Métodos de Aprendizaje Automático
Práctico 1

Bruno Garate & Nicolás Izquierdo & Guillermo Siriani

1. Ejercicio 2 - Damas

1.1. Implementación

La implementación del ejercicio se separó en dos partes. Por un lado se implementaron los componentes requeridos para modelar el juego de damas, la validación de los movimientos y su ejecución. Para validar este aspecto se implementó además una interfaz gráfica en la cual ejecutar las jugadas.

Por otro lado, se implementaron los componentes requeridos para el aprendizaje por LMS. Se implentó un ejecutor del algoritmo (*learner*), al cual se le pasa una representación, que calcula un vector representación dado un tablero. Además se le pasan dos maquinas, una, la maquina A, sobre la que se aplica el algoritmo de aprendizaje, y una maquina B, contra la que compita la primera.

A nivel de archivos:

- **ej*.py.** Ejemplos de la ejecución del algoritmo de aprendizaje.
- **ej-gui.txt.** Ejemplo del comando a correr para jugar contra la maquina en la gui
- **gui.py.** Interfaz gráfica para la validación de los algoritmos del core del juego. Permite la carga de contrincantes.
- **damas.py.** Contiene las clases que modelan e implementan el juego de las damas.
- **learn.py.** Contiene las clases utilizadas para la ejecución del algoritmo de aprendizaje. Permite la salida a consola de información de *debugging* así como a un archivo en formato CSV los pesos.
- **representacion.py.** Contiene una clase que implementa una de las representaciones usadas para la ejecución del aprendizaje LMS.
- **maquina.py.** Máquina que elije según los valores de sus *weights* la próxima jugada a ejecutar.
- **maquinaAzar.py.** Máquina que elije una jugada al azar entre aquellas válidas. Se utiliza para validar el aprendizaje.

Se proveyeron opciones para configurar las salidas, por ejemplo salida en consola o a un archivo formato CSV.

```
learner = Learner(maquinaEstatica, maquinaEstaticaC,
        0.0001, TipoAprendizaje.APRENDEN_AMBAS_MAQUINAS,
        "pesos.csv")
```

Además se incorporó la capacidad de cargar jugadores maquina en la interfaz gráfica, pasando por parámetro la ubicación de un archivo json que describe los contrincantes.

```
{
    "negras": {
        "tipo":"azar"
    },
    "blancas": {
        "tipo":"comun",
            "pesos" : [26.187072695411917, -32.84680592423919,
                       13.213702407264911, 57.504267170163494,
                       21.574640421320762, -23.00357084783828,
                       -1.193679277554918, 1.9201039975176828,
                       20.288830091323383, -16.56871602519243,
                       -80.97633510212901]
    }
}
```

Figura 1: Ejemplo de comida simple

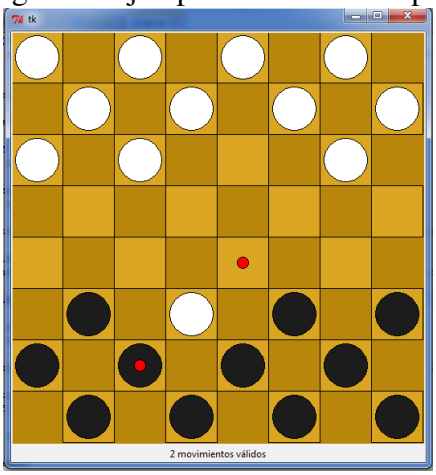


Figura 2: Ejemplo de múltiples comidas posibles

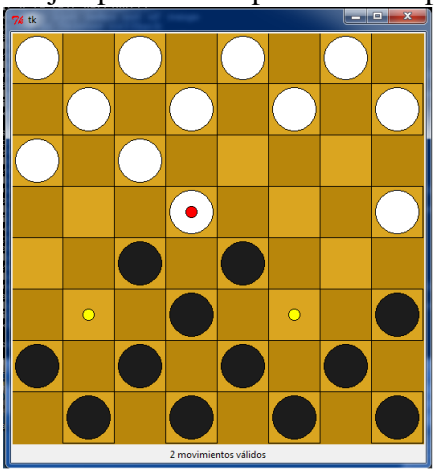
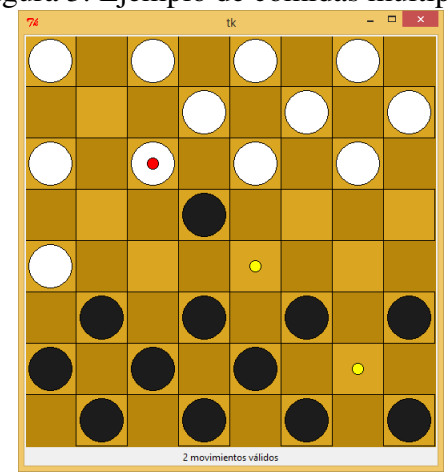


Figura 3: Ejemplo de comidas múltiples



1.2. Consideraciones prácticas

Existe un aspecto fundamental al que nos enfrentamos en la ejecución del algoritmo de Least Mean Squares, una caso particular del Stochastic Gradient Descent. Este es la convergencia.

Este algoritmo es sensible a la dimensionalidad del vector de representación a partir del cual se calculan los pesos. A mayor es el número de *features* consideradas, mayor es el número de iteraciones del algoritmo requeridas para alcanzar cierto nivel de convergencia.

Otro aspecto al que es sensible este algoritmo, es al *factor de aprendizaje*. A mayor es este valor, más afecta cada iteración el valor de los pesos, y más rápido es el aprendizaje, pero se corre el riesgo de que el algoritmo falle al converger.

Por otro lado, a menor valor, más lenta es la convergencia, pero se disminuye la posibilidad de *overshoot* del valor real, y la posibilidad de que el valor no converga, o incluso diverga.

Otra característica a tomar en cuenta es la posibilidad de encontrar mínimos locales, dependientes del *factor de aprendizaje* y los *pesos iniciales*.

En vista de los aspectos anteriores fueron tomados en el diseño y ejecución del algoritmo:

- Selección de las *features* a considerar: **cantidad de fichas blancas y negras, cantidad de fichas blancas y negras seguras (sobre los límites del tablero), distancia agregada a la linea de coronación de las fichas blancas y negras, cantidad de movimientos no comida de blancas y negras y cantidad de piezas blancas y negras amenazadas**, dando un total de 10 *features* consideradas.

- Búsqueda de un factor de aprendizaje apropiado. Para esto se redujo a partir de una semilla, hasta encontrar el primer valor que no divergiese.

- Búsqueda de una cantidad suficiente de iteraciones N tales que dado un cierto ϵ cualquier par de *weigths* $w_{n_0}, w_{n_1} / n_0, n_1 \geq N, |w_{n_0} - w_{n_1}| < \epsilon$

1.3. Ejecución

A continuación, se resumen a modo de ejemplo, los resultados para un cierto número de iteraciones, dadas ejecuciones, los pesos resultantes de la aplicación del algoritmo. Se utilizaron las siguientes referencias:

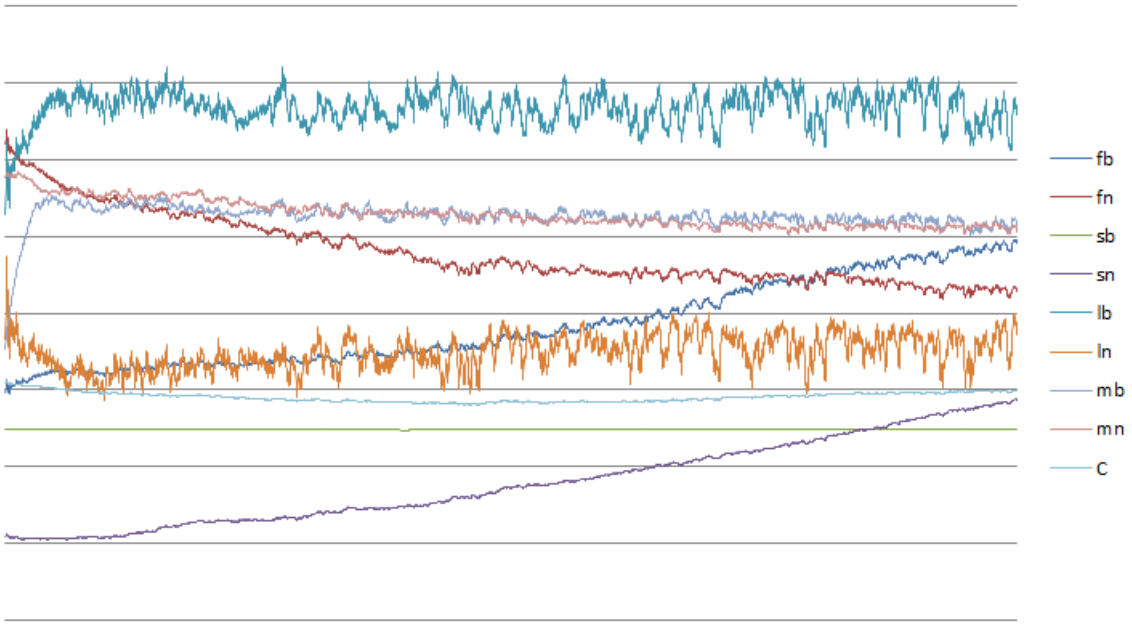
- i - Número de iteraciones
- f_b - Número de fichas blancas
- f_n - Número de fichas negras
- s_b - Número de fichas blancas seguras.
- s_n - Número de fichas negras seguras.
- l_b - Distancia agregada de las fichas blancas a su línea de coronación
- l_n - Distancia agregada de las fichas negras a su línea de coronación
- m_b - Cantidad de movimientos no comida de las fichas blancas, suponiendo su turno
- m_n - Cantidad de movimientos no comida de las fichas negras, suponiendo su turno
- c_b - Cantidad de fichas blancas diferentes amenazadas
- c_n - Cantidad de fichas negras diferentes amenazadas
- C - Constante aplicada

Para un primer entrenamiento, se tomo el valor 100 como el caso de victoria de las blancas, -100 como el de derrota, y 0 el de empate.
Aquellos elementos que presentan los menores valores de contribución relativo, inciden en una menor cantidad que aquellos que presentan un mayor valor. Teóricamente, el valor límite de aquellos valores que no contribuyen al resultado de la partida, debería ser cero. Por lo tanto, aquellos que presentaron un valor de contribución cercano a cero, podrían ser removidos de la representación final. Estos mismo valores, además, toman un mayor tiempo en converger ya que presentan ajustes menores en el algoritmo.

Además, se tomó en cuenta la contribución de cada peso w_i a la evaluación del resultado. Esto se tomó como $p_i = \frac{t_i}{\sum_{i=0}^{n} t_i}$ con $t_i = \sum_{i=0}^{n} |w_{i,n}|$ siendo $w_{i,n}$ el valor del i-esimo peso, en la n-esima iteración del algoritmo y f_i el valor de la función de *fitness* en esta misma iteración.

Es interesante ver la evolución de los diferentes pesos.
Existen diferentes comportamientos observados:
- Algunos valores, entre estos, aquellos que presentan el menor impacto relativo en la función de *fitness*, no parecen converger, oscilando en una banda de valores de distinta amplitud.
- Otros parecen converger muy lentamente a un valor límite.
- A su vez, otros, particularmente aquellos que contribuyen de forma más significativa (mayor peso relativo) convergen rápidamente. En algunos casos, esto produce un efecto de látigo (*overshooting*) alrededor del valor final.

Figura 4: Evolución de algunos pesos seleccionados a lo largo de cinco mil iteraciones entre las máquinas D y Dc



Se consideraron dos tipos de evaluaciones para las posiciones finales del tablero: Estática y Dinámica. En la primera, el valor de retorno de la función de evaluación depende únicamente del signo de la diferencia entre fichas blancas y negras, mientras que en la segunda depende del valor de esta diferencia.

$$f_{estatica} = \begin{cases} 100 & f_b > f_n \\ 0 & f_b = f_n \\ -100 & f_b < f_n \end{cases} = 100.sgn(f_b - f_n)$$

$$f_{dinamica} = 100.(f_b - f_n)$$

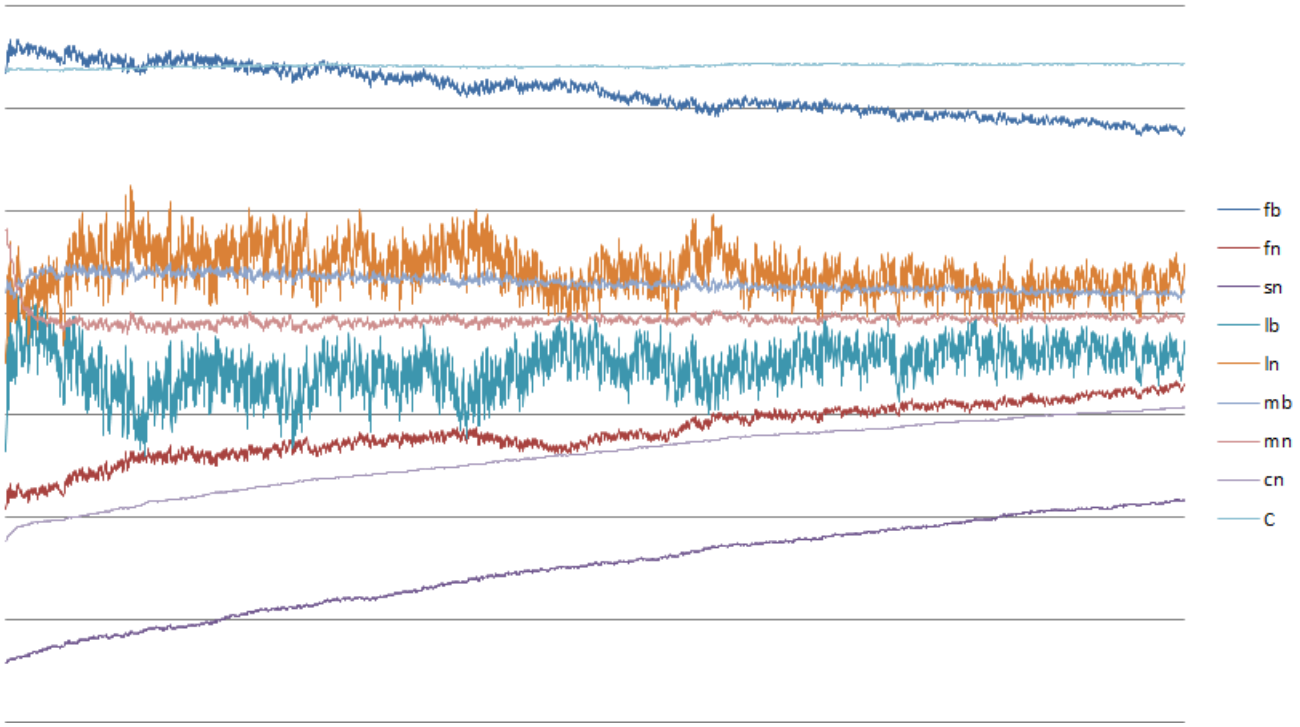
La evaluación dinámica, ofrece un mayor incentivo por jugadas que conducen a una mayor diferencia de piezas. Asumimos que estas jugadas tienden a presentar una mejor situación a la larga en la partida por lo que este tipo de evaluación es más agresiva a la hora de converger.

Aunque no fueron estudiadas en detalle, es interesante observar no sólo la diferencia de piezas en las partidas sino además la cantidad de jugadas en las que se llega a la victoria.

Tomamos E y Ec como máquinas estáticas, D y Dc como maquinas dinámicas y A como una máquina que juega al azar. Con tres mil iteraciones de aprendizaje de las maquinas E y D contra Ec y Dc respectivamente. Tres mil iteraciones para los enfrentamientos cruzados entre las máquinas E y D, D y E, en el cual solo aprende la primer máquina, y finalmente tres mil iteraciones de aprendizaje por parte de las maquinas D y E contra la máquina A. Una vez acabado el entrenamiento se decidió enfrentar las maquinas E, D y A sin ningún tipo de aprendizaje

	E vs Ec	D vs Dc	E vs D	D vs E	D vs A	E vs A	E vs D
Victorias	47.86 %	46.96 %	27.66 %	52.26 %	79.93 %	75.16 %	51.4 %
Derrotas	47.23 %	47.03 %	67.46 %	40.63 %	14.56 %	21.86 %	1.5 %
Empates	4.9 %	6.0 %	4.86 %	7.1 %	5.5 %	2.96 %	47.1 %

Figura 5: Evolución de algunos pesos seleccionados a lo largo de cinco mil iteraciones entre las máquinas E y Ec



De forma muy interesante, la estática parece que finalmente alcanzó una performance ligeramente superior a la dinámica, pero aparentemente convergió mas lentamente, como parece evidenciar los resultados de entrenamientod E vs D y luego D vs E.

Por un lado creemos que una estrategia más agresiva produjo un ajuste más rápido de las máquinas con valoración dinámica de las partidas finales. Por el otro lado, los valores finales de esta estrategia parecen diverger, tal vez explicando un peor performace frente a la evaluación estática.

Una posible explicación a esta divergencia es la agresividad, dado un factor elevado de multiplicación de la diferencia. Una valor inferior para este factor o un factor de aprendizaje menor. Valores elevados del factor aplicado sobre la diferencia varían la presión del algoritmo sobre esta misma. Incluso sería posible emplear una función $sgn(f_b - f_n).[\lambda.(f_b - f_n) + c]$ que es una aproximación más flexible a la valoración del tablero final.

En la siguiente tabla se muestra el coeficiente y el peso relativo de cada feature para la máquina dinámica antes de diverger (D), la máquina estática (E) y la máquina dinámica luego de diverger (D').

	f_b	f_n	s_b	s_n	l_b	l_n	m_b	m_n	c_b	c_n	C
D	32.3	-14.3	-78.9	89.7	30.6	-22.9	-3.01	7.6	18.5	-15.1	-44
	12.3 %	2.85 %	0.65 %	3.65 %	36.1 %	42.5 %	0.64 %	0.61 %	0.18 %	0.27 %	
E	33.7	-2.25	86.6	-24.37	38.5	-29.6	0.8	-0.42	8.9	-27.5	-53
	6.5 %	0.9 %	0.7 %	1.98 %	41.1 %	46.9 %	0.21 %	1.04 %	0.06 %	0.38	
D'	13113	-11515	2452	-5827	46057	-30550	-2769	5272	1664	-1926	41
	3.06 %	3.18 %	0.01 %	0.18 %	45.51 %	46.75 %	0.63 %	0.6 %	0.01 %	0.01	

1.4. Mejoras

Sobre este algoritmo se plantean un número de mejoras a fin de mejorar su performance:

- Aplicación del algoritmo min-max o de evaluaciones multijugada. El aplicar la función de evaluación sobre un número de jugadas en el futuro reporta una sensibilidad mucho mayor a la situación real y futura del tablero.
- Consideración de distribuciones y posiciones especificas de las piezas como distribuciones número de pieza en posiciones bajas, medias y superiores, posiciones sobre las diagonales, agujeros, piezas solitarias y agrupadas (*pattern*) así como disposiciones triangulares, puentes y circulares (*layout*).
- Heuristicas no lineales. Algunos papers reconocen el aporte de dividir el estudio del juego en tres o más fases, por ejemplo, según el número de fichas disponibles. Para cada una de las fases se calcula una función de *fitness* diferente.
- Emplear base de datos de aperturas y fin de juego. De esta forma se incorpora un número de tableros finales e iniciales conocidas junto a sus valoraciones, mejorando el ajuste, y reduciendo el espacio de soluciones a explorar.
- Aplicación de factores de aprendizaje variables, que se reduzcan en el tiempo, así como aplicaciones del *momento* u otras optimizaciones
- Evaluaciones no estáticas. Además de considerar la evaluación del tablero de forma estática, es interesante considerar la distancia que se encuentra un tablero evaluado del comienzo y del final de la partida para el aprendizaje, así como otros factores globales.

2. Ejercicio 9 - Aprendizaje Conceptual

2.1. Parte a

Para este nuevo espacio, la hipótesis más específica es :

$$S =< \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing >$$

Se observa que en este nuevo espacio, esta hipótesis es la única que es falsa para toda instancia del problema, a diferencia del espacio visto en el teórico en el cual había múltiples hipótesis equivalentes.

Por otro lado, en este espacio hay múltiples hipótesis que sirven como la mas general, ya que cualquier hipótesis con el símbolo '?' en al menos uno de sus componentes, será verdadera para toda instancia del problema. Siguiendo la notación vista, decidimos tomar como la hipótesis mas general a:

$$G =<?, ?, ?, ?, ?, ? >$$

Interesa destacar que en este nuevo espacio cambia la forma de escribir hipótesis mas específicas. Por ejemplo, sean las hipótesis:

$$H_1 =< \varnothing, \varnothing >$$
$$H_2 =< A, \varnothing >$$
$$H_3 =< A, B >$$

Entonces:

$$H_1 \leq H_2$$
$$H_2 \leq H_3$$

2.2. Parte b

Partimos de la hipótesis mas específica y la mas general vistas en la parte a:

$$S_0 = \{< \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing >\}$$
$$G_0 = \{<?, ?, ?, ?, ?, ? >\}$$

El primer ejemplo a considerar es positivo y es:
$< Soleado, Templado, Normal, Fuerte, Templada, SinCambios >$

Siguiendo el algoritmo, transformamos S_0 en un conjunto de hipótesis específicas consistentes con el ejemplo.

$$S_1 = \left\{ \begin{array}{c} < Soleado, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing > \\ < \varnothing, Templado, \varnothing, \varnothing, \varnothing, \varnothing > \\ < \varnothing, \varnothing, Normal, \varnothing, \varnothing, \varnothing > \\ < \varnothing, \varnothing, \varnothing, Fuerte, \varnothing, \varnothing > \\ < \varnothing, \varnothing, \varnothing, \varnothing, Templada, \varnothing > \\ < \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, SinCambios > \end{array} \right\}$$

$$G_1 = \{<?, ?, ?, ?, ?, ? >\}$$

El siguiente ejemplo es negativo y es:

$$< Lluvioso, Frío, Alta, Fuerte, Templada, Cambiante >$$

Removemos las hipótesis incosistentes de S y modificamos G para que deje de reconocer este ejemplo. La modificación de G se hace de modo que se siga reconociendo el resto del espacio, lo que hace aparecer valores que no habíamos visto hasta ahora porque estaban cubiertos por ?:

$$S_2 = \left\{ \begin{array}{c} < Soleado, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing > \\ < \varnothing, Templado, \varnothing, \varnothing, \varnothing, \varnothing > \\ < \varnothing, \varnothing, Normal, \varnothing, \varnothing, \varnothing > \\ < \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, SinCambios > \end{array} \right\}$$

$$G_2 = \left\{ \begin{array}{c} < Soleado, Templado, Normal, Suave, Fría, SinCambios > \\ < Nublado, Templado, Normal, Suave, Fría, SinCambios > \end{array} \right\}$$

El último ejemplo es positivo y es:

$$< Soleado, Templado, Alta, Fuerte, Fría, Cambiante >$$

No hay ninguna hipótesis de G inconsistente con el ejemplo, por lo que no se modifica. Hay dos hipótesis de S inconsistentes con el ejemplo, pero no existe para ninguna de las dos una hipótesis mas general que a su vez sea mas específica que alguna de G, por lo cual solamente se eliminan. Con el cual obtenemos:

$$S_3 = \left\{ \begin{array}{c} < Soleado, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing > \\ < \varnothing, Templado, \varnothing, \varnothing, \varnothing, \varnothing > \end{array} \right\}$$

$$G_3 = \left\{ \begin{array}{c} < Soleado, Templado, Normal, Suave, Fría, SinCambios > \\ < Nublado, Templado, Normal, Suave, Fría, SinCambios > \end{array} \right\}$$

Referencias

[1] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, http://sebastianruder.com/optimizing-gradient-descent/

[2] Magdalena Kusiak, Karol Waledzik, and Jacek Mandziuk, *Evolutionary approach to the game of checkers*, Faculty of Mathematics and Information Science, Warsaw University of Technology