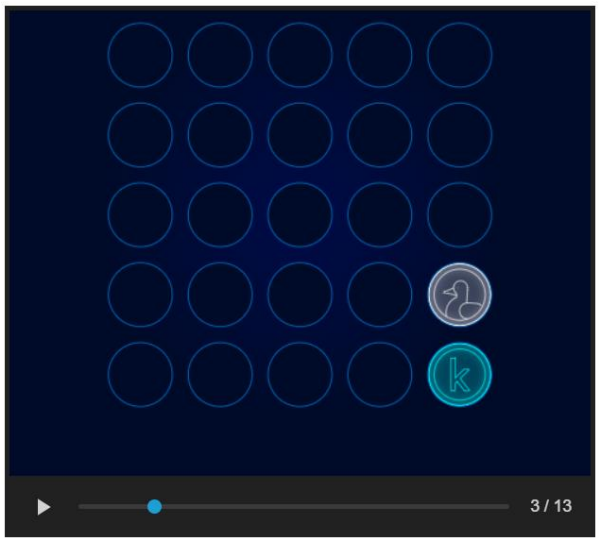


En el siguiente movimiento gana la partida



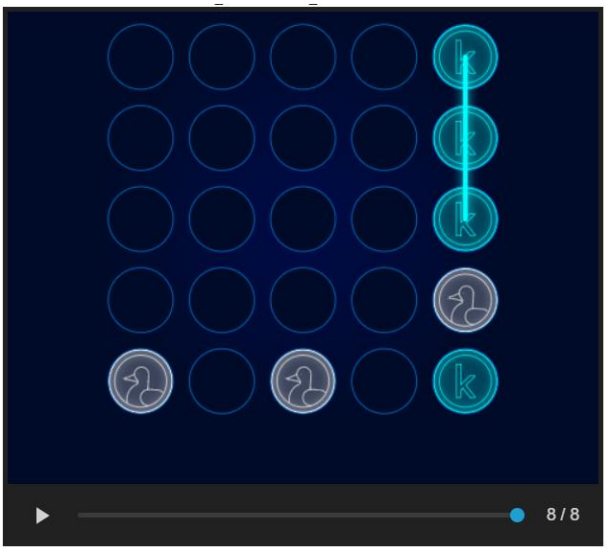
Ejemplo: negamax (azul) vs TD Double QL (gris)

El jugador 1 (negamax) empieza la partida colocando la ficha en la columna 5. Nuestro agente tiene q-valores positivos para las fichas colocadas en las columnas 4 y 5, y el resto son ceros. Ahora, al actuar como jugador 2, busca el mínimo Q-valor distinto de 0, por lo que sitúa la ficha en una de esas dos columnas, a pesar de que tengan q-valores positivos.



Al evitar las columnas con q-valor cero, intentamos evitar tableros desconocidos, pero en este caso llegamos rápido a un tablero sin Q-valores. Esto se debe a que hemos utilizado el mismo número de iteraciones de entrenamiento que en el resto de agentes, pero en este caso para generar dos tablas de Q-valores. Así que las recompensas, que solo se dan al final de la partida, tardan el "doble" en llegar a los estados iniciales.

Vemos que rápidamente el jugador 1 gana la partida



2.7. Temporal Difference Deep Q-learning

Utilizamos Deep Q-learning para aproximar la función Q-valor, y con ella la política óptima. La diferencia con el Q-learning clásico es que en este caso se aproxima la función con una red neuronal, en vez de hacerlo con una tabla. Para este tipo de entornos, en los que hay un número muy elevado de estados es una buena aproximación, ya que se tiene un q-valor incluso para los estados que no se han visto durante el entrenamiento.

Por otro lado, es más complicado entrenar este tipo de redes, ya que la recompensa solo se tiene al final de la partida. Una posibilidad es añadir recompensas en los pasos intermedios (reward shaping) propagando la recompensa final por todos los pasos. A medida que se aleja del estado final, la recompensa va siendo menor.

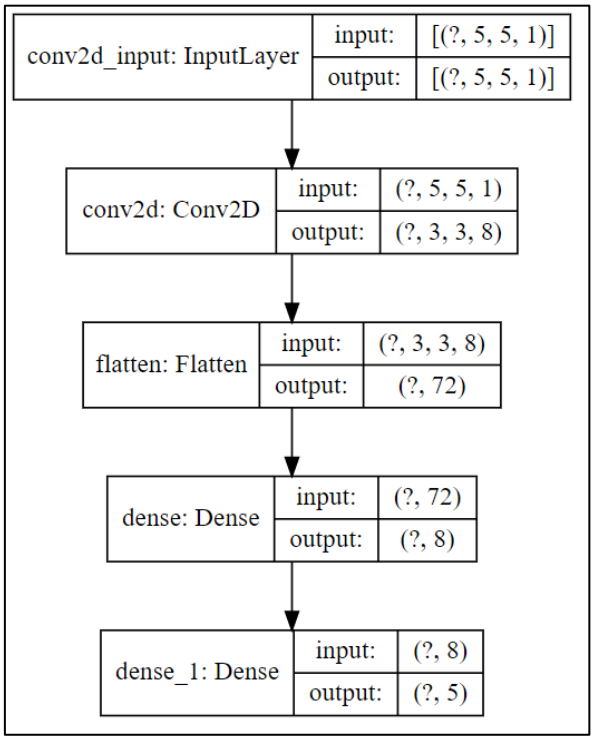
Debido a este descenso en la recompensa a medida que nos alejamos del final, empezamos entrenando con estados finales y poco a poco vamos añadiendo los pasos anteriores. Lo que buscamos con esto es facilitar el entrenamiento.

Dado que este método utiliza una red neuronal para aproximar el q-valor, se utilizan las interacciones con el entorno para generar los datos de entrenamiento (experience replay). De todas esas interacciones se seleccionan unas cuantas aleatoriamente, para intentar conseguir que los datos sean independientes e idénticamente distribuidos.

Que los datos sean idénticamente distribuidos es más complicado. Si entrenamos contra negamax perderemos muchas más partidas de las que ganamos, así que habrá muchos datos con recompensas negativas. Si lo hacemos contra random, acabaremos ganando más partidas de las que perdamos y pasará lo contrario. Lo que hacemos es transformar los datos antes de almacenarlos, de modo que todos se vean desde la perspectiva del jugador 1. Así pasamos de tener todos los rewards positivos o negativos, a tener la mitad positivos y la otra mitad negativos.

De este modo también evitamos tener que pasarle a la red la información del usuario que va a hacer el movimiento (un mismo tablero y acción tiene q-valores distintos dependiendo de quién haga el movimiento), ya que solo se entrena con el jugador 1.

Para la red neuronal, se utiliza una capa convolucional a la entrada, de modo que sea capaz de analizar el tablero y encontrar patrones.



Como funciones de activación hemos puesto la función relu en las capas intermedias (para añadir la no linealidad y facilitar el entrenamiento) y la función lineal en la capa de salida.

No hemos añadido capas de pooling, ya que queremos que conserve, en la medida de lo posible, la situación del patrón encontrado en el tablero.

Transformamos el tablero que se utiliza en el entorno de una lista a una matriz con dimensionalidad filas x columnas x 1. Inicialmente tiene valores 0 (vacío), 1 (ficha del jugador 1) o 2 (ficha del jugador 2), así que sustituimos las fichas del jugador 2 para que tengan valor -1. De este modo las fichas de cada jugador están a la misma "distancia" de estar vacías.

Aproximamos la política objetivo de la siguiente manera:

- Generamos N episodios (200.000)
- Una vez finalizado cada episodio, almacenamos los n últimos pasos en el buffer (experience replay) y entrenamos el modelo.
- La recompensa sólo está al final de la partida, así que hemos generado recompensas intermedias para facilitar el entrenamiento. También se ha multiplicado por 2 la recompensa final (-2, 2) para que sea mayor que la suma de recompensas parciales.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mayor valor (greedy).
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:

  $$\text{New } Q(s, a) = R(s, a) + \gamma \max Q'(s', a')$$

  Con este nuevo Q-valor se entrena a la red.

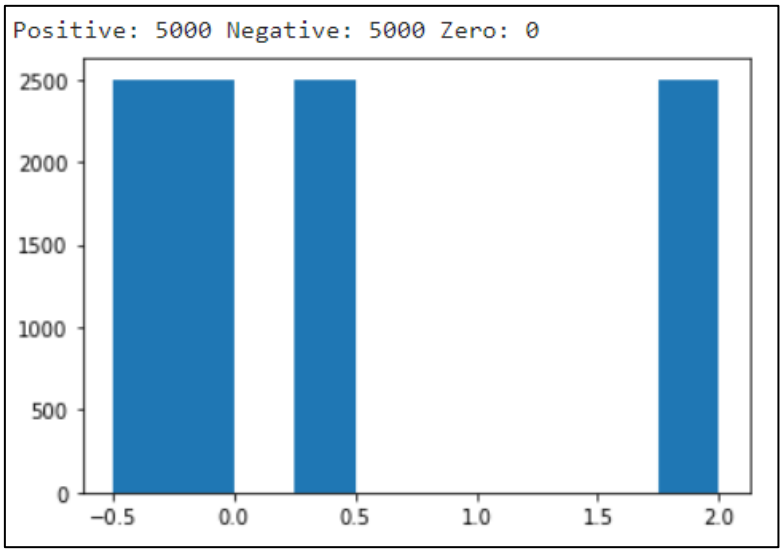
- Cuando el agente actúa como jugador 2, lo que hacemos es transformar el tablero cambiando las fichas de un jugador por otro, y evaluar el Q-valor como si fuésemos el jugador 1.

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que, dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

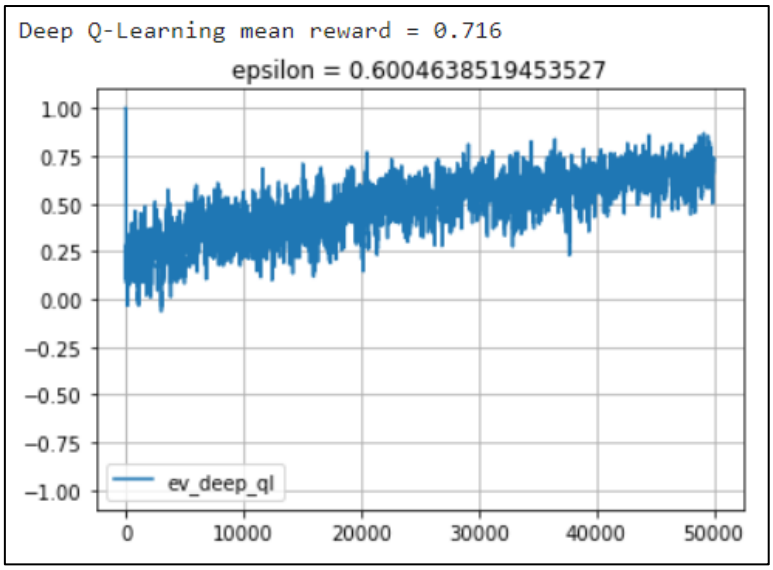
La generación de episodios se hace contra el agente preconfigurado "random", ya que contra "negamax" no hemos conseguido que mejore.

Una vez terminado el entrenamiento esta es la distribución que obtenemos de los Q-valores en el buffer (experience replay), que tiene un tamaño de 10.000.



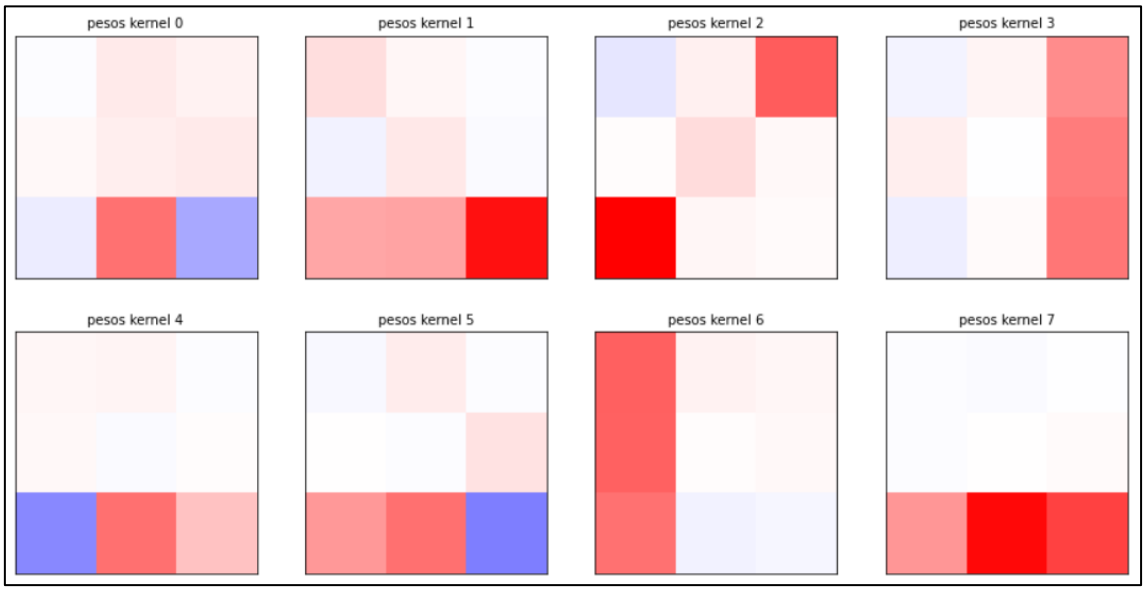
Como vemos, los datos están bien distribuidos en cuanto a valores positivos/negativos. La mayoría de partidas las gana el jugador 1, y por eso las recompensas más altas (en valor absoluto) son positivas. Las recompensas negativas son debidas a pasos previos al final, realizados por el jugador 2, pero convertidos en pasos del jugador 1.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. Obtenemos la media ponderada de la lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual. De ese modo, se ve mejor la evolución.



Se puede ver un aumento progresivo de las recompensas, a medida que el modelo va aprendiendo y que va disminuyendo épsilon (cada vez hay menos exploración y se elige la mejor acción).

Mirando los pesos de los filtros de la capa convolucional, se ve que ha empezado a "aprender" a buscar fichas en horizontal (kernels 1 y 7), vertical (kernels 3 y 6) y en diagonal (kernel 2)



Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, con el resto de agentes utilizando la función evaluate(), incluida en el entorno. Esta función simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado
random	negamax	-0,80	TD Expected Sarsa	random	0,60
random	MonteCarlo Control	-0,60	TD Expected Sarsa	negamax	0,80
random	TD Q-learning	-0,60	TD Expected Sarsa	MonteCarlo Control	0,40
random	TD Sarsa	-0,60	TD Expected Sarsa	TD Q-learning	0,40
random	TD Expected Sarsa	0,00	TD Expected Sarsa	TD Sarsa	0,40
random	TD Minimax QL	-1,00	TD Expected Sarsa	TD Minimax QL	0,80
random	TD Double QL	0,00	TD Expected Sarsa	TD Double QL	0,80
random	TD Deep QL	-0,40	TD Expected Sarsa	TD Deep QL	-0,80
negamax	random	1,00	TD Minimax QL	random	0,40
negamax	MonteCarlo Control	0,80	TD Minimax QL	negamax	1,00
negamax	TD Q-learning	1,00	TD Minimax QL	MonteCarlo Control	1,00
negamax	TD Sarsa	1,00	TD Minimax QL	TD Q-learning	1,00
negamax	TD Expected Sarsa	1,00	TD Minimax QL	TD Sarsa	1,00
negamax	TD Minimax QL	0,00	TD Minimax QL	TD Expected Sarsa	1,00
negamax	TD Double QL	1,00	TD Minimax QL	TD Double QL	0,40
negamax	TD Deep QL	1,00	TD Minimax QL	TD Deep QL	-0,40
MonteCarlo Control	random	0,60	TD Double QL	random	0,40
MonteCarlo Control	negamax	1,00	TD Double QL	negamax	0,80
MonteCarlo Control	TD Q-learning	1,00	TD Double QL	MonteCarlo Control	-1,00
MonteCarlo Control	TD Sarsa	0,40	TD Double QL	TD Q-learning	0,40
MonteCarlo Control	TD Expected Sarsa	1,00	TD Double QL	TD Sarsa	0,60
MonteCarlo Control	TD Minimax QL	1,00	TD Double QL	TD Expected Sarsa	1,00
MonteCarlo Control	TD Double QL	0,80	TD Double QL	TD Minimax QL	1,00
MonteCarlo Control	TD Deep QL	-1,00	TD Double QL	TD Deep QL	-0,40
TD Q-learning	random	0,80	TD Deep QL	random	0,80
TD Q-learning	negamax	1,00	TD Deep QL	negamax	0,20
TD Q-learning	MonteCarlo Control	1,00	TD Deep QL	MonteCarlo Control	-1,00
TD Q-learning	TD Sarsa	0,80	TD Deep QL	TD Q-learning	-0,40
TD Q-learning	TD Expected Sarsa	0,80	TD Deep QL	TD Sarsa	1,00
TD Q-learning	TD Minimax QL	1,00	TD Deep QL	TD Expected Sarsa	-0,40
TD Q-learning	TD Double QL	0,60	TD Deep QL	TD Minimax QL	-1,00
TD Q-learning	TD Deep QL	-0,40	TD Deep QL	TD Double QL	1,00
TD Sarsa	random	0,80			
TD Sarsa	negamax	0,60			
TD Sarsa	MonteCarlo Control	0,80			
TD Sarsa	TD Q-learning	0,40			
TD Sarsa	TD Expected Sarsa	0,60			
TD Sarsa	TD Minimax QL	0,60			
TD Sarsa	TD Double QL	0,00			
TD Sarsa	TD Deep QL	-0,40			

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	MonteCarlo Control	3,40
2	TD Deep QL	3,00
3	TD Minimax QL	3,00
4	TD Q-learning	2,40
5	negamax	2,20
6	TD Sarsa	-1,20
7	TD Expected Sarsa	-1,60
8	TD Double QL	-1,80
9	random	-9,40

Si nos quedamos solo con las partidas entre los agentes que han mostrado mejor comportamiento a lo largo de las pruebas (negamax, MonteCarlo, Minimax QL y Deep Q-Learning), el resultado es el siguiente:

Posición	Agente	Puntuación
1	TD Minimax QL	1,60
2	MonteCarlo Control	0,20
3	negamax	-0,40
4	TD Deep QL	-1,40

Conclusiones

Al aproximar la función Q-valor con una red neuronal, tenemos siempre un valor, aunque no se haya visto ese estado en el entrenamiento. En esos casos los modelos anteriores actuaban de forma aleatoria.

Por otro lado, los modelos anteriores evaluaban mejor los estados-acciones que sí habían visto en el entrenamiento. Es decir, que ahora tenemos un Q-valor para cualquier estado y acción, pero es menos preciso. Esto es suficiente para ganar a los agentes con peor resultado (y con ello obtener buenas puntuaciones), pero en los enfrentamientos directos con los mejores agentes no obtiene resultados tan buenos.

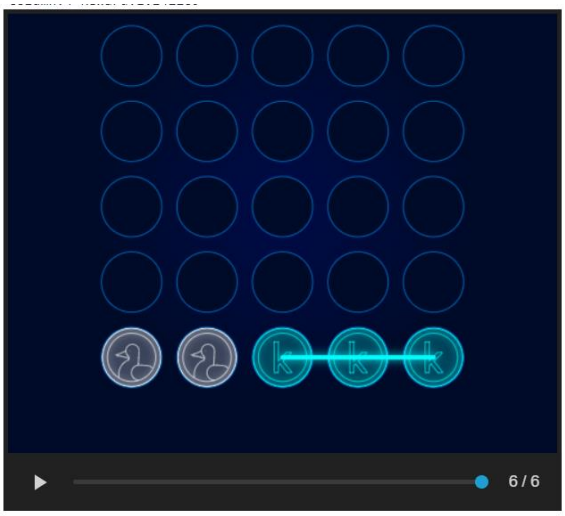
Hemos intentado mejorar el entrenamiento aplicando recompensas intermedias y entrenando el modelo solo con pasos finales, pero no hemos conseguido grandes mejoras.

En este caso la política se comporta igual actuando como jugador 1 o como jugador 2, pero como jugador 2 pierde más, debido a la desventaja que supone esto en el juego.

Ejemplos

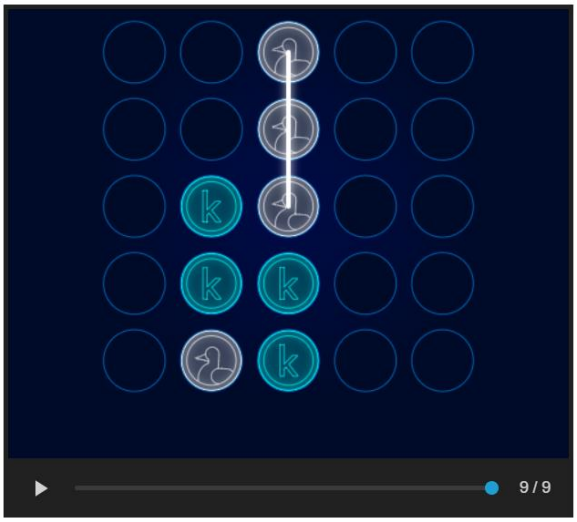
Ejemplo: TD Deep QL (azul) vs random (gris)

Nuestro agente gana al jugador random con cierta facilidad. En seguida junta las fichas necesarias.



Ejemplo: TD Deep QL (azul) vs negamax (gris)

El jugador 2 (negamax) gana la partida con cierta facilidad. Nuestro agente intenta ir acumulando fichas, pero no evita que lo haga el contrincante.



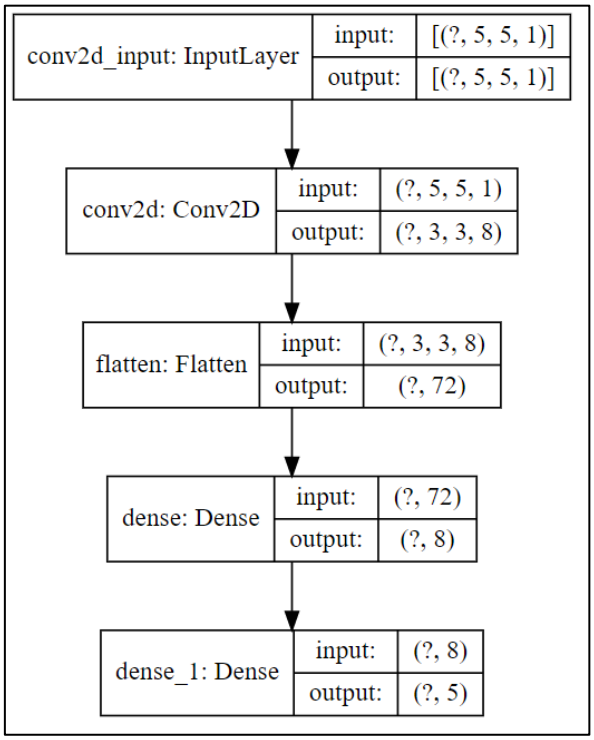
2.8. Temporal Difference Double Deep Q-learning

Utilizamos Double Deep Q-learning para aproximar la función Q-valor, y con ella la política óptima. La diferencia con Deep Q-learning es que en este caso se utilizan dos redes neuronales. Una para seleccionar la próxima acción (online) y otra para evaluarla a la hora de actualizar el Q-valor (target). Con ese valor actualizado se entrena la red online, y cada cierto tiempo se actualizan los pesos de la red target con los de la red online.

Al igual que en Double Q-learning, con esta aproximación intentamos evitar sobreestimar o subestimar determinadas acciones.

Hemos utilizado las mismas transformaciones y el mismo tipo de entrenamiento que con Deep Q-learning.

Para las redes neuronales, hemos utilizado la misma arquitectura que en Deep Q-learning.



Como funciones de activación hemos puesto la función relu en las capas intermedias (para añadir la no linealidad y facilitar el entrenamiento) y la función lineal en la capa de salida.

No hemos añadido capas de pooling, ya que queremos que conserve, en la medida de lo posible, la situación del patrón encontrado en el tablero.

Transformamos el tablero que se utiliza en el entorno de una lista a una matriz con dimensionalidad filas x columnas x 1. Inicialmente tiene valores 0 (vacío), 1 (ficha del jugador 1) o 2 (ficha del jugador 2), así que sustituimos las fichas del jugador 2 para que tengan valor -1. De este modo las fichas de cada jugador están a la misma "distancia" de estar vacías.

Aproximamos la política objetivo de la siguiente manera:

- Generamos N episodios (200.000)
- Una vez finalizado cada episodio, almacenamos los n últimos pasos en el buffer (experience replay) y entrenamos el modelo.
- La recompensa sólo está al final de la partida, así que hemos generado recompensas intermedias para facilitar el entrenamiento. También se ha multiplicado por 2 la recompensa final (-2, 2) para que sea mayor que la suma de recompensas parciales.
- En la política objetivo siempre se va a seleccionar la acción que nos lleve al estado con mayor valor (greedy).
- TD(0). Sólo tenemos en cuenta el estado siguiente.
- El Q-valor de cada (estado, acción) se va actualizando de la siguiente manera:

  $$\text{New } Q(s, a) = R(s, a) + \gamma\, Q'_{\theta_{target}}(s', \text{argmax } Q'_{\theta}(s', a'))$$

Con este nuevo Q-valor se entrena a la red online.

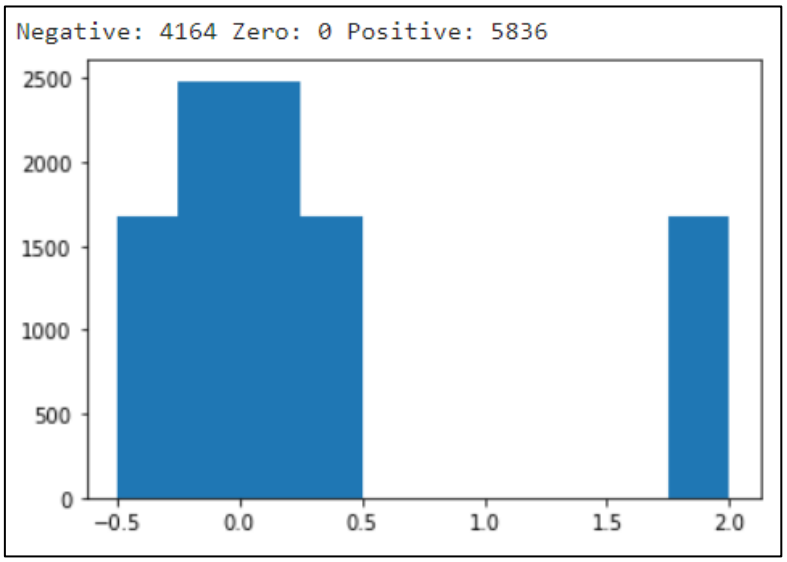
- Cuando el agente actúa como jugador 2, lo que hacemos es transformar el tablero cambiando las fichas de un jugador por otro, y evaluar el Q-valor como si fuésemos el jugador 1.

Tomamos como estado el tablero de juego, así que dentro de un mismo episodio no puede haber dos estados iguales. La acción es la columna del tablero en la que se mete la ficha.

Utilizamos un método off-policy, ya que aprendemos la política objetivo π utilizando la política μ en la que, dependiendo de un número aleatorio, se selecciona la acción con mayor Q-valor o una acción aleatoria (ε-greedy). Esta política va evolucionando de modo que al principio se eligen más acciones aleatorias, y al final se seleccionan más las acciones con mayor Q-valor.

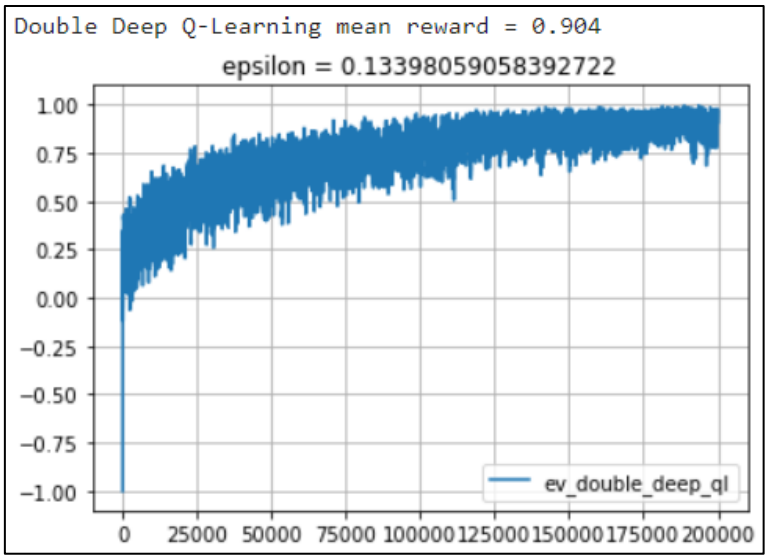
La generación de episodios se hace contra el agente preconfigurado "random", ya que contra "negamax" no hemos conseguido que mejore.

Una vez terminado el entrenamiento esta es la distribución que obtenemos de los Q-valores en el buffer (experience replay), que tiene un tamaño de 10.000.



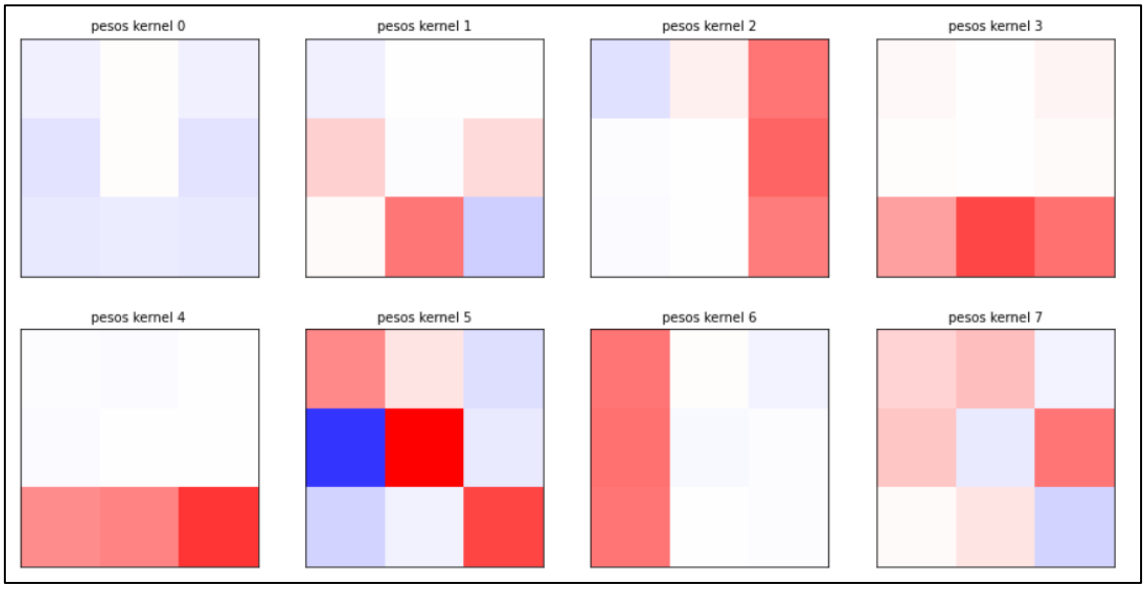
Como vemos, los datos están relativamente bien distribuidos en cuanto a valores positivos/negativos. La mayoría de partidas las gana el jugador 1, y por eso las recompensas más altas (en valor absoluto) son positivas. Las recompensas negativas son debidas a pasos previos al final, realizados por el jugador 2, pero convertidos en pasos del jugador 1.

Vemos cómo han ido evolucionando las recompensas durante el entrenamiento. Obtenemos la media ponderada de la lista de recompensas donde el elemento actual tiene el mayor peso, y los anteriores tienen pesos que tienden a 0 a medida que se alejan del actual. De ese modo, se ve mejor la evolución.



Se puede ver un aumento progresivo de las recompensas, a medida que el modelo va aprendiendo y que va disminuyendo épsilon (cada vez hay menos exploración y se elige la mejor acción).

Mirando los pesos de los filtros de la capa convolucional, se ve que ha empezado a "aprender" a buscar fichas en horizontal (kernels 3 y 4), vertical (kernels 2 y 6) y en diagonal (kernel 5)



Evaluación

Enfrentamos al agente que utiliza la política que hemos aprendido, contra los mejor agentes implementados. Para ello usamos la función evaluate(), incluida en el entorno, que simula n partidas y devuelve una lista de resultados (1 para la victoria y -1 para la derrota).

Vamos a simular 10 partidas y a obtener la media de los resultados (1 si ha ganado todas, -1 si ha perdido todas). Cada agente jugará como jugador 1 y jugador 2

Jugador 1	Jugador 2	Resultado	Jugador 1	Jugador 2	Resultado

negamax	MonteCarlo Control	1,00	TD Deep QL	negamax	0,40
negamax	TD Minimax QL	0,20	TD Deep QL	MonteCarlo Control	-1,00
negamax	TD Deep QL	1,00	TD Deep QL	TD Minimax QL	-1,00
negamax	TD Double Deep QL	1,00	TD Deep QL	TD Double Deep QL	1,00
MonteCarlo Control	negamax	1,00	TD Double Deep QL	negamax	-1,00
MonteCarlo Control	TD Minimax QL	1,00	TD Double Deep QL	MonteCarlo Control	-1,00
MonteCarlo Control	TD Deep QL	-1,00	TD Double Deep QL	TD Minimax QL	-1,00
MonteCarlo Control	TD Double Deep QL	1,00	TD Double Deep QL	TD Deep QL	1,00
TD Minimax QL	negamax	1,00			
TD Minimax QL	MonteCarlo Control	1,00			
TD Minimax QL	TD Deep QL	0,20			
TD Minimax QL	TD Double Deep QL	0,00			

Con el resultado de esas partidas realizamos una clasificación por agentes

Posición	Agente	Puntuación
1	TD Minimax QL	3,00
2	MonteCarlo Control	2,00
3	negamax	1,80
4	TD Deep QL	-1,80
5	TD Double Deep QL	-5,80

Conclusiones

Al aproximar la función Q-valor con una red neuronal, tenemos siempre un valor, aunque no se haya visto ese estado en el entrenamiento. En esos casos los modelos anteriores actuaban de forma aleatoria.

Por otro lado, los modelos anteriores evaluaban mejor los estados-acciones que sí habían visto en el entrenamiento. Es decir, que ahora tenemos un Q-valor para cualquier estado y acción, pero es menos preciso. Esto no es suficiente al enfrentarse a los mejores agentes.

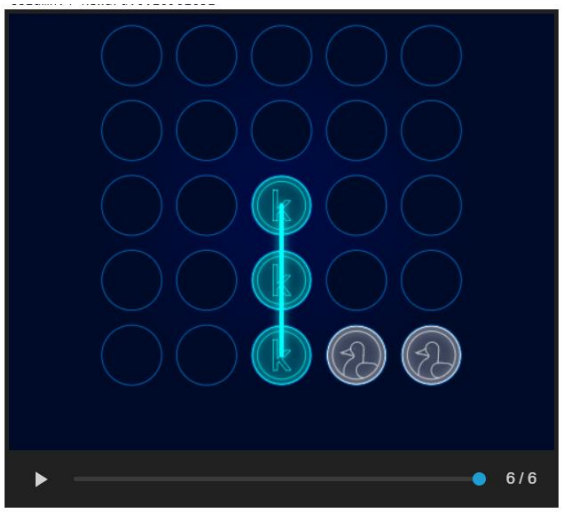
Hemos intentado mejorar el entrenamiento aplicando recompensas intermedias y entrenando el modelo solo con pasos finales, pero no hemos conseguido grandes mejoras.

En este caso la política se comporta igual actuando como jugador 1 o como jugador 2, pero como jugador 2 pierde más, debido a la desventaja que supone esto en el juego.

Ejemplos

Ejemplo: TD Double Deep QL (azul) vs random (gris)

Nuestro agente gana al jugador random con cierta facilidad. En seguida junta las fichas necesarias.



Ejemplo: TD Deep QL (azul) vs negamax (gris)

El jugador 2 (negamax) gana la partida con cierta facilidad. Nuestro agente intenta evita alguno de los intentos del oponente de conseguir la victoria, pero finalmente pierde.