BOURBAKI

COLEGIO DE MATEMÁTICAS

BOURBAKI

COLEGIO DE MATEMÁTICAS

Índice

01. Introducción ____ pág. **03**

01. Frank Rosenblatt ____ pág. 04

02. Keras ____ pág. 05

02. El lenguaje de las redes neuronales ____ pág. **06**

01. Funciones de activación ____ pág. 08

02. Funciones de pérdida ____ pág. 10

03. Perceptrón multicapa ____ pág. 12

03. Autoencoders ____ pág. **14**

01. Análisis de componentes principales ____ pág. 14

02. Definición de Autoencoder ____ pág. 16

04. El problema de los valores faltantes ____ pág. **17**

01. Estrategia para la imputación de datos ____ pág. 19

ÍNDICE

05. Complementos sobre entrenamiento de redes neuronales profundas ____ pág. **22**

01. La derivada ____ pág. 22
02. El método del gradiente de Cauchy ____ pág. 24
03. Método del gradiente estocástico ____ pág. 26

06. Caso de uso: segmentación de clientes

pág. **29**

01 Introducción

Las presentes notas son la bitácora del primero de los 6 módulos de nuestro curso Deep Learning Avanzado. En este curso trataremos modelos neuronales diversos para garantizar un mejor rendimiento dependiendo de la estructura interna de las bases de datos.

Este módulo está enfocado en las redes neuronales densas y lo impartimos junto a Édison Vázquez. Además de este documento los invitamos a consultar el Github del curso en este link.

El curso es una invitación al uso de las redes neuronales profundas, la organización de las clases es la siguiente:

1. Introducción a redes neuronales profundas (una hora).
2. Introducción a Tensor Flow y capas densas (una hora).
3. Reducción de la dimensión (dos horas).
4. Caso de uso sobre clustering de clientes (dos horas).
5. Dudas y complementos sobre redes neuronales (dos horas).
6. Asesoría sobre el reto (6 horas).

🔗 El repositorio de Github para esta semana se puede encontrar en esta liga.

Frank Rosenblatt

Es un psicólogo estadounidense quien es 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.

Keras

Keras es un API sobre Machine Learning escrita en Python. Fue creada con la finalidad de permitir a los usuarios experimentar, de modo rápido y sencillo, con varios modelos.

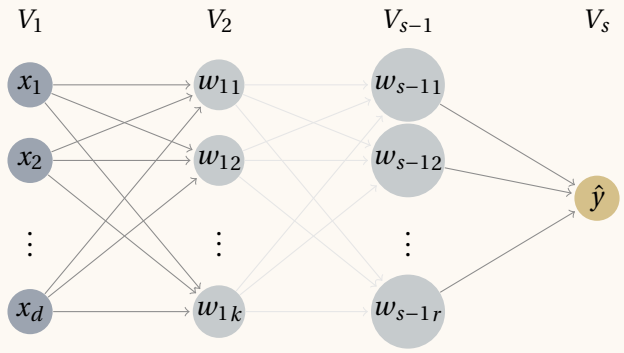
Permite al usuario la creación de algoritmos de modo sencillo para dejarle tiempo a que se enfoque en las partes importantes sobre su investigación en lugar de gastar tiempo escribiendo código de un modelo desde cero.

El módulo se encuentra bastante optimizado para realizar operaciones tensoriales en CPU, GPU o TPU, también permite calcular los gradientes de expresiones arbitrarias, entre otros. Además es bastante intuitiva para uso y tiene una documentación clara.

02 El lenguaje de las redes neuronales

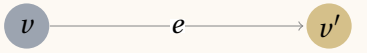
Definition 00.1. La **arquitectura** de una **red neuronal feed-forward** es una familia de funciones que satisfacen lo siguiente:

- 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 vertices que poseen una dirección, además no hay una secuencia de elementos en E que empiece y termine en un vértice.
- A los elementos en V los llamaremos **neuronas**.
- Los elementos en E serán transformaciones lineales.
- Una función llamada **función de activación** $\rho : \mathbb{R} \to \mathbb{R}$
- Una partición disjunta del conjunto de vértices $V = V_1 \cup \ldots \cup V_s$ donde cada nodo en V_{t-1} está conectado a algún elemento de V_t.
- El parámetro s será el **número de capas**,
- V_1 es un conjunto disjunto de vértices con tamaño $d + 1$ y V_s tiene un solo **nodo** al que denotaremos como $\hat{y}$.



Definition 00.2. Dada una arquitectura de una red neuronal feed-forward, **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.



$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{w_2(e)} \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_m \end{pmatrix}$$

- Una función $f : X^d \to Y$ que puede calcularse utilizando la informa-

ción 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:

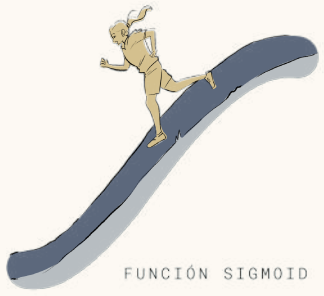
$$\rho\left(w_2\left(e\right) w_1\left(v\right) + b\right) \tag{02.1}$$

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:

Definition 01.1. Definimos a la **función sigmoide** $\sigma : \mathbb{R} \to \mathbb{R}$ utilizando la siguiente fórmula:

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

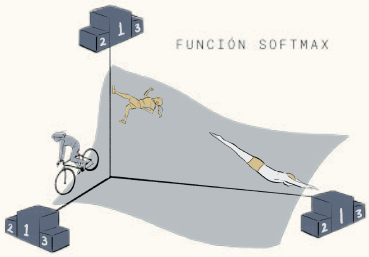


La función sigmoide es ampliamente utilizada por ejemplo en el algoritmo de la regresión logística debido a sus propiedades, por ejemplo, que es una función continua, acotada entre 0 y 1, que modela muy bien la probabilidad

condicional $\mathbb{P}(x|y=1)$. Es posible generalizar la función anterior a vectores con tamaño superior de la siguiente manera:

Definition 01.2. Dado un $v = (v_1, v_2, \ldots, v_d) \in \mathbb{R}^d$, definimos la función **SoftMax** de v como

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



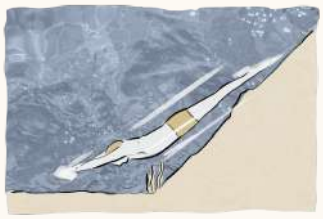
Exercise 01.1. *Demuestre que si $v \in \mathbb{R}^d$ entonces:*

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

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

Definition 01.3. Definimos a la función $RELU : \mathbb{R} \to \mathbb{R}$ utilizando la siguiente fórmula:

$$RELU(x) = \text{máx}\{0, -x\} \tag{02.4}$$



Remark 01.2. *Utilizando función RELU es posible definir el algoritmo de entrenamiento del perceptrón como veremos más adelante.*

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 mencionarems algunas de las funciones de pérdida más utilizadas.

Definition 02.1. Dada la base de datos para una regresión lineal,

$$S = \left\{ \left(x_1, y_1\right), \ldots \left(x_N, y_N\right) \right\} \tag{02.5}$$

dónde $\left(x, y\right) \in \mathbb{R}^d \times \mathbb{R}$ y una función $f : \mathbb{R}^d \to \mathbb{R}$, 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.

$$err_S(f) = \frac{1}{N} \cdot \sum_{i \leq N} \left(f(x_i) - y_i\right)^2 \tag{02.6}$$

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

Definition 02.2. Dada la base de datos para una clasificación binaria

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

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

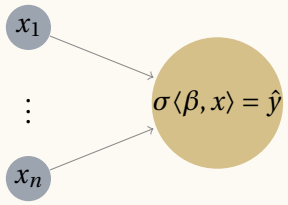
$$H\left(y, f(x)\right) = -\sum_{i \leq d} y_i \log\left(f(x_i)\right) \tag{02.8}$$

Perceptrón multicapa

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.

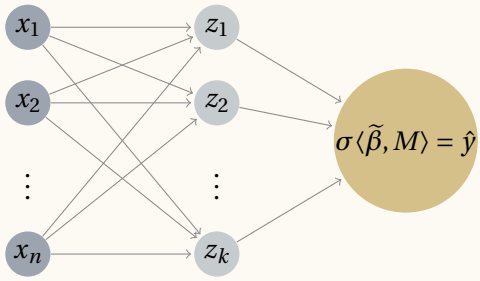
Una **red neuronal densa** con una capa es la arquitectura que conecta todas las d características (coordenadas) de $x = (x_1, \dots x_d)$ con una neurona $\hat{y}$, de tal manera que a cada característica se le asocia un peso.

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



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 $z = \sigma\langle\beta, x\rangle$ 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 queremos hacer una predicción binaria para un conjunto de datos, como en 02.7, tendremos que añadir una capa final con un solo nodo, $\hat{y} = \sigma\langle\widetilde{\beta}, M\rangle$ donde M es una matriz que contiene a todos los nodos de las capas internas, y σ es la función 02.2, como formalizaremos en la siguiente definición.



Definition 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 w son:

$$w_j = \rho\left(\langle m_j, x\rangle + b_j\right) \tag{02.9}$$

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

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. Asímismo, hay heurísticas propias para determinar cuál función de activación será la óptima según el problema que se pretende resolver.

03 Autoencoders

En este capítulo hablaremos sobre dos de los algoritmos más utilizados en ciencia de datos a saber el Análisis de Componentes Principales y los Autoencoders. Utilizaremos estos algoritmos para la imputación de datos.

Análisis de componentes principales

En esta sección describiremos las distintas versiones de PCA y cómo puede entenderse como una técnica para reducir la dimensión. La tercera formulación que enunciaremos será la base para comprender cómo funcionan los autoencoders.

La siguiente definición del análisis de componentes principales es bastante parecida a la definición de la regresión lineal, excepto que la manera de aproximar es distinta.

Definition 01.1. ***PCA como aproximación lineal****. Sea* $X = \{x_1, \ldots, x_N\} \in \mathbb{R}^d$ *y* $k < d$*. El problema de aproximación de k componentes principales corresponde con la solución del siguiente problema de minimización:*

$$PCA(k) = \underset{V \subset \mathbb{R}^d, dim(V)=k}{argmin} \left(\frac{1}{N} \sum_{j \leq N} \left(d\left(x_j, V\right)^2 \right) \right) \tag{03.1}$$

Utilizando proyecciones ortogonales es posible reescribir la ecuación 03.1 de

la siguiente forma:

Proposition 01.2. ***PCA como aproximación de*** k ***vectores ortonormales.*** *Sea* $X = \{x_1, \dots, x_N\} \in \mathbb{R}^d$ *y* $k < d$*. El problema de aproximación por los primeros* k *componentes principales* $v_1, \dots, v_k$ *corresponde con la solución del siguiente problema de minimización:*

$$(V, v_1, \dots, v_k) = \underset{V \in \mathbb{R}^{dk}, V^T V = Id_k, a_j \in \mathbb{R}^k}{argmin} \left(\frac{1}{N} \sum_{j \leq N} ||x_j - V a_j||_2^2 \right) \tag{03.2}$$

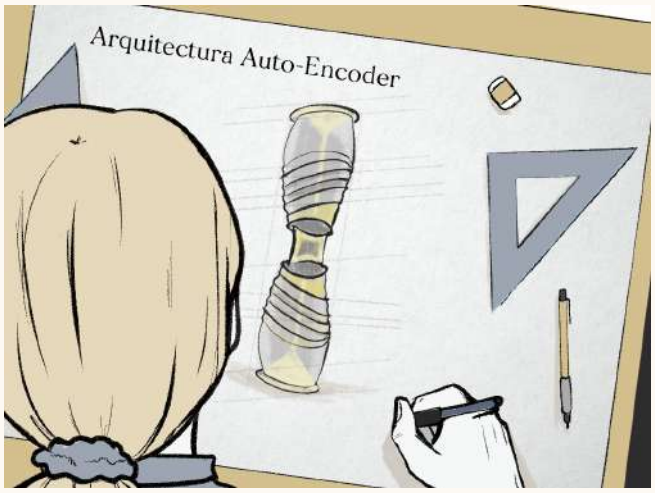
Aunque no es inmediato demostrar la siguiente proposición, es posible reescribir PCA como sigue.

Proposition 01.3. ***PCA como reducción de la dimensión.*** *La solución de la ecuación en la definición 01.2 es equivalente a la siguiente:*

$$(V, U) = \underset{V \in \mathbb{R}^{dk}, U \in \mathbb{R}^{kd}}{argmin} \left(\frac{1}{N} \sum_{j \leq N} ||x_j - VUx_j||_2^2 \right) \tag{03.3}$$

En el análisis de componentes principales las funciones que reducen la dimensión son funciones lineales sin embargo los autoencoders utilizan funciones no lineales para modificar el espacio en el que viven nuestros datos.

Definición de Autoencoder



Los autoencoders son un algoritmo muy similar a la última versión que dimos sobre PCA con la diferencia de que permitiremos transformaciones no necesariamente lineales.

Definition 02.1. *Un **autoencoder** es la solución al siguiente problema de minimización donde f, g son dos redes densas:*

$$(f, g) = \underset{f,g}{argmin}\left(\frac{1}{N}\sum_{j\leq N}\|x_j - g(f(x_j))\|_2^2\right) \tag{03.4}$$

En este caso la reducción de los datos es $f(x_j)$.

Remark 02.2. *Es importante mencionar que aunque esta vez hemos utilizado la distancia euclidiana para medir la diferencia entre nuestros datos es posible usar otras métricas.*

04 El problema de los valores faltantes

En el análisis de datos, uno de los desafíos más comunes es tratar con valores faltantes en un conjunto de datos. La falta de datos es un problema común que aparece en contextos reales y puede comprometer el rendimiento de la mayoría de los modelos de aprendizaje. Los valores faltantes pueden surgir por diversas razones, como errores en la recopilación de datos, la no disponibilidad de información o incluso decisiones del sistema de almacenamiento de datos.

Estos valores ausentes pueden afectar negativamente el rendimiento de los modelos de aprendizaje automático, especialmente si son una parte significativa del conjunto de datos. Si no se abordan adecuadamente, los valores faltantes pueden generar sesgos, disminuir la precisión de los modelos y afectar la generalización de las predicciones.

Existen varios enfoques para manejar los valores faltantes, y los métodos clá-

sicos suelen ser los primeros recursos utilizados para este propósito. Entre los métodos más comunes se encuentran:

- Imputación por media, moda o muestreo: Consiste en reemplazar los valores faltantes con la media (para variables ordenadas) o la moda (para variables no-ordenadas) de los datos no faltantes. Aunque es un enfoque sencillo que puede estar justificado en la ley de los grandes números, no es un enfoque que utiliza el resto de la estructura en la base de datos. También es posible rellenar con un muestreo de una distribución aproximada correspondiente, ya sea ordenada o no. La desventaja de este método es que nos estamos concentrando únicamente en las columnas de manera independiente y no en sus posibles interacciones.

- Imputación por regresión: Utiliza una técnica de regresión para predecir los valores faltantes basándose en las relaciones existentes entre las variables. Este enfoque puede ser más preciso que la imputación por media, pero depende de poder elegir correctamente la variable predictiva.

- Imputación por el valor más cercano (KNN): Este método usa los valores observados más cercanos a los faltantes para estimar los valores ausentes. Si bien es una técnica potente en algunos casos, puede ser computacionalmente costosa y no siempre captura las relaciones más complejas entre las variables.

- Imputación Multivariante por Ecuaciones Encadenadas (MICE): Es un

enfoque estadístico que ofrece una imputación de valores faltantes mediante un proceso iterativo, utilizando un modelo de imputación distinto para cada variable con datos faltantes. Este método trabaja encadenando ecuaciones de regresión multivariante, donde cada variable con valores faltantes se modela condicionalmente a las demás variables, y se imputan los valores faltantes utilizando las predicciones de este modelo. MICE se adapta bien a situaciones en las que las relaciones entre las variables son complejas, ya que tiene en cuenta las interacciones entre las mismas. El método que se propone en este trabajo es un poco similar.

Como en todos los problemas de machine learning, los enfoques clásicos tienen limitaciones, especialmente cuando las interacciones entre las diferentes variables no se consideran adecuadamente debido a los mismos valores faltantes. Es aquí donde las redes neuronales, pueden ofrecer una mejora significativa en la imputación de valores faltantes, considerando no solo las características individuales de las variables, sino también sus relaciones subyacentes.

Estrategia para la imputación de datos

Supongamos que tenemos acceso a un conjunto de datos con valores faltantes al que llamaremos X_{train}, al que un experto (o un histórico) le ha impu-

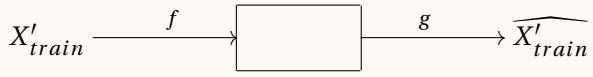
tado algunos de los valores faltantes. A este nuevo conjunto lo llamaremos Y_{train}, quizás este conjunto aún tiene valores faltantes. Podremos suponer que este rellenado de los valores se hizo por medio de un proceso costoso como una auditoría por ejemplo.

La estrategia consiste en llenar los datos faltantes de Y_{train} mediante algún método ingenuo (con promedios, por ejemplo) para construir Y'_{train}.

Ahora entrenaremos un modelo neuronal como un autoencoder que arroje una transformación f a un conjunto de variables latentes; y otra transformación g al conjunto de variables originales. Nuestra intención es aprender una nueva representación del conocimiento de experto.

$$Y'_{train} \xrightarrow{f} \boxed{} \xrightarrow{g} \widehat{Y'_{train}}$$

Utilizando una imputación de datos ingenua para X_{train}, digamos X'_{train}; podemos evaluar las funciones f y g para construir las predicciones de imputación $\widehat{X'_{train}}$.



Idealmente las funciones f y g que han aprendido la representación del conocimiento del experto están forzando a X'_{train} a meterse en la representación en pocas dimensiones. La evaluación del modelo en el conjunto de entrenamiento hará comparando esta predicción en contra de Y'_{train}.

Cuando recibamos un nuevo conjunto de datos X_{test}, podemos imputar datos ingenuamente y obtener X'_{test}. Con las funciones f y g lograremos hacer una predicción para los valores faltantes. Si tenemos un conjunto Y_{test} entonces se puede estudiar el sobre-ajuste de este método.

En algunos casos estos métodos nos dan mejores resultados que los métodos clásicos que hemos mencionado inicialmente. No hemos estudiado teóricamente este algoritmo propuesto sin embargo es un excelente ejemplo de cómo se pueden aprovechar a las redes neuronales profundas.

05 Complementos sobre entrenamiento de redes neuronales profundas

En esta sección presentamos algunos complementos del curso que trataremos en los últimos días de la semana.

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.

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

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

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 $S = \left\{(x_i, y_i)\right\}_{i \leq N}$ 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

$$l(m) = err_S(m) = \frac{1}{N} \cdot \sum_{i \leq N} \left(mx_i - y_i\right)^2 \tag{05.1}$$

Exercise 01.1. *Calcule la derivada de l en 05.1.*

Definition 01.2. Si $l : \mathbb{R}^d \to \mathbb{R}$ 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:

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

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 $l : \mathbb{R}^d \to \mathbb{R}$ es una función que entenderemos como la función de error en algún modelo, por ejemplo, podríamos suponer que

$$l(\beta) = \frac{1}{N} \sum_{i \leq N} \left(y_i - \langle \beta, x_i \rangle \right)^2 \tag{05.2}$$

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

Definition 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. $\beta_0 = (1, 1, \ldots, 1)$
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:

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

La idea principal detrás del método del gradiente se podría resumir para el caso uno-dimensional, $l : \mathbb{R} \to \mathbb{R}$, 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.

Remark 02.1. *Si una función es diferenciable un punto, entonces alrededor de ese punto se tiene la siguiente* ***aproximación lineal****: Si* β *es cercana a* β_t*, entonces*

$$l(\beta) \sim l(\beta_t) + \langle \beta - \beta_t, \nabla l(\beta_t) \rangle \tag{05.3}$$

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

$$f(\beta) = \frac{1}{N} \sum_{i \leq N} f_i(\beta) \tag{05.4}$$

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

Es inmediato que el gradiente de f es

$$\nabla f(\beta) = \frac{1}{N} \sum_{i \leq N} \nabla f_i(\beta) \tag{05.5}$$

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

1. $\beta_0 = (1, 1, \ldots, 1)$
2. $\beta_{t+1} = \beta_t - \nu \nabla f_j(\beta_i)$

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

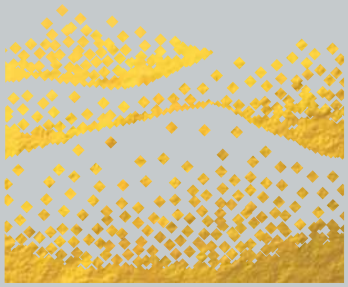
Supongamos que elegimos j de manera aleatoria y uniforme en $\{1, 2, 3, \ldots, N\}$. Es claro que β_{i+1} será una variable aleatoria que depende de j. Calculemos su esperanza:

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

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.

06 Caso de uso: segmentación de clientes

En la actualidad las compañías registran una gran cantidad de características de sus clientes, lo cual tiene un gran beneficio pues podemos utilizar mucha información sobre ellos dependiendo de lo que se desee predecir. El objetivo del caso de uso que presentaremos esta semana es utilizar a las redes neuronales profundas, en particular a los autoencoders para reducir la dimensión de los vectores con los que representamos a un conjunto de usuarios de tarjetas de crédito. Una vez que hayamos reducido la dimensión vamos a agrupar a nuestros clientes por medio de un algoritmo no-supervisado llamado K-Means. Es importante mencionar que este algoritmo difícilmente daría buenos resultados debido a la maldición de la dimensión.



escuela-bourbaki.com