



Math & Deep Learning

Redes densas y otras funciones no lineales

Índice

01. Introducción _____ pág. 03

 01. Frank Rosenblatt _____ pág. 04

 02. Un poco más de historia _____ pág. 05

02. Redes Neuronales Densas _____ pág. 08

 01. El lenguaje de las redes neuronales

 pág. 08

 02. Redes Densas _____ pág. 15

 03. Perceptrón multicapa _____ pág. 17

03. El perceptrón simple _____ pág. 19

 01. Bases de datos supervisadas y linealmente separables _____ pág. 19

 02. El entrenamiento _____ pág. 21

 03. Datos casi-linealmente separables

 pág. 23

 04. Evaluación de modelos de clasificación

 pág. 25

05. Margen, sobreajuste y regularización

pág. 27

04. Stochastic Gradient Descent _____ págs. 30

 01. La derivada _____ págs. 30

 02. El método del gradiente de Cauchy

 pág. 32

 03. Método del gradiente estocástico _____ -

 pág. 34

01 Introducción

Bienvenidos al curso Math & Deep Learning. En él buscamos introducir al estudiante en los fundamentos matemáticos y conceptuales que sustentan las arquitecturas modernas de redes neuronales. Más allá de la aplicación práctica, se enfatiza el entendimiento riguroso de cómo las estructuras algebraicas, los procesos estocásticos y las funciones no lineales dan forma al aprendizaje profundo. El estudiante no solo obtendrá destrezas técnicas, sino que comprenda la matemática que hace posible el aprendizaje profundo, con el fin de construir y adaptar modelos más sólidos, interpretables y eficientes.

A lo largo del programa estudiaremos redes densas, convolucionales y recurrentes; vinculándolas con herramientas analíticas como el análisis de Fourier y los procesos estocásticos. También abordaremos el entrenamiento y la optimización de redes, explorando la paralelización y técnicas de tuning, así como la aplicación de auto-encoders en conjunto con la teoría de valores extremos. Finalmente, exploraremos los modelos de Generative AI y GANs, conectando la matemática con los desarrollos más recientes en inteligencia artificial. Los módulos del curso son los siguientes:

1. Redes densas & otras funciones no lineales
2. Redes convolucionales (CNN) & Análisis de Fourier

3. Redes recurrentes (RNN, LSTM, GRU) & Procesos estocásticos
4. Optimización en redes densas & GPU
5. Autoencoders (densas & recurrentes) v.s. Teoría de los Valores Extremos
6. Generative Adversarial Networks (GAN) & Teoría de Juegos

⌚ El repositorio de los casos prácticos se puede encontrar en el siguiente [link](#).

Frank Rosenblatt



Es un psicólogo estadounidense conocido como el padre del Aprendizaje Profundo, sus investigaciones en neurociencias lo acercaron a lo que hoy conocemos como la inteligencia artificial. En 1960 construyó

Mark I Perceptron la primera computadora que logró aprender utilizando un algoritmo. Actualmente este modelo y algoritmo son la base de las redes neuronales, una de sus obras escritas más importantes es "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" donde resume sus investigaciones sobre este tema.

Un poco más de historia

Históricamente se cuentan varias etapas del desarrollo del Deep Learning desde su surgimiento hasta nuestros días. En esas etapas recibió distintos nombres de acuerdo a las diferentes perspectivas e investigadores que han aportado a su desarrollo. Por esta razón da la impresión de ser una disciplina novedosa, sin embargo, su origen se remonta a los años cuarentas, pero es hasta 2006 que se conoce con el nombre de Deep Learning.

Los primeros modelos, como el de McCulloch y Pitts, fueron un intento por emular la manera en que el aprendizaje ocurre del cerebro, por ello uno de los nombres que recibió el aprendizaje profundo fue el de redes neuronales artificiales (ANN'S). Hoy día la neurociencia ya no es una guía predominante para el deep learning.

En los años cincuentas, el perceptrón de Rosenblatt se convirtió en el primer modelo que aprendía los pesos a partir de ejemplos dados, pero

tenía limitaciones al enfrentarse con modelos no lineales.

Una segunda ola de conocimiento surgió en los ochentas conocida como conexiónismo. Planteaba que un gran número de unidades computacionales simples trabajando juntas podía alcanzar un comportamiento inteligente. Los resultados representativos de ésta corriente fueron el perceptrón multicapa, el algoritmo de backpropagation y la modelación de las series de tiempo. Las primeras redes neuronales convolucionales y recurrentes datan de esta oleada. Sin embargo, esta era vio limitaciones en su desarrollo principalmente a falta de poder computacional asequible, recursos de los que ya pudo disfrutar en el siguiente periodo.

En 2006, la publicación del artículo de Hinton A Fast Learning Algorithm for Deep Belief Nets inspiró el florecimiento de nuevos algoritmos y redes neuronales cada vez más profundas. Posteriormente se adoptaron las tarjetas gráficas para acelerar el entrenamiento y otorgar más poder computacional; a la par que se incorporaron grandes bases de datos de las que se carecía en la segunda ola.

A partir de 2014 se sobrevino la revolución en el procesamiento del lenguaje natural con los mecanismos de atención Seq2Seq, Attention y Transformers. Más tarde se desarrollaron las redes generativas GANs y otros enfoques para generación no supervisada.

El aprendizaje por refuerzo comenzó a desarrollarse en la década de

2010 con las Deep Q-networks, y se hizo particularmente popular con el aporte de Alpha Go en 2016 y posteriormente, con los grandes modelos de lenguaje.

02 Redes Neuronales Densas

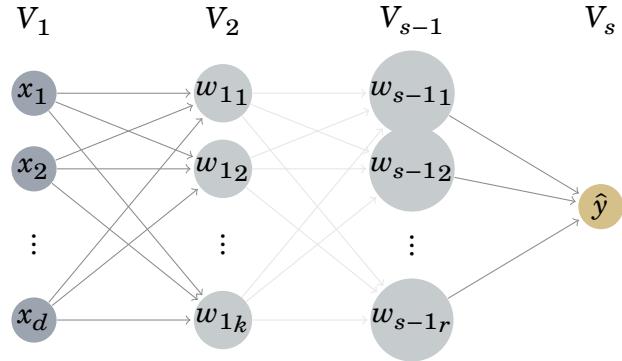
El lenguaje de las redes neuronales

Las redes neuronales feedforward son modelos fundamentales en Deep Learning. Dada una base de datos $S = \{(x_i, y_i)\}_{i \leq N}$, las redes neuronales feedforward tienen el propósito de aproximar la relación entre las características x_i y la supervisión correspondiente $f^*(x_i) = y_i$. Para ello, define una función $f(x, \beta) = y$; y durante el entrenamiento, la red va aprendiendo los parámetros β necesarios para aproximar f a f^* . De acuerdo con [1], se suele pensar a las redes feedforward como máquinas de aproximación de funciones, diseñadas para lograr la generalización estadística.

Estas redes reciben su nombre porque la información fluye en una sola dirección a través de una secuencia de composiciones de funciones (capas), sin recibir retroalimentación de los pasos previos. Solamente se sitúa el input x en la primera capa y la y señala el objetivo que deben alcanzar los parámetros, para concluir con \hat{y} en la última capa. Lo que ocurre en resto de las capas (capas ocultas) no está directamente especificado por los datos de entrenamiento, sino que es deber del algoritmo de aprendizaje decidir qué hacer con esas capas para obtener el output deseado. A continuación formalizaremos estos conceptos.

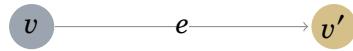
Definición 01.1. La arquitectura de una red neuronal feedforward es una familia de funciones que satisfacen lo siguiente:

- Sea $G = (V, E)$ un grafo dirigido, finito y acíclico; es decir, tenemos un conjunto finito de vértices $v \in V$. Los elementos $e \in E$ se pueden interpretar como flechas entre vértices que poseen una dirección, además no hay una secuencia de elementos en E que empiece y termine en un mismo vértice.
- A los elementos en V los llamaremos neuronas.
- Los elementos en E serán transformaciones lineales.
- Una función llamada función de activación $\rho : \mathbb{R} \rightarrow \mathbb{R}$
- Una partición disjunta del conjunto de vértices $V = V_1 \cup \dots \cup V_s$ donde cada nodo en V_{t-1} está conectado a algún elemento de V_t .
- El parámetro s será el número de capas, y usualmente se conoce como la profundidad al modelo.
- V_1 es un conjunto disjunto de vértices con tamaño $d + 1$ llamada la capa de entrada y V_s , llamada la capa de salida, tiene un solo nodo al que denotaremos como \hat{y} .



Definición 01.2. Dada una arquitectura de una red neuronal feedforward, una red neuronal es lo siguiente:

- Una asignación $w_1(v)$ de un vector de cierta dimensión para cada neurona $v \in V$,
- Una asignación $w_2(e)$ de una matriz para cada arista $e \in E$,
- Las asignaciones anteriores satisfacen que si $v \in V_i, v' \in V_{i+1}$ están conectados por algún $e \in E$ entonces el tamaño de la matriz $w_2(e)$ es $n \times m$ donde el tamaño de los vectores $w_1(v), w_1(v')$ son iguales a n y m respectivamente.



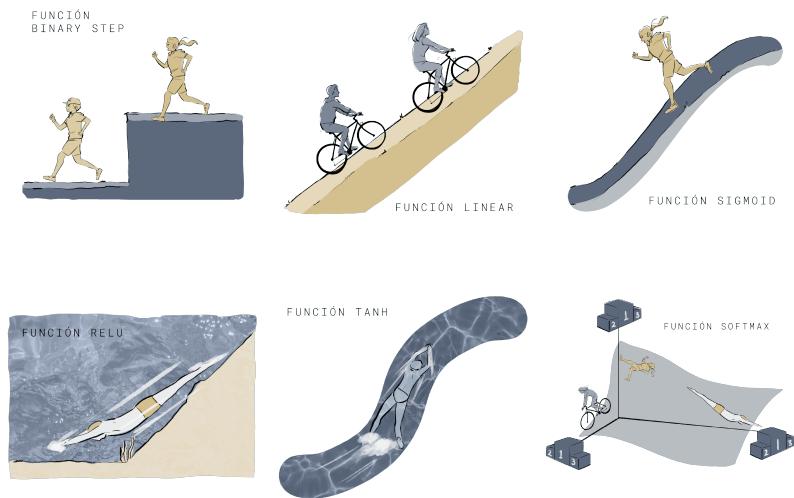
$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{w_2(e)} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_m \end{pmatrix}$$

$$w_2(e) = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix}$$

- Una función $f : X^d \rightarrow Y$ que puede calcularse utilizando la información anterior en orden de izquierda a derecha V_{t-1} to V_t . La operación parcial en cada una de las neuronas se ve de la siguiente forma:

$$\rho(w_2(e)w_1(v) + b) \quad (02.1)$$

01.1 Funciones de activación



Como lo vimos en la definición, una red neuronal depende de una elección de las funciones de activación. En esta sección hablaremos sobre todo de dos funciones de activación:

Definición 01.3. Definimos a la función sigmoide $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ utilizando la siguiente fórmula:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (02.2)$$

La función sigmoide es ampliamente utilizada por ejemplo en el algoritmo de la regresión logística debido a sus propiedades, por ejemplo, que es una función continua, acotada entre 0 y 1, que modela muy bien la probabilidad condicional $\mathbb{P}(x|y = 1)$. Es posible generalizar la función anterior a vectores con tamaño superior de la siguiente manera:

Definición 01.4. Dado un $v = (v_1, v_2, \dots, v_d) \in \mathbb{R}^d$, definimos la función SoftMax de v como

$$\text{SoftMax}(v) = \left(\frac{e^{v_1}}{\sum_{j \leq d} e^{v_j}}, \dots, \frac{e^{v_d}}{\sum_{j \leq d} e^{v_j}} \right) \quad (02.3)$$

Ejercicio 1. Demuestre que si $v \in \mathbb{R}^d$ entonces:

1. $\frac{e^{v_i}}{\sum_{j \leq d} e^{v_j}} \in [0, 1]$
2. $\sum_{i \leq d} \left(\frac{e^{v_i}}{\sum_{j \leq d} e^{v_j}} \right) = 1$

Otra función de activación muy importante para la clasificación es la función RELU:

Definición 01.5. Definimos a la función $\text{RELU} : \mathbb{R} \rightarrow \mathbb{R}$ utilizando la siguiente fórmula:

$$\text{RELU}(x) = \max\{0, -x\} \quad (02.4)$$

Observación 2. Utilizando función RELU es posible definir el algoritmo de entrenamiento del perceptrón como veremos más adelante.

O1.2 Funciones de pérdida

Para pasar de la arquitectura de una red neuronal a una red neuronal, es necesario un proceso de entrenamiento utilizando algoritmos de optimización (que en su mayoría no serán convexos). A su vez para definir un problema de optimización es necesario contar con una función de pérdida.

En esta sección mencionaremos algunas de las funciones de pérdida más utilizadas.

Definición O1.6. Dada la base de datos para una regresión lineal,

$$S = \left\{ (x_1, y_1), \dots, (x_N, y_N) \right\} \quad (O2.5)$$

dónde $(x, y) \in \mathbb{R}^d \times \mathbb{R}$ y una función $f : \mathbb{R}^d \rightarrow \mathbb{R}$, definimos el error de mínimos cuadrados de f como el promedio de los cuadrados de las diferencias entre $f(x_i)$ y las y_i .

$$\text{err}_S(f) = \frac{1}{N} \cdot \sum_{i \leq N} (f(x_i) - y_i)^2 \quad (O2.6)$$

Esta función de pérdida normalmente se utiliza para evaluar el error de la regresión lineal.

Definición 01.7. Dada la base de datos para una clasificación binaria

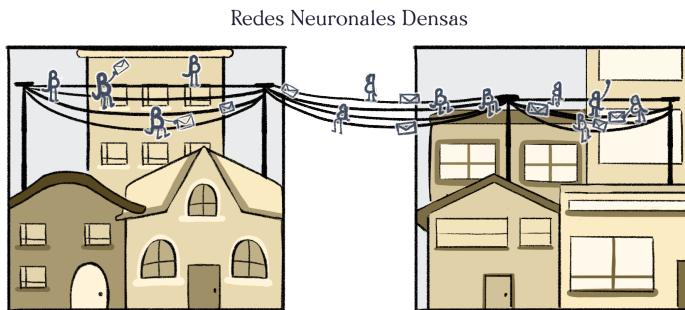
$$S = \left\{ (x_1, y_1), \dots (x_N, y_N) \right\} \quad (02.7)$$

tal que $(x, y) \in \mathbb{R}^d \times \{-1, +1\}$ y una función $f : \mathbb{R}^d \rightarrow \mathbb{R}$, definimos la función de pérdida de la entropía cruzada de f en (x, y) :

$$H(y, f(x)) = - \sum_{i \leq d} y_i \log(f(x_i)) \quad (02.8)$$



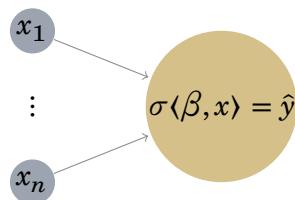
Redes Densas



Una de las redes que utilizaremos comúnmente, es la del perceptrón multicapa en la que las neuronas de una capa están conectadas con todas las neuronas de la siguiente capa.

Una red neuronal densa con una capa es la arquitectura que conecta todas las d características (coordenadas) de $x = (x_1, \dots, x_d)$ con una neurona \hat{y} , de tal manera que a cada característica se le asocia un peso β .

El perceptrón (que estudiamos con profundidad en el curso de ML & IA) es una generalización de ésta idea.



El algoritmo del perceptrón es una generalización de ésta idea. Fue uno de los primeros modelos de aprendizaje supervisado, originalmente propuesto por Frank Rosenblatt. Su input es un vector $x = (x_1, x_2, \dots, x_n)$

y su salida es binaria $y \in \{0, 1\}$.

Como primer paso calcula la suma ponderada con pesos $\beta = (\beta_1, \beta_2, \dots, b_n)$ inicializados de manera aleatoria o cero, más un sesgo b .

$$z = \sum_{j=1}^n \beta_j \cdot x_j + b$$

Como segundo paso, aplica la función de activación obteniendo $\hat{y} = \rho(z)$. Originalmente Rosenblatt utilizó la función escalón (Heaviside), pero en versiones más modernas es común utilizar las funciones sigmoidal, tanh o ReLU.

$$H(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases} \quad (02.9)$$

Si $\hat{y} \neq y$, entonces se actualizan los pesos y sesgos de la siguiente manera:

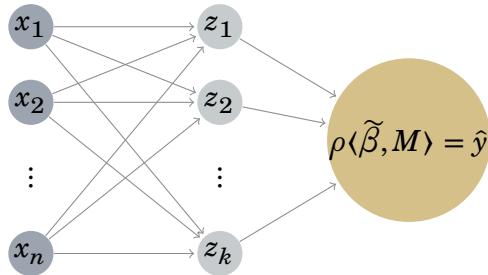
$$\beta_{t+1} \leftarrow \beta_t - \eta (\hat{y} - y) x$$

$$b_{t+1} \leftarrow b_t + \eta (\hat{y} - y)$$

Perceptrón multicapa

En un perceptrón multicapa, los k nodos de la capa V_{t-1} están relacionados con cada uno de los nodos de V_t mediante una regresión logística, de tal modo que $z = \sigma(\beta, x)$ para $x = (x_{t-1,1} \dots x_{t-1,k})$ y $\beta \in \mathbb{R}^k$ para cada $z \in V_t$. En este punto es preciso notar que los pesos β dependen de toda la capa anterior.

Si quisieramos hacer una predicción binaria para un conjunto de datos, como en O2.7, tendremos que añadir una capa final con un solo nodo, $\hat{y} = \rho(\tilde{\beta}, M)$ donde M es una matriz que contiene a todos los nodos de las capas internas, y ρ es una función de activación. Sin embargo, es posible realizar clasificación multiclasa utilizando la función de activación softmax y entropía cruzada como función de pérdida.



Definición 03.1. Si una capa intermedia tiene un vector de neuronas $x \in \mathbb{R}^n$ y la siguiente capa tiene un vector de neuronas $z \in \mathbb{R}^m$, entonces un perceptrón multicapa entre ellas con función de activación ρ es una matriz $M \in \mathbb{R}^{m \times n}$ y un vector $b \in \mathbb{R}^m$ tales que las entradas del vector

z son:

$$z_j = \rho(\langle m_j, x \rangle + b_j) \quad (O2.10)$$

Comúnmente lo abreviaremos con la notación: $z = \rho(Mx + b)$

La cuestión sobre cuántas capas se deben utilizar para abordar un caso como el de la predicción binaria, se estudiará más adelante y siempre depende del tipo de problema. Asimismo, hay heurísticas propias para determinar cuál función de activación será la óptima según el problema que se pretende resolver.

03 El perceptrón simple

Bases de datos supervisadas y linealmente separables

A lo largo del curso supondremos que enfrentamos un problema clásico de aprendizaje supervisado tipo clasificación binaria en \mathbb{R}^d , donde buscamos entrenar a nuestro modelo con un conjunto de ejemplos

$$S = \left\{ (x_1, y_1), \dots, (x_N, y_N) \right\}$$

Donde $x_i \in \mathbb{R}^d$ son las variables explicativas y $y_i \in \{-1, +1\}$ es la supervisión¹.

En la práctica, estos ejemplos son bases de datos con d características.

S :	$\begin{array}{ c c }\hline x_1 & (x_{1,1}, x_{1,2}, \dots, x_{1,d}) \\ \hline x_2 & (x_{2,1}, x_{2,2}, \dots, x_{2,d}) \\ \hline \vdots & \vdots \\ \hline x_N & (x_{N,1}, x_{N,2}, \dots, x_{N,d}) \\ \hline & y_N \\ \hline \end{array}$
-----	---

Para aligerar la notación, en esta sección supondremos que la última coordenada de todos nuestras observaciones x_i es igual a uno i.e. $x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d-1}, 1)$. Hacer esta suposición siempre es posible

¹Utilizamos los valores 1 y -1 para clasificar nuestros registros, pero podría ser cualquier clasificación binaria

si en lugar de considerar nuestras observaciones en \mathbb{R}^d lo hacemos en el hiperplano $X_{d+1} = 1$ en \mathbb{R}^{d+1} .

Haremos una suposición llamada separabilidad que puede parecer exagerada sin embargo será necesaria para el tratamiento matemático formal, más tarde explicaremos cuándo es plausible suponerla:

Definición 01.1. Decimos que S es separable por una función lineal o linealmente separable, si existe un hiperplano $H \subseteq \mathbb{R}^d$ tal que de un lado de este hiperplano están todos los puntos clasificados con $+1$ y del otro lado estarán todos aquellos clasificados con -1 .

Formalmente la hipótesis de separabilidad significa lo siguiente:

Existe un vector $\beta^* \in \mathbb{R}^d$ de tal forma que siempre que $y_i = 1$ entonces $\langle \beta^*, x_i \rangle > 0$ y si $y_i = -1$ entonces $\langle \beta^*, x_i \rangle < 0$ (el caso en el que algún ejemplo x_i satisface la igualdad, corresponde al hecho geométrico de pertenecer al hiperplano definido por β^*).

Ejercicio 3. Sea S una base de datos tal que los $x_i = (x_i, y_i)$ pertenecen al plano cartesiano. Verifique que la hipótesis de separabilidad en este caso significa que existen números reales $m, b \in \mathbb{R}$ tales que $\langle \beta^*, (x_i, y_i, 1) \rangle = \langle (m, -1, b), (x_i, y_i, 1) \rangle$ es mayor o menor a cero.

Sea $\beta^* \in \mathbb{R}^d$ los parámetros del hiperplano que separa a nuestros

ejemplos. Definamos la función signo $sign_{\beta^*} : \mathbb{R}^d \rightarrow \{-1, 1\}$ como

$$sign_{\beta^*}(x) = \begin{cases} 1 & \text{si } \langle x, \beta^* \rangle > 0 \\ -1 & \text{si } \langle x, \beta^* \rangle < 0 \end{cases} \quad (03.1)$$

Utilizando esta función, podemos reescribir la hipótesis de separabilidad de la siguiente manera: Sea $sign_{\beta^*}(x_i) = \hat{y}_i$ entonces la hipótesis de separabilidad significa que $y_i \cdot sign_{\beta^*}(x_i) > 0$ para todos los $i \leq N$.

El entrenamiento

El algoritmo del perceptrón busca descubrir los valores de β^* únicamente conociendo S , para ello procede inductivamente de la siguiente manera:

Definición 02.1. Algoritmo del perceptrón en T épocas

1. Comenzamos con $\beta_0 = (1, 1, \dots, 1)$
2. Para cada $t \leq T$ vamos a re-ordenar los elementos en la base de datos de manera aleatoria y vamos a recorrer la base de datos completa en ese orden. Así, en la primera época denotaremos $x_i = x_{i,1}$ a los elemenots de S . En la segunda época, $t = 2$ denotaremos como $x_{i,2}$ a los elementos reordenados de S .

3. Si suponemos que $\beta_{i,t}$ está definido, buscamos el primer ejemplo (en ese orden) $(x_j, y_j) \in S$ en el que NO se cumple que:

$$y_j \cdot \text{sign}_{\beta_i}(x_j) > 0.$$

- Si no encontramos algún índice j que cumpla los criterios de nuestra búsqueda entonces β_i es nuestra propuesta para clasificar puntos. Es decir $\beta_{i,t} = \beta_{i+1,t}$.
- Si por el contrario encontramos algún índice j que cumpla los criterios de nuestra búsqueda entonces actualizamos

$$\beta_{i+1,t} := \beta_{i,t} + y_j x_j$$

4. Volvemos al paso dos, esta vez $t + 1$.

Observación 4. Notemos que si nos equivocamos en el ejemplo j , entonces

$$y_j \langle \beta_{i+1}, x_j \rangle = y_j \langle \beta_i + y_j x_j, x_j \rangle = y_j \langle \beta_i, x_j \rangle + \|x_j\|_2$$

Si el algoritmo se detiene en T épocas, tenemos una red neuronal neuronal $f : \mathbb{R}^{d+1} \rightarrow \{-1, +1\}$ de una capa y $d + 1$ neuronas definida por $f(x) = \text{sign}_{\beta_T}(x)$.

02.1 Teorema de convergencia del perceptrón

Teorema 5. Denotemos por R al máximo del producto punto de todos los elementos de S (el máximo de la norma de los vectores de entrada); y denotemos como B al mínimo de todas las betas para las cuales $y_i \langle x_i, \beta \rangle$ es positivo, es decir, de entre todos los vectores β que separan correctamente los datos, tomaremos el que tenga norma menor.

$$R = \max_{i \leq N} \{ \langle x_i, x_i \rangle \} = \max_i \{ \|x_i\|^2 \} \quad (O3.2)$$

$$B = \min_{\beta \in \mathbb{R}^d} \{ \langle \beta, \beta \rangle : y_i \cdot \langle x_i, \beta \rangle > 0 \text{ para todo } i \} \quad (O3.3)$$

Bajo la hipótesis de separabilidad sobre S el algoritmo del perceptrón se detiene (es decir encuentra alguna β que clasifica correctamente a los ejemplos en S) en $(RB)^2$ -pasos.

Desafortunadamente la geometría del problema algunas veces implica que tiempo en el que el perceptrón se detiene es demasiado grande.

Datos casi-linealmente separables

La versión del algoritmo que utilizan la mayoría de las librerías es una variante del algoritmo del percetrón anteriormente estudiado. Recorremos que su eficacia solo está garantizada cuando la información es

linealmente separable. En esta sección presentaremos una variante del perceptrón útil cuando solo una pequeña porción de los datos no son linealmente separables.

Definición 03.1. Perceptrón parcial:

- Etapa 0 1. Comenzamos con $\beta_0 = (1, 1, \dots, 1)$.
 2. Definimos un parámetro β_s y le asignamos el valor $\beta_s = \beta_0$.
 3. Definimos un parámetro $Ite_s(0)$ y le asignamos el valor $Ite_s(0) = 0$.

- Etapa i 1. El perceptrón se actualizará de manera usual como en el algoritmo 02.1, sin embargo esta vez guardaremos el tamaño I de la cadena exitosa de pruebas de ejemplos en S (i.e. β_{i-I} clasificó correctamente a I ejemplos en S).
 2. Se actualizará $Ite_s = I$ cuando $Ite_s < I$.
 3. Se actualizará $\beta_s = \beta_{i-I}$.

Observación 6. El algoritmo anterior tiene un problema pues podría actualizar al vector β_s por un vector inferior a los probados anteriormente ya que solo guarda la información de la última iteración.

Teorema 7. Si los coeficientes β_i son números racionales entonces el algoritmo anterior, con probabilidad uno converge a una solución óptima.

Evaluación de modelos de clasificación

Nos concentraremos con un problema de clasificación binaria cuyas clases corresponden a $-1, +1$, vale la pena notar que mucho de lo que haremos para este caso, tiene sentido para problemas multi-clases.

Muchos de los algoritmos que vamos a trabajar en el curso requieren que hagamos una partición de la base de datos en dos secciones. El conjunto de entrenamiento y el conjunto test. Es importante reservar una porción de datos que el modelo no ha visto, para evaluar su desempeño en datos completamente ajenos a los utilizados en su entrenamiento.

En este ejemplo supondremos que $+1$ significa no-churn (clientes que no tienen desapego a un producto) y -1 significa churn (clientes que tienen desapego al producto).

Definición 04.1. Definimos las siguientes cantidades asociadas a un modelo de clasificación f y a un conjunto de test S de la siguiente forma:

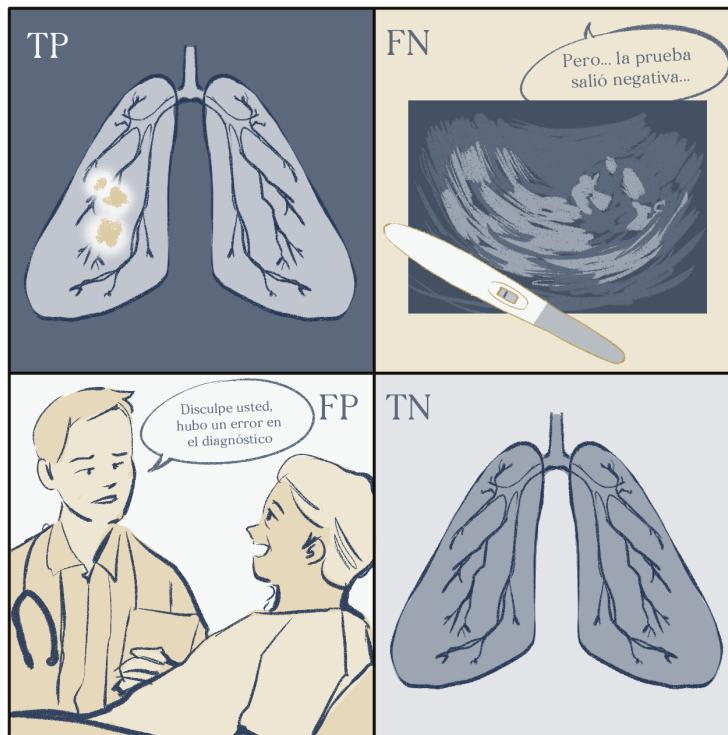
TP es la cantidad de elementos en S etiquetados como $+1$ tales que el modelo f también los etiqueta como $+1$.

TN es la cantidad de elementos en S etiquetados como -1 tales que el modelo f también los etiqueta como -1 .

FP es la cantidad de elementos en S etiquetados como -1 tales que el modelo f los etiqueta como $+1$.

FN es la cantidad de elementos en S etiquetados como $+1$ tales que el modelo f los etiqueta como -1 .

Una representación común de esta evaluación es conocida como la matriz de confusión, que usualmente se presenta como en la imagen:



Definición 04.2. Definimos las siguientes medidas de evaluación del modelo f en el conjunto de test S :

- La sensibilidad $D = \frac{TP}{TP + FN}$
- La precisión $P = \frac{TP}{TP + FP}$

En un modelo de diagnóstico médico como el de la detección de cáncer, la sensibilidad mide la capacidad del modelo para detectar como

positivos a los caso realmente positivos. Mientras que la precisión mide la disposición del modelo a no arrojar falsos positivos.

Margen, sobreajuste y regularización

En esta sección hablaremos sobre uno de los problemas más comunes y que pueden convertir machine learning en una experiencia desagradable: el sobreajuste.

Intuitivamente, el sobreajuste se puede ver como una diferencia sustancial entre el error del entrenamiento y el error del test. Éste último se puede calcular fácilmente como $Err_{test} = \frac{FP + FN}{TP + FN + FP + TN}$. Más precisamente, tendremos sobreajuste cuando el error en el entrenamiento es mucho más pequeño que el error en el test.

Decimos que el modelo subajusta cuando el error en el entrenamiento es muy grande. En este caso el error del test es irrelevante puesto que un modelo tal no es confiable.

Definición 05.1. Si hemos fijado un tiempo t y una base de datos S_t , supongamos que utilizaremos un algoritmo A para predecir un modelo M , se dice que el el modelo M ha incurrido en sobreajuste relativo a la información anterior cuando M incluye sistemáticamente el ruido estocástico (aleatorio) de la base S_t .

El proceso de regularización en Machine Learning es un método para

evitar el sobreajuste, existen distintas maneras de intentarlo y su eficacia no solo depende de nuestras bases de datos sino del algoritmo que estamos utilizando, en el caso del perceptrón la regularización se puede interpretar como un margen entre el hiperplano que predecimos y nuestra base de datos.

Definición 05.2. El algoritmo del perceptrón regularizado con un parámetro $\alpha > 1$:

1. Comenzamos con $\beta_0 = (1, 1, \dots, 1)$.
2. Si suponemos que β_i está definido, buscamos algún ejemplo $(x_j, z_j) \in S$ tal que lo siguiente NO sea cierto: $z_j \cdot \text{sign}_{\beta_i}(x_j) > 0$.
3. Si no encontramos algún índice j que cumpla los criterios de nuestra búsqueda entonces β_i es nuestra propuesta para clasificar puntos entrenada con S .
4. Si por el contrario encontramos algún índice j que cumpla los criterios de nuestra búsqueda entonces actualizamos a $\beta_{i+1} := \beta_i + \alpha z_j x_j$.
5. Volvemos al paso dos.

Notemos que si $\alpha = 1$ entonces el algoritmo del perceptrón regularizado es el algoritmo usual. La regularización también tiene ventajas en la eficiencia computacional del perceptrón como lo muestra el siguiente resultado que generaliza el teorema 5.

Teorema 8. Supongamos que S es separable linealmente con un margen δ , entonces el algoritmo regularizado del perceptrón se detiene (es decir encuentra alguna β que clasifica correctamente a los ejemplos en S) en $\left(\frac{R}{\delta}B\right)^2$ pasos.

Es importante notar que no existe una relación sencilla de explicar entre el margen δ y α .

04 Stochastic Gradient Descent

El método del gradiente es la técnica más utilizada en deep learning para entrenar a una red neuronal. Comenzaremos con un caso muy sencillo para el caso de las regresiones lineales.

La derivada

En esta sección introduciremos las derivadas en una sola dimensión y más adelante serán necesarias las derivadas respecto a más de una variable.

Definición 01.1. Si $l : \mathbb{R} \rightarrow \mathbb{R}$ es una función, entonces diremos que l es diferenciable en un punto $x \in \mathbb{R}$ cuando el siguiente límite existe:

$$\lim_{\delta \rightarrow 0} \frac{l(x + \delta) - l(x)}{\delta}$$

Cuando existe ese límite, lo denotaremos $l'(x)$ y llamaremos "la derivada de l en el punto x ".

A partir de ahora nos interesarán las funciones l que se utilizan como métrica para un algoritmo, por ejemplo, la función de error 02.6.

Supongamos que $S = \{(x_i, y_i)\}_{i \leq N}$ es una base de datos asociada a

una regresión univariada (es decir, $d = 1$), tal que la señal de esta función satisface: $f^*(X) = mX$, entonces

$$l(m) = \text{err}_S(m) = \frac{1}{N} \cdot \sum_{i \leq N} (mx_i - y_i)^2 \quad (04.1)$$

Ejercicio 9. Calcule la derivada de l en 04.1.

Definición 01.2. Si $l : \mathbb{R}^d \rightarrow \mathbb{R}$ es una función, entonces diremos que l es diferenciable en un punto $x \in \mathbb{R}^d$, con respecto a la coordenada $j \leq d$ cuando el siguiente límite existe:

$$\lim_{h \rightarrow 0} \frac{l(x_1, \dots, x_{j-1}, x_j + h, x_{j+1}, \dots, x_d) - l(x_1, \dots, x_d)}{h}$$

Cuando ese límite existe, lo denotaremos $\frac{\partial}{\partial x_j} l(x)$ y llamaremos la derivada parcial de l en el punto x y con dirección j .

El método del gradiente de Cauchy

El método del gradiente es un algoritmo que es comúnmente utilizado en Machine Learning para el proceso de entrenamiento de diversos modelos. En esta sección supondremos que $l : \mathbb{R}^d \rightarrow \mathbb{R}$ es una función que entenderemos como la función de error en algún modelo, por ejemplo, podríamos suponer que

$$l(\beta) = \frac{1}{N} \sum_{i \leq N} (y_i - \langle \beta, x_i \rangle)^2 \quad (04.2)$$

en el caso de una regresión lineal. El método del gradiente consiste en actualizar iterativamente los parámetros β para optimizar la función l .

Definición 02.1. Sea f como en el párrafo anterior y $\beta \in \mathbb{R}^d$, para $\nu > 0$ definimos el algoritmo del gradiente descendente de la siguiente manera:

1. $\beta_0 = (1, 1, \dots, 1)$
2. $\beta_{t+1} = \beta_t - \nu \nabla l(\beta_t)$

El parámetro ν sirve para garantizar que la aproximación no se aleja demasiado de β y se conoce como factor de aprendizaje.

Si desentrañamos con cuidado la definición anterior obtenemos:

$$\beta_{t+1,j} = \left(\beta_{t,1} - \nu \frac{\partial l}{\partial \beta_{t,1}}(\beta_t), \dots, \beta_{t,j} - \nu \frac{\partial l}{\partial \beta_{t,j}}(\beta_t), \dots, \beta_{t,d} - \nu \frac{\partial l}{\partial \beta_{t,d}}(\beta_t) \right)$$

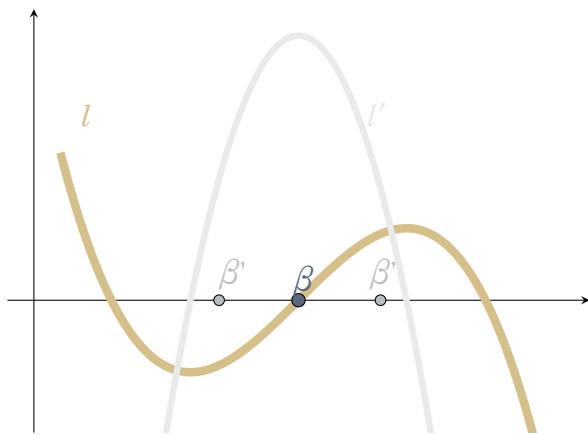
La idea principal detrás del método del gradiente se podría resumir para el caso uno-dimensional, $l : \mathbb{R} \rightarrow \mathbb{R}$, de la siguiente manera. Buscamos un β tal que $l'(\beta) = 0$. Cuando $d = 1$ y $\beta_0 = 0$, la función de pérdida l depende únicamente del parámetro β .

Si hemos encontrado un $\beta' \in \mathbb{R}$ que aún no satisface que $l'(\beta') = 0$, entonces podrían ocurrir los siguientes dos casos:

$$\begin{cases} l'(\beta') > 0 & \text{al disminuir } \beta', \text{ el error } l(\beta') \text{ disminuye} \\ l'(\beta') < 0 & \text{al aumentar } \beta', \text{ el error } l(\beta') \text{ disminuye} \end{cases}$$

Cuando $l'(\beta') > 0$ significa que la pendiente es positiva, por lo que para minimizar la función debemos movernos en dirección negativa. Igualmente, cuando $l'(\beta') < 0$, debemos movernos en dirección positiva para disminuir el error.

Por eso en el primero de los casos deseamos sumarle una cantidad negativa a β' , a saber $l'(\beta')$, y en el segundo caso deseamos sumarle una cantidad positiva, a saber $-l'(\beta')$ para acercarnos al β que optimiza la función. Así la actualización de β_t a $\beta_{t+1} = \beta_t - l'(\beta)$ es cada vez más cercano a β como se aprecia en la siguiente ilustración.



Observación 10. Si una función es diferenciable en un punto, entonces alrededor de ese punto se tiene la siguiente aproximación lineal: Si β es cercana a β_t , entonces

$$l(\beta) \sim l(\beta_t) + \langle \beta - \beta_t, \nabla l(\beta_t) \rangle \quad (04.3)$$

Cuando la función l satisface algunas condiciones de regularidad y convexidad, es posible garantizar la convergencia del método del gradiente, sin embargo algunas veces podría ser problemático. Afortunadamente para las redes neuronales profundas estas técnicas funcionan muy bien.

Método del gradiente estocástico

Uno de los casos principales cuando el método del gradiente podría no converger es cuando la cantidad de datos es demasiado extensa,

en este caso el método del gradiente estocástico permite solucionar la velocidad de convergencia.

Supongamos que $f : \mathbb{R}^d \rightarrow \mathbb{R}$ es una función diferenciable que podemos escribir de la siguiente manera:

$$f(\beta) = \frac{1}{N} \sum_{i \leq N} f_i(\beta) \quad (04.4)$$

Por ejemplo la función de error de las regresiones que vimos anteriormente en donde las $f_i(\beta) = (\langle \beta, x_i \rangle - y_i)^2$.

Es inmediato que el gradiente de f es

$$\nabla f(\beta) = \frac{1}{N} \sum_{i \leq N} \nabla f_i(\beta) \quad (04.5)$$

En este caso el algoritmo de actualización del gradiente estocástico se define de la siguiente manera:

$$1. \beta_0 = (1, 1, \dots, 1)$$

$$2. \beta_{t+1} = \beta_t - \nu \nabla f_j(\beta_t)$$

Para algún $j \in \{1, 2, \dots, N\}$. Por supuesto, la pregunta más natural es: cómo elegir j ?

Supongamos que elegimos j de manera aleatoria y uniforme en $\{1, 2, 3, \dots, N\}$. Es claro que β_{t+1} será una variable aleatoria que de-

pende de j . Calculemos su esperanza:

$$\mathbb{E} [\beta_{t+1}] = \frac{1}{N} \sum_{j=1}^N (\beta_t - \nu \nabla f_j (\beta_t)) = \beta_t - \nu \frac{1}{N} \sum_{j=1}^N \nabla f_j (\beta_t) = \beta_t - \nu \nabla f (\beta_t)$$

Lo anterior es muy interesante porque significa que, en promedio, el efecto del m'etodo del gradiente estocástico será igual al del gradiente determinista, con la enorme ventaja de solo requerir el cálculo de un único gradiente ∇f_j en cada iteración, en lugar de sumar N gradientes ∇f_i como requeriría el método determinista.

B O U R B A K I

COLEGIO DE MATEMÁTICAS

Bibliografía

- [1] Yoshua Bengio, Ian Goodfellow, Aaron Courville, et al. Deep learning, volume 1. MIT press Cambridge, MA, USA, 2017.

BOURBAKI

ESCUELA DE MATEMÁTICAS

CURIOSIDAD Y SABER

CDMX, MEXICO

