Machine Learning II
Redes Neuronales Artificiales (RNAs)

U-TAD

¿Qué es una Red Neuronal Artificial (RNA)?

- Tipo de algoritmo de ML inspirado en las redes neuronales biológicas (surgen como consecuencia de pensar cómo funciona nuestro cerebro, pero la implementación no tiene absolutamente nada que ver).
- Componente principal del Deep Learning.
- Se introducen por primera vez en 1943 por el neuropsicólogo Warren McCulloch y el matemático Walter Pitts (pero sobre los 60's se deja de investigar porque ese potencial que se presuponía que tenían, se intuye que no es para tanto. Y en los 80's se intentan recuperar).
- Alrededor de 1990, pierden interés a favor de técnicas de ML como el SVM.
- En 2012, resurgen con más fuerza que nunca cuando Geoffrey Hinton gana el reto **ImageNet** con una Red Neuronal Convolucional (consistía en clasificar un conjunto de imágenes). A partir de aquí, el uso aumenta, también gracias al Big Data.

¿Por qué resurgen las RNA?

- En la actualidad, existe una gran cantidad de datos disponibles (big data)
- En las últimas décadas, el poder computacional se ha multiplicado (nvidia, aws), permitiendo la ejecución de algoritmos costosos y complejos en periodos de tiempo razonables.
- Los algoritmos en los que se basan las RNAs han mejorado.
- Muchas limitaciones que se habían intuido de manera teórica resultaron no cumplirse en la práctica.

A priori, cuando se estudiaron estos tipos de algoritmos, surgían algunas restricciones a la hora de usarlos, como que en el proceso de entrenamiento se presuponía que estos algoritmos iban a quedarse en mínimos locales y, por lo tanto, no iban a alcanzar el mínimo global de la función de error (no daría buenas predicciones). Pero, en la práctica, resultó que raras veces se atascaba en un mínimo local y la mayoría de las veces era capaz de converger en el mínimo global.

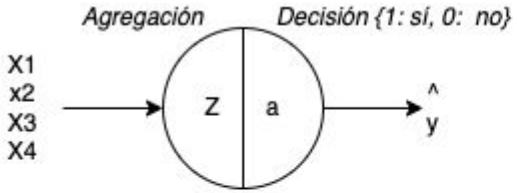
La primera neurona artificial (1943)

Neurona de McCulloch y Pitts (Neurona M-P)

- Se corresponde con la primera neurona artificial de la historia
- Se caracteriza porque recibe uno o más valores binarios {0, 1} y retorna otro valor binario {0, 1}
- Activa su salida cuando más de un número determinado *n* de valores de entrada se encuentran activos (=1). Es decir, si de las características de entrada *n* son igual a 1, se activará, y si no, entonces devolverá 0.
- Debe establecerse manualmente el número de valores de entrada *n* que deben estar activos, a este valor se le denomina *threshold* (umbral, límite)

Por ejemplo, si threshold=5, debería haber al menos 5 características de entrada activas (=1), para que la salida sea 1 (la neurona se activa).

Funcionamiento de la Neurona M-P



$$Z(X1, X2, X3, X4) = Z(X) = \sum_{i=1}^{n} Xi = X1 + X2 + X3 + X4$$

n = #características = 4

$a(Z(x)) = 1$ si $Z(X) >= \theta$ ó 0 si $Z(X) < \theta$ (Threshold, por ejemplo 2, que se define manualmente)

y -> ¿Voy al cine hoy?

X1 -> ¿es fin de semana?

X2 -> ¿tengo tiempo librer?

X3 -> ¿está el cine cerrado?

X4 -> ¿estrenan película hoy?

Características de entrada y limitaciones:

¿Qué ocurre si X3 es 1? que el resto ya no importa y la neurona debería devolver 0, sí o sí.

Para evitar este problema, se introducen dos tipos de inputs: inhibidores (X3) y excitadores (X1, X2, X4)

Si X3 está activada (=1), La neurono siempre devolverá 0 independientemente del resto de características.

Nos permite configurar puertas lógicas (AND, OR, …): Para dos entradas, con threshold 1, sería OR, y con threshold 2, sería AND.

Limitaciones de la Neurona M-P

- Recibe únicamente valores binarios {0, 1}, y en la mayoría de los problemas reales se dispondrá de valores de otros tipos.
- Requiere la selección del *threshold* de manera manual.
- Todas las entradas son iguales. No se le puede asignar un mayor peso a una de las entradas.
- No son capaces de resolver problemas que no sea linealmente separables (lo veremos a continuación en un ejemplo práctico), por ejemplo, la operación XOR (problema no linealmente separable)

Caso Práctico: Implementando la Neurona M-P

Veremos cómo evoluciona hasta lo que hoy conocemos como el Perceptrón.

Implementando la **Neurona M-P.ipynb**

En este caso, como ya no se usa, no viene implementado en las librerías más conocidas de ML, por lo que solo en este caso, lo vamos a implementar.

Caso Práctico: Diagnóstico con la Neurona M-P

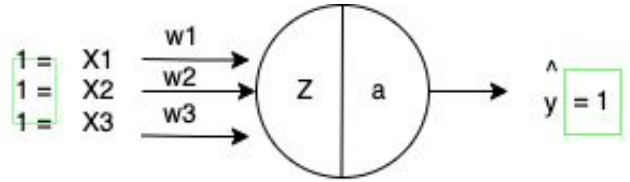
Diagnóstico de cáncer de mama con el Perceptrón.ipynb

El Perceptrón

De M-P al Perceptrón. Este es unos de los algoritmos más conocidos dentro de la disciplina del Deep Learning, o de la IA en general, y sienta las bases de muchas de las arquitecturas de redes neuronales artificiales en la actualidad.

- Propuesto por Frank Rosenblatt en 1958 (M-P fue en 1943)
- En 1969 fue refinado y analizado en detalle por Marvin Minsky y Seymour Papert (muy similar a las que se utilizan a día de hoy)
- Mejora la neurona M-P añadiendo el concepto de "**peso**" numérico a las entradas y **planteando un mecanismo para ajustarlos** (aprende)
- No recibe únicamente valores de entrada binarios, permite valores de entrada reales
- Se basa en un tipo de neurona artificial conocida como **Threshold Logic Unit** (**TLU**). El perceptrón puede ser una o varias TLUs.
- La TLU computa una **suma parametrizada** de las entradas

Neurona M-P vs Perceptrón (TLU, en concreto)



X1 -> ¿Fin de Semana?
X2 -> ¿Estrenan película?
X3 -> ¿Tengo tiempo libre?

$\theta = 2$

M-P

$Z(X1, X2, X3) = Z(X) = \sum Xi = X1 + X2 + X3$

$a(Z(x)) = 1$ si $Z(X) >= \theta$ ó 0 si $Z(X) < \theta$

TLU

$Z(X1, X2, X3) = Z(X) = \sum Xi*wi = X1*w1 + X2*w2 + X3*w3$

$a(Z(x)) = 1$ si $Z(X) \ \theta$ ó 0 si $Z(X) \ \theta$

Imaginemos que X1, para nosotros, es más importante
Este concepto de peso, lo vamos a denotar como w
De modo que ahora puedo decir que w1 = 2, w2 = 1 y w3 = 1
Ahora Z será una suma parametrizada en base a los pesos

Con X1 = 1, X2 = 0 y X3 = 0
M-P devolvería 0, pero TLU:
z(1,0,0) = 1*2 + 0*1 + 0*1 = 2
a(Z(X)) = 1 ya que Z(X) >=2

La diferencia principal es que los pesos no se van a seleccionar de manera manual. Es decir, utilizaremos un algoritmo que va a seleccionar estos pesos de manera automática en base a experiencia pasada:

X1	X2	X3	y
1	0	0	1
0	1	0	0

Perceptrón y eliminación del threshold

Vamos a hacer que el threshold también se aprenda con otro algoritmo en base a esa experiencia pasada.

Se realiza cambiando el Threshold y tratándolo como un peso más, como una importancia más, para una característica que siempre va a ser igual a uno.

Fundamentalmente, se crea una nueva característica de entrada X0, que siempre va a ser igual a 1, y que va a tener asociado como peso el Threshold.

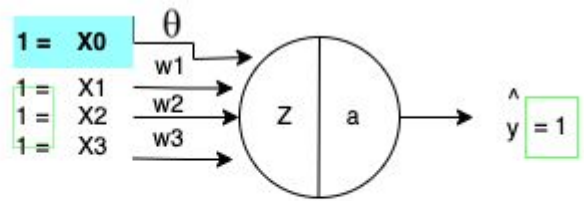
De manera, que ese mismo algoritmo que utilizábamos para ajustar el valor de la importancia, de los pesos w1, w1 y w3 de manera automática, también sirva para seleccionar ese Threshold automáticamente en base a la experiencia pasada.

Perceptrón y eliminación del threshold

Por tanto, la función Z, cambia un poco:

Z(X) = X0*w0 + X1*w1 + …

Siendo X0 = 0 y w0 = θ

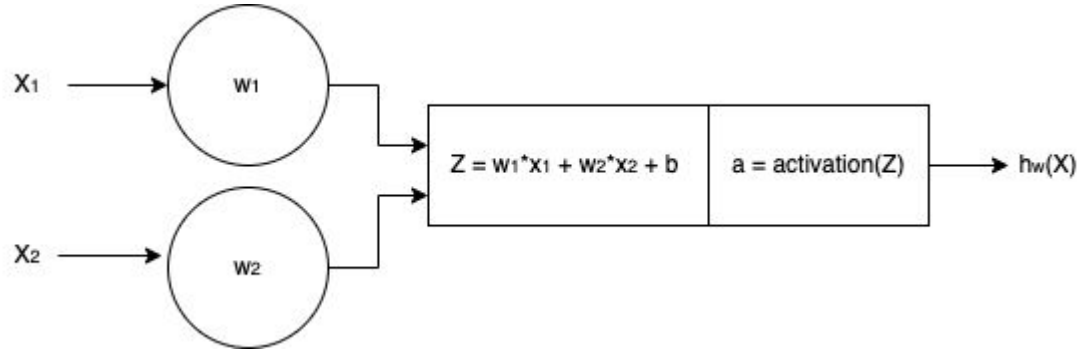


$Z(X1, X2, X3) = Z(X) = \sum Xi*wi = \mathbf{X0*w0} + X1*w1 + X2*w2 + X3*w3$

$a(Z(x)) = 1$ si $Z(X) >= \theta$ ó 0 si $Z(X) < \theta$

Threshold Logic Unit (TLU)

- La TLU computa una **suma parametrizada** de las entradas
- Después, aplica una **función de activación** sobre la suma calculada



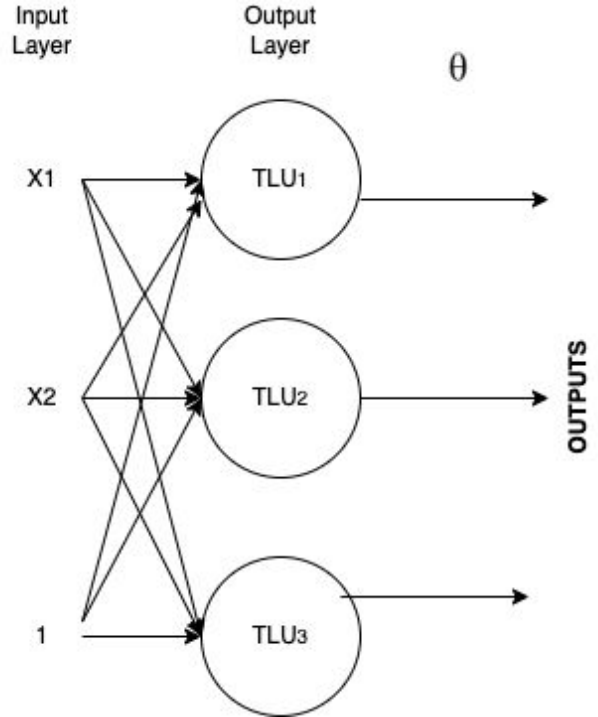
Siendo *b* el término bias que veíamos anteriormente como $X_0 * \theta$

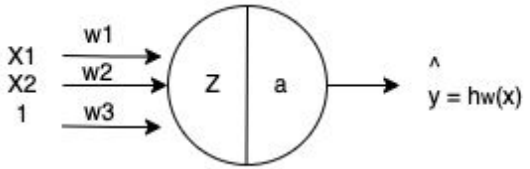
Como X_0 es siempre 1, el Threshold θ en TLU se representa como *b*

Veremos más adelante cómo se incorporan con el Perceptrón nuevas funciones de activación que permiten tomar algunas decisiones un poco más complejas.

Notación y funcionamiento del Perceptrón

- El **perceptrón** se corresponde con una arquitectura compuesta por **una única capa de TLUs.**
- Permite la clasificación de instancias en diferentes clases binarias de manera simultánea





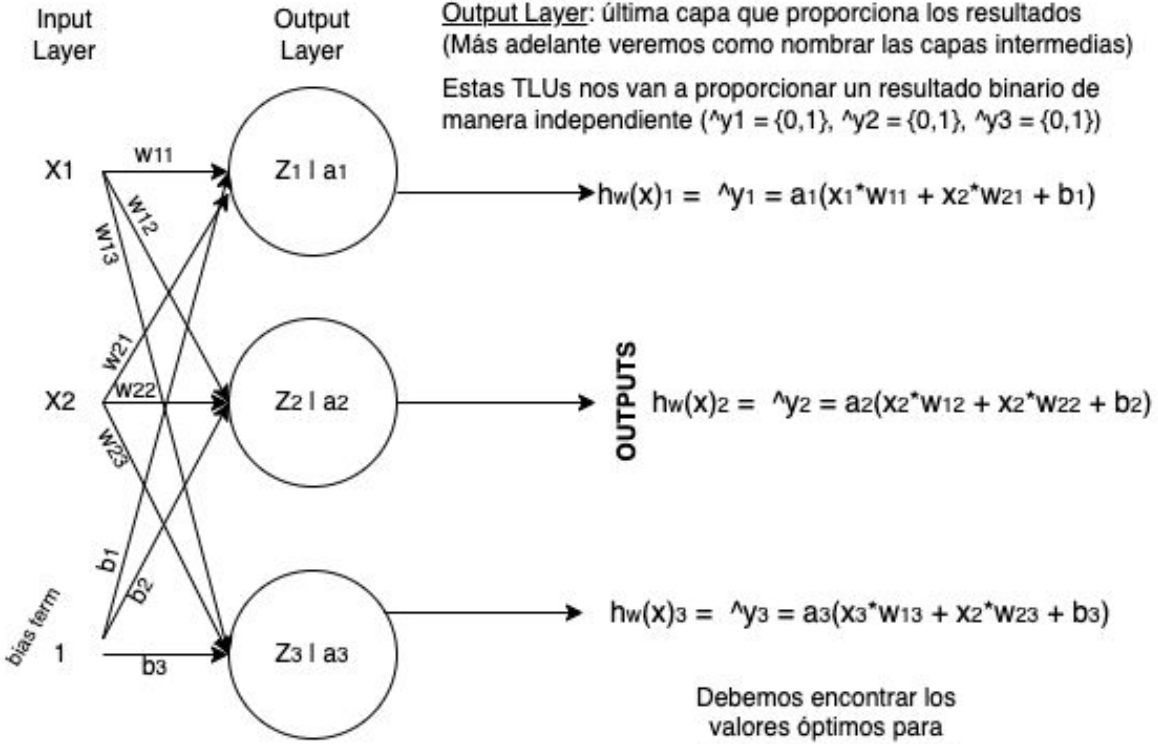
$$Z(X) = w1*X1 + w2*X2 + b$$

$$hw(X) = a(Z(X)) = a(w1*X1 + w2*X2 + b)$$

Los pesos, en este caso w1, w2 y b, se denominan parámetros del modelo. Y llamamos modelo a estas fórmulas una vez ajustados esos pesos o parámetros. Lo que quiere decir que el resultado del algoritmo de este perceptrón estaría parametrizado por estas tres variables w1, w2 y b

Notación y funcionamiento del Perceptrón

Cuando tenemos varias TLUs:



Input Layer: capa de entrada
Output Layer: última capa que proporciona los resultados
(Más adelante veremos como nombrar las capas intermedias)

Estas TLUs nos van a proporcionar un resultado binario de manera independiente (^y1 = {0,1}, ^y2 = {0,1}, ^y3 = {0,1})

$h_w(x)_1 = \hat{y}_1 = a_1(x_1 * w_{11} + x_2 * w_{21} + b_1)$

$h_w(x)_2 = \hat{y}_2 = a_2(x_2 * w_{12} + x_2 * w_{22} + b_2)$

$h_w(x)_3 = \hat{y}_3 = a_3(x_3 * w_{13} + x_2 * w_{23} + b_3)$

Debemos encontrar los valores óptimos para cada uno de los pesos

Cada una de las características de entrada tendrá asociado un peso por cada una de las TLUs:
(w11, w12, w13, w21, w22, w23, b1, b2, b3)

Notación y funcionamiento del Perceptrón

Si quiero construir un filtro de spam que me clasifique emails como legítimos o spam utilizando este algoritmo.

Necesitaré un conjunto de datos de entrada y sus etiquetas. Por ejemplo, si quiero extraer dos características de los emails, como podría ser el número de faltas de ortografía y el número de tags html

	Input features		*Label*
Train Dataset	# faltas ort. (x1)	# tags html (x2)	¿Spam? (y)
email 1	0	0	0
email 2	5	10	1
email 3	0	2	0
email m	…	…	…

Notación y funcionamiento del Perceptrón

El algoritmo ajustará los pesos en base a los datos proporcionados, y una vez ajustados al valor óptimo, se podrá tomar un nuevo email, hacer la extracción únicamente de estas dos características (faltas de ortografía y número de tags html), y el perceptrón haría la predicción de *y*.

Puede ser una predicción binaria o, en el caso del algoritmo visto previamente (con 3 TLUs), cada una de las TLUs nos haría una predicción de 0 ó 1.

Por ejemplo, podríamos tener la salida: $\hat{y}_1 = 0$, $\hat{y}_2 = 1$, $\hat{y}_3 = 0$

Pudiendo aplicar diferentes sistemas para llegar a una conclusión final, por ejemplo, un sistema de clasificación por voto (decidir por el número que más se repita), en este caso $\hat{y} = 0$

Notación y funcionamiento del Perceptrón

Otra cosa interesante teniendo varias TLUs en la output layer, es que nos permite realizar clasificación multiclase en lugar de clasificación binaria.

¿En qué se basa la clasificación multiclase?
Si queremos resolver un problema en el que en lugar de tener únicamente dos categorías de salida, como en este caso, spam o legítimo, tenemos tres o más categorías de salida. Entonces, podría utilizar este Perceptrón para que realice una clasificación entre las tres o más categorías.

¿Cómo? Codificando estas categorías, por ejemplo, de la siguiente forma:
Categoría 1 = [1, 0, 0]
Categoría 2 = [0, 1, 0]
Categoría 3 = [0, 0, 1]

¿Cómo resuelve mi algoritmo? Si predice $\hat{y}_1 = 0$, $\hat{y}_2 = 0$ e $\hat{y}_3 = 1$, lo trato como un vector, en este caso como [0, 0, 1] (categoría 3)

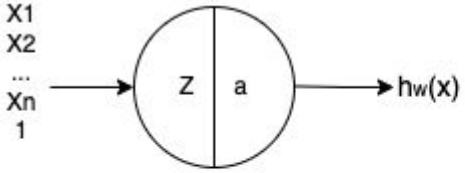
Perceptrón: Funciones de activación (a)

- El perceptrón permite la clasificación de instancias en clases binarias de manera simultánea
- Existen múltiples funciones de activación. Algunas de las más comunes son:
  - **Heaviside step function**

    heaviside(z) = { 0 si z < 0; 1 si >= 0 }

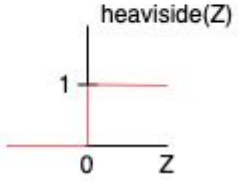
  - **Sign function**

    sgn(z) = { -1 si z < 0; 0 si z = 0; 1 si z > 0 }



$Z(X) = X_1W_1 + X_2W_2 + ... + X_nW_n + b$

$a(Z(X)) = \text{heaviside}(Z(X)) = 0 \text{ si } Z(X) < 0$
$1 \text{ si } Z(X) >= 0$



¿Gráfica y fórmula de **sgn(z)**?

Podemos usar una función de activación u otra de modo intercambiable dentro de una misma arquitectura: para cada TLU de mi Perceptrón elegir ***a*** (heaviside o sign).
A día de hoy, existen otras func. de activación más complejas, como la sigmoide, que nos proporcionan valores continuos que se corresponderá con la probabilidad de pertenecer a una clase u otra.

Perceptrón: Caso práctico I

Visualización del límite de decisión del Perceptrón.ipynb

El perceptrón sí puede recibir valores continuos en sus datos de entrada (características). Además, las funciones que computa la neurona no va a ser únicamente una suma de las entradas, sino que va a ser una suma ponderada por una serie de parámetros del modelo w1, w2, …, wn (función de agregación), y también, ya no vamos a disponer de un Threshold, ya que se traduce en una nueva característica de entrada que siempre va a ser igual a 1, bias, y que tendrá un peso asociado *b* (función de activación o de decisión)

Vamos a ver en este caso práctico muy básico, cómo funciona el Perceptrón, es decir, cuál es el modelo que construye. El objetivo, únicamente, es ver cómo se crea el límite de decisión.

Perceptrón: Construcción del modelo

Forma en la que se construye el modelo o encuentra el valor óptimo de esos parámetros o pesos que hemos visto anteriormente.

Hemos visto como el Perceptrón es una arquitectura de red neuronal bastante sencilla, formada por una única capa de neuronas artificiales TLUs.

En esta capa podemos tener tantas TLUs como nosotros cosideremos, y hemos visto que se utiliza fundamentalmente para realizar tareas de clasificación.

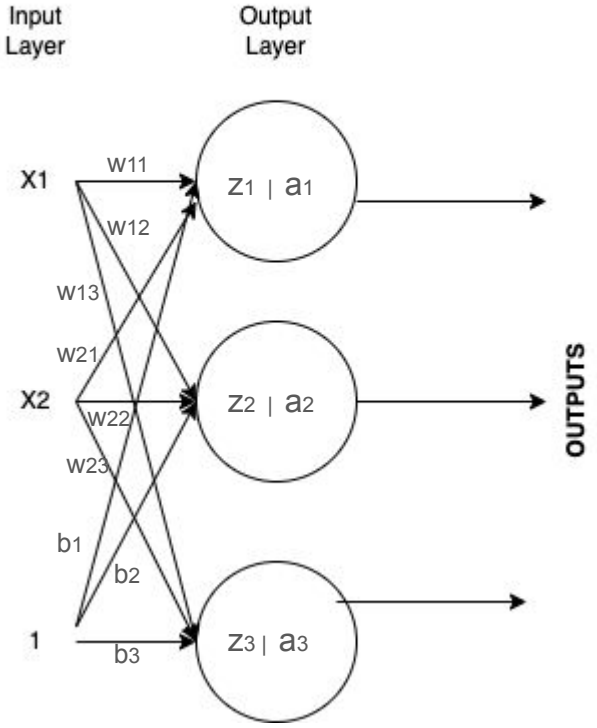
Concretamente, veíamos funciones de activación y veíamos como cada una de las TLU proporcionaba. Por ejemplo, con heaviside era un valor binario.

Básicamente, está tomando una decisión entre cero y uno (clasificando entre clase positiva o negativa).

Perceptrón: Construcción del modelo

- Entrenamiento del Perceptrón

$$W_{i,j} = W_{i,j} + \eta (y_j - \hat{y}_j) X_i$$



- Ejemplo con filtro de Spam

(X_1) #faltas ort.	**(X_2) #html**	**(label) y**
1	5	1
1	0	0
…	…	…

TLU1 -> (w11, w21, b1)

1. Inicializamos aleatoriamente los parámetros:
   - w11 = -1; w21 = 0; b1 = -0.5
   - Teniendo en cuenta estos valores, calculamos z1 | a1 para x1 y x2:

$z_1(X) = W_{11}*X_1 + W_{21}*X_2 + b_1 = -1*1 + 0*5 + (-0.5) =$
$= -1 + 0 - 0.5 = -1.5$

$a_1(z(X))$ = heaviside = 0 (porque $Z(X) < 0$) $\rightarrow \hat{y}_1 = 0$

Ahora vamos ajustando los parámetros iterativamente:

Perceptrón: Construcción del modelo

$w_{11} = w_{11} + \eta(y_1 - \hat{y}_1)X_1 = -1 + 0.5(1 - 0)1 = -1 + 0.5 =$ **-0.5**

η -> learning rate: hiperparámetro que indica la velocidad con la que vamos a modificar esos parámetros del modelo para que se acerquen al valor óptimo (se verá más en detalle). Si es muy grande puede ser perjudicial para el aprendizaje.

Continuaría haciendo el cálculo iterativo, tanto para la característica w21 y b1 de modo que vayan moviéndose hacia su valor óptimo.

Aleatoriamente, cuando iniciamos w11 era -1, y ahora es -0.5, es decir, lo ha modificado en una dirección que hace que el resultado de este término Z vaya más ajustado con la etiqueta *y* que tenemos en nuestro conjunto de datos:

$Z_1(X) = -0.5*1+ 0*5 + (-0.5) = -1$ (antes era -1.5)

Necesitamos que el valor de los pesos, cuando calculamos Z para este ejemplo, nos dé un valor positivo y, por tanto, la función de activación nos devuelva 1 (como es y_1)

Esto es lo que vamos a ir haciendo iterativamente para cada uno de los pesos y para cada uno de los ejemplos. El resultado final será un valor de W1, W2, W3 que se ajusta lo mejor posible a lo que tenemos marcado en nuestro conjunto de datos (entrenamiento).

Perceptrón: Construcción del modelo

La clave es que, una vez que hemos encontrado ese valor óptimo de los pesos w1, w2 y w3, que hace las mejores predicciones para nuestro conjunto de datos de entrenamiento de experiencia pasada, podemos utilizar este Perceptrón para coger un nuevo email que nos llegue, extraer de este nuevo email el número de faltas de ortografía y el número de tags html. Y, proporcionándole estas dos características X1 y X2, al tener ya ajustados los pesos, podemos presuponer que nuestro Perceptrón realizará una predicción acertada sobre si ese correo electrónico se corresponde con spam o legítimo.

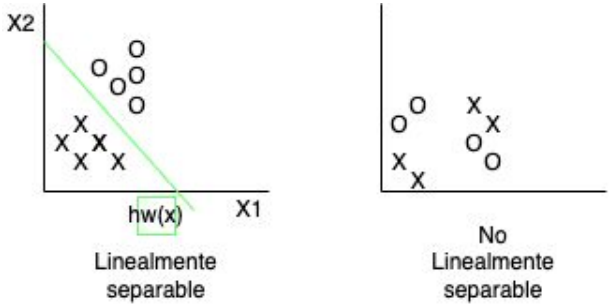
Más adelante, veremos cómo, al cambiar la función de activación, este proceso de aprendizaje se va a complicar un poco. Pero la intuición es muy similar a esto:

vamos a ir comparando las predicciones de nuestro algoritmo con lo que tenemos etiquetado en nuestro conjunto de datos (aprendizaje supervisado).

Perceptrón: Limitaciones

- El Perceptrón no proporciona como resultado una probabilidad, sino que realiza las predicciones basándose en un t**hreshold estático**
  - *^y = heaviside(z) = 0 si Z < 0 ó 1 si Z >= 0*
- Construye **límites de decisión lineales**

(solo resuelve los linealmente separables)



- El uso de Perceptrones tiene grandes limitaciones, no permite resolver problemas sencillos como el problema de clasificación XOR
- Más adelante, se descubre que muchas de las limitaciones del Perceptrón se solucionan añadiendo más capas de TLUs, apareciendo el **Multi-Layer Perceptron** (este algoritmo sí es muy utilizado a día de hoy)

Caso práctico: Clasificación de imágenes

Clasificación de imágenes con el Perceptrón.ipynb

Este es el último ejercicio donde usamos el Perceptrón, después no lo usaremos más de manera aislada, pero es muy importante comprenderlo bien porque forma la base de los algoritmos complejos de DL.

En este caso, nuestro Perceptrón estará formado por 10 neuronas (output layer) para predecir los números entre 0 y 9.

Usamos un límite de decisión lineal para tratar de clasificar imágenes (usando sus píxeles) en 10 clases diferentes, por lo que la predicción no siempre es buena.

A continuación, veremos Perceptrón multicapa, es decir, concatenar más capas de neuronas para mejorar los resultados de predicción de este algoritmo.