Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Departamento de Matemática

FACTORIZACIÓN MATRICIAL PARA SISTEMAS DE RECOMENDACIÓN

Luciano Miguel Delgado Cota
Asesorado por Ing. Edgar Damián Ochoa Hernández

Guatemala, 25 de noviembre de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

FACTORIZACIÓN MATRICIAL PARA SISTEMAS DE RECOMENDACIÓN

TRABAJO DE GRADUACIÓN
PRESENTADO A LA JEFATURA DEL
DEPARTAMENTO DE MATEMÁTICA
POR

LUCIANO MIGUEL DELGADO COTA
ASESORADO POR ING. EDGAR DAMIÁN OCHOA HERNÁNDEZ

AL CONFERÍRSELE EL TÍTULO DE
LICENCIADO EN MATEMÁTICA APLICADA

GUATEMALA, 25 DE NOVIEMBRE DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

CONSEJO DIRECTIVO INTERINO

Director	M.Sc. Jorge Marcelo Ixquiac Cabrera
Representante Docente	Arqta. Ana Verónica Carrera Vela
Representante Docente	M.A. Pedro Peláez Reyes
Representante de Egresados	Lic. Urías Amitaí Guzmán García
Representante de Estudiantes	Elvis Enrique Ramírez Mérida
Representante de Estudiantes	Oscar Eduardo García Orantes
Secretario	Ing. Edgar Damián Ochoa Hernández

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

Director	M.Sc. Jorge Marcelo Ixquiac Cabrera
Examinador	Lic. Billy Othmaro Quevedo
Examinador	Lic. Ronald Oliverio Chubay
Examinador	Lic. Frank Jorge Fritzsche Barrios
Secretario	Ing. Edgar Damián Ochoa Hernández

Ref. D.DTG. 008-2022
Guatemala 25 de noviembre de 2022

El Director de la Escuela de Ciencias Físicas y Matemáticas de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Coordinador de la Licenciatura en Matemática Aplicada, al trabajo de graduación titulado: "FACTORIZACIÓN MATRICIAL PARA SISTEMAS DE RECOMENDACIÓN", presentado por el estudiante universitario Luciano Miguel Delgado Cota, autoriza la impresión del mismo.

IMPRÍMASE.

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA · ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS · DIRECCIÓN · USAC-ECFM ·

M.Sc. Jorge Marcelo Ixquiac Cabrera
Director

Ref. 123
JMIC/Paola

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$\|\cdot\|$	Norma de un vector.
$x \cdot y$	Producto punto entre x e y.
$\mathcal{L}(V)$	Conjunto de operadores en V.
$\|\cdot\|^2$	Norma de Frobenius.
$\mathbf{F}$	Conjunto sobre el cual se define un campo vectorial, puede ser $\mathbb{R}$ o $\mathbb{C}$.
$\text{span}(v_1, ..., v_m)$	Conjunto generado por las combinaciones lineales de los vectores $(v_1, ..., v_m)$.
λ	Autovalor de un operador.
$\nabla f(x)$	Gradiente de $f(x)$.
$E[X]$	Valor esperado de la variable aleatoria X.
$\text{Var}[X]$	Varianza de la variable aleatoria X.
$\Sigma = \text{Cov}(\mathbf{X})$	Matriz de covarianza del vector aleatorio $\mathbf{X} = (X_1, ..., X_n)$.
(V, E, f)	V denota el conjunto de neuronas, E el conjunto de aristas dirigidas, y f la función de activación.
$\mathcal{C}_{(V,E,f)}$	Conjunto de todas las redes neuronales cuya arquitectura es (V, E, f).
$\mathcal{L}_p$	Función de pérdida.
$\odot$	Producto de Hadamard
$\mathcal{O}$	Notación O-grande.
χ	Factor de regularización.

OBJETIVOS

General

Implementar y entrenar un modelo de factorización de matrices con un factor de regularización y restricciones por términos de pesos y sesgos para un recomendador Top-N.

Específicos

1. Comparar el comportamiento de la función de pérdida del modelo entrenado con diferentes algoritmos de optimización.
2. Explicar las ventajas de los modelos de factores latentes respecto a otros métodos de reducción de la dimensionalidad.
3. Mostrar las bases matemáticas utilizadas en los modelos y técnicas de optimización en aprendizaje profundo.

INTRODUCCIÓN

Los sistemas de recomendación son poderosas herramientas de filtrado de información que pueden facilitar servicios personalizados y proporcionar una experiencia personalizada a usuarios individuales, en resumen, éstos juegan un papel fundamental en la utilización de la gran cantidad de datos disponibles para tomar decisiones [1]. Hoy en dia dichas herramientas son el núcleo de una serie de servicios en línea tales como Netflix, Amazon o Spotify [3].

El **filtrado colaborativo** (FC) es una técnica utilizada para sistemas de recomendación y solo usa los datos de interacción usuario-elemento que se pueden ordenar en una matriz de calificaciones, cuya entrada (i, j) es la calificación asignada por el usuario i al elemento j. En general, las técnicas de filtrado colaborativo se pueden clasificar en: FC basado en memoria, FC basado en modelo y su híbrido. Los modelos de factores latentes, como la factorización de matrices, son ejemplos de FC basados en modelos.

Los enfoques colaborativos puros toman una matriz de calificaciones dadas de usuario-elemento como la única entrada y normalmente producen los siguientes tipos de salida: (a) una predicción (numérica) que indica qué medida le gustará o disgustará al usuario actual un determinado elemento y (b) una lista de n artículos recomendados. La idea principal es explora la información sobre el comportamiento pasado de usuarios para predecir que elementos probablemente le gustarán.

La factorización de matrices también es un método de reducción de la dimensionalidad, como el análisis de componentes principales, pero con ciertas ventajas como la predicción de valores (entradas en la matriz de calificaciones) y de no necesitar trabajar en una matriz de covarianza.

El modelo de factoriación a estudiar se implementa en una red neuronal y se entrena con los algoritmos de optimización de descenso del gradiente estocástico y Adam para encontrar los mejores tiempos de ejecución.

El primer capítulo pretende resumir de la forma más sistematizada posible los conceptos sobre álgebra lineal, cálculo, estadística y teoría de la optimización que

serán utilizados en los siguientes capítulos con el desarrollo de temas como análisis de componentes principales (ACP) o los métodos para optimizar funciones de pérdida en el entrenamiento de redes neuronales. El capítulo dos introduce las ideas básicas sobre aprendizaje de máquina, el tipo de aprendizaje de los algoritmos que pueden ser supervisados o no supervisados, y modelos de evaluación para dichos algoritmos. Posteriormente se desarrolla la teoría necesaria para implementar un optimizador de una red neuronal, empezando desde la definición de perceptrón, las clases de redes neuronales hasta el algoritmo de propagación hacia atrás. Finalmente en el capítulo tres se introducen los conceptos de sistemas de recomendación, filtrado colaborativo y factorización de matrices, partiendo de los métodos de reducción de dimensionalidad para éste último, para implementar un modelo de factorización con factor de regularización, permite al modelo converger durante el entrenamiento, y términos de sesgo para usuario y elemento. También se comparan sus diferencias y ventajas respecto al análisis de componentes principales, otro método de reducción de la dimensionalidad.

1. CONCEPTOS PRELIMINARES

En este capítulo se introduce la teoría matemática sobre la cual se respaldan los siguientes capítulos para describir los principales conceptos de aprendizaje profundo y reducción de la dimensionalidad como los modelos utilizados para factorizar de matrices. Los elementos de álgebra lineal se tomaron de [10], los de cálculo de [4], para estadística se utilizó [4, 7, 9] y la última sección de optimización de [4, 8].

1.1. Álgebra lineal

1.1.1. Espacios vectoriales

En esta sección se entenderá a $\mathbb{F}$ como el conjuntos sobre el cual se definirá un espacio vectorial. Este puede ser $\mathbb{R}$ el campo de los números reales o $\mathbb{C}$ el campo de los números complejos.

Definición 1.1. Sea V un conjunto no vacío, definimos dos operaciones:

- **Adición** en V es una función que asigna un elemento $u + v \in V$ a cada par de elementos $u, v \in V$.
- **Multiplicación** por escalar en V es una operación que asigna un elemento $\lambda v \in V$ a cada $\lambda \in \mathbb{F}$ y cada $v \in V$.

Definición 1.2. Un **espacio vectorial** es un conjunto V junto con una adición y una multiplicación por escalar en V tal que las siguientes propiedades son ciertas

conmutación: $u + v = v + u$ para toda $u, v \in V$;

asociatividad: $(u + v) + w = u + (v + w)$ y $(ab)v = a(bv)$ para toda $v \in V$ y toda $a, b \in \mathbb{F}$;

identidad aditiva: existe un elemento $0 \in V$ tal que $v + 0 = v$ para $v \in V$;

inverso aditivo: para cada $v \in V$, existe $w \in V$, existe $w \in V$ tal que $v + w = 0$;

identidad multiplicativa: $1v = v$ para toda $v \in V$;

propiedades distributivas: $a(u + v) = au + av$ y $(a + b)v = av + bv$ para toda $a, b \in \mathbf{F}$ y toda $u, v \in V$.

Se acostrumbra llamar a los elementos de un espacio vectorial vectores o puntos.

Definición 1.3. Sea V un espacio vectorial sobre $\mathbb{F}$

- Si $\mathbb{F} = \mathbb{R}$ es llamado un espacio vectorial real.
- Si $\mathbb{F} = \mathbb{C}$ es llamado un espacio vectorial complejo.

Definición 1.4. Un subconjunto U de V es llamado un **espacio vectorial** de V si U también es un **espacio vectorial** (usando la misma adición y multiplicación por escalar que en V).

Definición 1.5. Una **combinación lineal** de una lista $v_1, ..., v_m$ de vectores en V es un vector de la forma

$$a_1v_1 + \cdots + a_mv_m$$

donde $a_1, ..., a_m \in \mathbb{F}$.

Definición 1.6. El conjunto de todas las combinaciones lineales de una lista de vectores $\{v_1, ..., v_m\}$ en V es llamado el subespacio generado por $\{v_1, ..., v_m\}$, denotado por span$\{v_1, ..., v_m\}$. Es decir,

$$\text{span}\{v_1, ..., v_m\} = \{a_1v_1 + \cdots a_mv_m; a_1, ..., a_m \in \mathbb{F}\}.$$

El span de una lista vacía $\{\}$ es definida como $\{0\}$.

Definición 1.7. Una lista $\{v_1, ...v_m\}$ de vectores en V es linealmente dependiente si existe $a_1, ...a_m \in \mathbb{F}$, no todos 0, tal que $a_1v_1 + \cdots + a_mv_m = 0$.

De esto se tiene la condición de independencia lineal: una lista de vectores es linealmente independiente cuando $a_1v_1 + \cdots + a_mv_m = 0$ implica que $a_1 = ... = a_m = 0$.

Proposición 1.8. *Suponga que $\{v_1, ..., v_m\}$ es una lista linealmente dependiente en V. Entonces existe $j \in \{1, 2, ..., m\}$ tal que lo siguiente es cierto:*

- $v_j \in \text{span}\{v_1, ..., v_{j-1}\}$;
- si el j-ésimo término es removido de $\{v_1, ..., v_m\}$, el span de la lista restante es igual a $\text{span}\{v_1, ..., v_m\}$.

Demostración. Ya que la lista $\{v_1, ..., v_m\}$ es linealmente dependiente, existen números $a_1, ..., a_m \in \mathbb{F}$, no todos 0, tal que

$$a_1 v_1 + \cdots a_m v_m = 0.$$

Sea j el elemento más grande de $\{1, ..., m\}$ tal que $a_j \neq 0$. Entonces

$$v_j = -\frac{a_1}{a_j} v_1 - \cdots - \frac{a_{j-1}}{a_j} v_{j-1}, \tag{1.1}$$

esto prueba el primer enunciado.

Para probar el segundo, suponga $u \in \text{span}\{v_1, ..., v_m\}$. Entonces existen números $c_1, ..., c_m \in \mathbb{F}$ tal que

$$u = c_1 v_1 + \cdots + c_m v_m.$$

En la ecuación de arriba, se puede reemplazar v_j con el lado derecho (1.1), esto muestra que u esta en el span de la lista obtenida de remover el j-ésimo término de $\{v_1, ..., v_m\}$. □

Definición 1.9. Un **mapeo lineal** de V a W es una función $T\colon V \to W$ con las siguientes propiedades:

aditividad $T(u + v) = Tu + Tv$ para toda $u, v \in V$;

homogeneidad $T(\lambda v) = \lambda(Tv)$ para toda $\lambda \in \mathbb{F}$ y toda $v \in V$.

Los mapeos lineales de un espacio vectorial a si mismos resultan muy importantes que reciben nombre propio y se denotan como $\mathcal{L}(V, W)$, el conjunto de aplicaciones lineales de V a W.

Definición 1.10.

- Un mapeo lineal de un espacio vectorial a si mismo es llamado operador.
- La notación $\mathcal{L}(V)$ denota el conjunto de operadores en V. En otras palabras, $\mathcal{L}(V) = \mathcal{L}(V, V)$.

Una lista de vectores linealmente independientes que son capaces de generar determinado espacio vectorial también recibe un nombre especial.

Definición 1.11. Una **base** de V es una lista de vectores $v_1, ..., v_n$ en V que es linealmente independiente tal que $\text{span}(v_1, ..., v_n) = V$.

Si $v_1, ..., v_n$ es una base de V y $T\colon V \to W$ es lineal, entonces los valores de $Tv_1, ..., Tv_n$ determinan los valores de T en vectores arbitrarios en V, consultar [10]. Las matrices son usadas como un método eficiente para registrar los valores de $Tv_j's$ en términos de una base de W, consultar [10].

Definición 1.12. Sean m y n enteros positivos. Una matriz A de $m \times n$ es un arreglo rectangular de elementos de $\mathbb{F}$ con m filas y n columnas:

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}.$$

La notación $a_{j,k}$ indica la entrada en la fila j y columna k de A.

1.1.2. Vectores y valores propios

Definición 1.13. Suponga que $T \in \mathcal{L}(v)$. Un número $\lambda \in \mathbb{F}$ es llamado **valor propio** de T si existe $v \in V$ tal que $Tv = \lambda v$.

Definición 1.14. Suponga que $T \in \mathcal{L}(V)$ y $\lambda \in \mathbb{F}$ es un valor propio de T. Un vector $v \in V$ es llamado un **vector propio** de T correspondiendo a λ si $v \neq 0$ y $Tv = \lambda v$.

1.1.3. Productos internos y normas

Para motivar el concepto de producto interno, considere vectores en $\mathbb{R}^2$ y $\mathbb{R}^3$ como flechas con punto inicial en el origen. La longitud de un vector $x \in \mathbb{R}^2$ o $x \in \mathbb{R}^3$ es llamada la **norma** de x, denotado por $\|x\|$. Así para $x \in \mathbb{R}^2$, $\|x\| = \sqrt{x_1^2 + x_2^2}$. Similarmente, si $x \in \mathbb{R}^3$, $\|x\| = \sqrt{x_1^2 + x_2^2 + x_3^2}$.

Incluso aunque no sea posible dibujar imágenes en dimensiones superiores, la generalización a $\mathbb{R}^n$ es:

$$\|x\| = \sqrt{x_1^2 + ... + x_n^2}.$$

La norma no es lineal en $\mathbb{R}^2$, pero el producto punto si lo es.

Definición 1.15. Un **producto interno** en V es una función que toma cada par ordenado (u, v) de elementos de V a un número $\langle u, v\rangle \in \mathbb{F}$ y tiene las siguientes propiedades

positividad
$\langle v, v\rangle \geq 0$ para todo $v \in V$;

definida
$\langle v, v\rangle = 0$ si y solo si $v = 0$;

aditividad en primera entrada
$\langle u + v, w\rangle = \langle u, w\rangle + \langle v, w\rangle$ para toda $u, v, w \in V$;

homogeneidad en la primera entrada
$\langle \lambda u, v\rangle = \lambda \langle u, v\rangle$ para toda $\lambda \in \mathbb{F}$ y toda $u, v \in V$;

simetría conjugada
$\langle u, v\rangle = \overline{\langle u, v\rangle}$ para toda $u, v \in V$.

En $\mathbb{R}^n$ el producto interno de dos vectores x e y denotado por $x \cdot y$ es

$$x \cdot y = x_1 y_1 + ... + x_n y_n$$

y se llama producto punto, donde $x = (x_1, ..., x_n)$ y $y = (y_1, ..., y_n)$.

Definición 1.16. Para $v \in V$, la norma de v, denotada $\|v\|$, es definida por

$$\|v\| = \sqrt{\langle v, v\rangle}.$$

1.2. Cálculo

1.2.1. Derivadas y diferenciación

Ahora se procede a introducir los conceptos referentes al cálculo, empezando por la derivada y cómo se calcula.

Definición 1.17. Sea $f : \mathbb{R} \to \mathbb{R}$. La derivada de f es definida como

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} = \frac{df}{dx},$$

si este límite existe.

Si $f'(a)$ existe, entonces se dice que f es **diferenciable** en a. Si f es diferenciable en cada número de un intervalo, entonces esta función es **diferenciable** en este intervalo. Se puede interpretar a la derivada como la tasa instantánea de cambio de $f(x)$ con respecto a x.

1.2.2. Derivadas parciales

Ahora se extiende la definición de derivada en varias dimensiones, es decir, aplicado a funciones multivariantes aparecen las siguientes definiciones.

Definición 1.18. Sea $y = f(x_1, ..., x_n)\colon \mathbb{R} \to \mathbb{R}$ una función con n variables. La derivada parcial de y con respecto al i-ésimo parámetro x_i, con $1 \leq i \leq n$, es

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, ..., x_{i-1}, x_i + h, x_{i+1}, ..., x_n) - f(x_1, ..., x_n)}{h}.$$

Para calcular $\partial y / \partial x_i$, simplemente se tratan el resto de variables $x_1, ..., x_{i-1}$, $x_{i+1}, ..., x_n$ como constantes y calcula la derivada de y con respecto a x_i. Para la notación de derivadas parciales, lo siguiente es equivalente

$$\frac{\partial y}{\partial x_i} = \frac{\partial f}{\partial x_i} = f_{x_i} = f_i = D_i f = D_{x_i} f.$$

1.2.3. Gradientes

Utilizando todas las derivadas parciales de una función puede obtenerse el vector gradiente de la función. El **gradiente** de una función $f(\mathbf{x})$ con respecto a $\mathbf{x} = (x_1, ..., x_n)$ es

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, ..., \frac{\partial f(\mathbf{x})}{\partial x_n} \right],$$

donde $\nabla_{\mathbf{x}} f(\mathbf{x})$ y $\nabla f(\mathbf{x})$ son notaciones equivalentes. Este vector apunta en la dirección donde f crece más rápido.

1.2.4. Regla de la cadena

Cuando el cálculo del gradiente resulta complicado de encontrar, debido a que las funciones utilizadas son frecuentemente compuestas, la regla de la cadena facilita enormemente el cálculo.

Para el caso unidimensional, sea $y = f(u)$ y $u = g(x)$ ambas funciones diferenciables, entonces la regla de la cadena establece

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

Volviendo al caso más general, donde las funciones tienen un número arbitrario de variables. Suponga que la función diferenciable y depende de las variables $u_1, ..., u_m$, donde cada u_i, $1 \leq i \leq n$, es diferenciable y depende de las variables $x_1, ..., x_n$.

Entonces y es una función de $x_1, ..., x_n$, así la regla de la cadena toma la forma

$$\frac{dy}{dx_i} = \frac{dy}{du_1}\frac{du_1}{dx_i} + \cdots + \frac{dy}{du_m}\frac{du_m}{dx_i}$$

para cualquier $i = 1, ..., n$.

1.2.5. Integración

Definición 1.19. Sea $[a, b]$ un intervalo dado. Por **partición** P de $[a, b]$ se entenderá un conjunto finito de puntos $x_0, ..., x_n$ donde

$$a = x_0 \leq x_1 \leq \ldots \leq x_{n-1} \leq x_n = b,$$

se toma $\Delta x_1 = x_i - x_{i-1}$, $1 \leq i \leq n$.

Sea f es una función real **acotada**, es decir, existe $M > 0$ tal que $|f(x)| \leq M$ para cada en $x \in [a, b]$ y considere

$$M_i = \sup\ f(x)\ (x_{i-1} \leq x \leq x_i)$$
$$m_i = \inf\ f(x)\ (x_{i-1} \leq x \leq x_i),$$

donde inf denota la máxima cota inferior y sup la mínima cota superior. Se definen los números

$$U(P, f) = \sum_{i=1}^{n} M_i \Delta x_i$$
$$L(P, f) = \sum_{i=1}^{n} m_i \Delta x_i$$

y apartir de éstos, se define

$$\overline{\int_a^b} f dx = \inf U(P, f), \tag{1.2}$$

$$\underline{\int_a^b} f dx = \sup\ L(P, f) \tag{1.3}$$

Los primeros miembros de (1.2) y (1.3) se llaman integral superior e inferior de Riemann sobre $[a, b]$, se escribe $f \in \mathscr{R}$ y representa el valor común de (1.2) y (1.3)

por

$$\int_a^b f dx. \tag{1.4}$$

1.3. Estadística

1.3.1. Probabilidad y resultados básicos

Antes de pasar directamente al concepto de probabilidad, primero se define un **espacio muestral** $S = \{s_1, ..., s_n\}$ como el conjunto de posibles resultados en un experimento aleatorio, una prueba que consiste en repetir un fenómeno aleatorio con el objetivo de analizarlo y extraer conclusiones sobre su comportamiento. A un subconjunto A de S se le llama **evento**, y sobre el cual se calcula el valor de la probabilidad. Además, la cardinalidad de S puede ser finita o infinita. A cada elemento a de S le llamaremos **punto muestral**.

Definición 1.20. Sea S un espacio muestral, asociado a cierto experimento aleatorio y consistiendo de un conjunto finito de puntos muestrales de tamaño n, cada uno con la misma probabilidad de ocurrir cuando al muestra se lleva a cabo. Entonces la probabilidad de cualquier evento A, consistiendo de m puntos muestrales está dado por $P(A) = m/n$, esta es la definición clásica de probabilidad.

Definición 1.21. Se tiene también un planteamiento axiomático de probabilidad, las siguientes propiedades para que la función P, definida en cada evento del espacio S y tomando valores en la recta real $\mathbb{R}$, sea una función de probabilidad son:

$P1$ $P(A) \geq 0$ para cada evento A (No negatividad de P).

$P2$ $P(S) = 1$ (P está normalizada).

$P3$ Para una cantidad contable de eventos disjuntos a pares A_i ($A_i \cap A_j = \emptyset$, $i = 1, 2..., i \neq j$), se cumple

$$P(A_1 \cup A_2 \cup ...) = P(A_1) + P(A_2) + \cdots ;$$

o

$$P(\cup_{i=1}^n A_i) = \sum_{i=1}^n P(A_i).$$

Teorema 1.22. *Sea S un espacio muestral y P una función de probabilidad. Entonces*

1. $P(\emptyset) = 0$.

2. *Para cada par de eventos disjuntos a pares* $A_1, ..., A_n$, $P(\cup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i)$,

3. *Para cada evento* A, $P(A^c) = 1 - P(A)$. *Aquí* A^c *indica el complemento del conjunto* A, *todos los elementos de* S *que no pertenecen a* A.

4- *Para* A_1 *y* A_2 *eventos tales que* $A_1 \subseteq A_2$, *implica* $P(A_1) \leq P(A_2)$ *y* $P(A_2 - A_1) = P(A_2) - P(A_1)$.

5. $0 \leq P(A) \leq 1$ *para cada evento* A.

6. *Para cualesquiera dos eventos* A_1 *y* A_2*:*

$$P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1 \cap A_2)$$

Demostración.

1. Usando la propiedad (P3) y que $S = S \cup \emptyset \cup \emptyset \cup \ldots$,

$$P(S) = P(S \cup \emptyset \cup \emptyset \cup \ldots) = P(S) + P(\emptyset) + P(\emptyset) + \cdots,$$

   por tanto $P(\emptyset) = 0$.

2. Sea $A_i = \emptyset$ para $i \leq n + 1$, usando (P3) y el resultado anterior

$$P\left(\cup_{i=1}^{n} A_i\right) = P\left(\cup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i) = \sum_{i=1}^{n} P(A_i).$$

3. De (P2) y el resultado anterior, $P(A \cup A^c) = P(S) = 1$ o $P(A) + P(A^c) = 1$, así $P(A^c) = 1 - P(A)$.

4. La relación $A_1 \subseteq A_2$, puede expresarse como $A_2 = A_1 \cup (A_2 - A_1)$, donde $(A_2 - A_1)$ es el conjunto de todos los elementos de A_2 eliminando los que pertenecen a la intersección de A_2 y A_1. Usando el segundo resultado, $P(A_2) = P(A_1) + P(A_2 - A_1)$, se obtiene $P(A_2 - A_1) = P(A_2) - P(A_1)$ y por (P1), se concluye que $P(A_1) \leq P(A_2)$. Si A_1 no está contenida en A_2 entonces $P(A_2 - A_1)$ no es necesariamente $P(A_2) - P(A_1)$.

5. Dado que $\emptyset \subseteq A \subseteq S$ es cierto para cualquier evento A. Entonces por (P1), el primer resultado y el cuarto se tiene $0 = P(\emptyset) \leq P(A) \leq P(S) = 1$.

6. Primero se toma la expresión $A_1 \cup A_2$ como sigue

$$A_1 \cup A_2 = A_1 \cup (A_2 \cap A_1^c) = A_1 \cup (A_2 \cap (S - A_1)) = A_1 \cup (A_2 - A_1 \cap A_2).$$

Entonces, usando los resultados segundo y cuarto:

$$P(A_1 \cap A_2) = P(A_1) + P(A_2 - A_1 \cap A_2) = P(A_1) + P(A_2) - P(A_1 \cap A_2).$$

□

1.3.2. Variables aleatorias y distribuciones de probabilidad

Para un subconjunto B de $\mathbb{R}$, se suele denotar por $(X \in B)$ el siguiente evento en S: $(X \in B)\{= s \in S \mid X(s) \in B\}$ por simplicidad. En particular $(X = x) = \{s \in S; X(s) = x\}$. La función de distribución de probabilidad de una variable aleatoria (v.a) X se denota P_x y es una función de probabilidad definida en subconjuntos de $\mathbb{R}$ como: $P_X(B) = P(X \in B)$. Una v.a X se dice que es **discreta** si existe una cantidad contable de puntos en $\mathbb{R}$, $x_1, x_2, ...$, tal que $P_X(\{x_j\}) > 0$, $j \leq 1$, y $\sum_j P_X(\{x_j\}) = (\sum_j P(X = x)) = 1$. Entonces la función f_x definida en $\mathbb{R}$ por la relación

$$f_x(x_j) = P_X(\{x_j\})(= P(X = x_j)) \text{ para } x = x_j$$

y $f_X(x) = 0$ en cualquier otro punto tiene las propiedades:

$$f_X(x) \leq 0, \text{ para cada } x, \text{ y } \sum_j f_X(x_j) = 1.$$

Además,

$$P(X \in B) = \sum_{x_j \in B} f_X(x_j).$$

Es decir, en vez de calcular la probabilidad del evento $\{s \in S \mid X(s) \in B\}$, únicamente se suman los valores de $f_X(x_j)$ para todos los $x_j s$ los cuales pertenecen a B; asumiendo que la función f_X es conocida. La función f_X es llamada **función de densidad de probabilidad** (f.d.p) de X.

Ahora, suponga que X es una v.a. la cual toma valores en un intervalo I, finito o infinito, en $\mathbb{R}$ con la siguiente condición: $P(X = x) = 0$ para cada $x \in I$, $I \subset \mathbb{R}$. Tal v.a. es llamada una **v.a. continua**. Análogo al caso discreto, una función f_X que satisface las propiedades $f_X(x) \geq 0$ para cada $x \in I$, y $P(X \in J) = \int_J f_X(x)dx$

es llamada función de densidad de probabilidad en el caso continuo.

Desde un punto de vista más formal, la idea que $P(X = x) = 0$ para cada x de una v.a continua puede ser interpretada como $\int_x^x f_X(y)dy$, y esto es cero para cada x. Para comprobar si una función f es una f.d.p únicamente se debe verificar que f es no negativa en ningún punto de su argumento, y que la suma o integral de sus valores es 1.

Cuando una función X está definida en un espacio muestral S y toma valores en un espacio n-dimensional $\mathbb{R}^n$, es llamado un **vector aleatorio** n-dimensional y es denotado por $\mathbf{X}$. Entonces las f.d.p discreta y continua toman la forma:

$$P(\mathbf{X} = \mathbf{x}_j) > 0, j = 1, 2, ..., \text{ con } \sum_j P(\mathbf{X} = \mathbf{x}_j) = 1, \text{ y } P(\mathbf{X} \in B) = \sum_{\mathbf{x}_j \in B} f_{\mathbf{X}}(\mathbf{x}_j)$$

siendo B un subconjunto de $\mathbb{R}^n$, $f_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} = \mathbf{x})$ y $\mathbf{x} = \mathbf{x}_j$ en el caso discreto; por otra parte, el vector aleatorio $\mathbf{X}$ es continuo si $P(\mathbf{X} = \mathbf{x}) = 0$ para cada $\mathbf{x} \in I$, hay una función $f_{\mathbf{X}}$ definida en $\mathbb{R}^n$ tal que

$$f_{\mathbf{X}}(\mathbf{x}) \geq 0 \text{ para cada } \mathbf{x} \in \mathbb{R}^n, \text{ y } P(\mathbf{X} \in I) = \int_I f_{\mathbf{X}}(\mathbf{X})d\mathbf{x}.$$

1.3.3. Función de distribución, probabilidad marginal y condicional

La distribución de un vector aleatorio n-dimensional $\mathbf{X}$ es una función $F\colon \mathbb{R} \to [0, 1]$ y se define como

$$F_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} \leq \mathbf{x})$$

El valor de la distribución se calcula según su tipo

$$F(\mathbf{x}) = P[\mathbf{X} \leq \mathbf{x}] \begin{cases} \sum_{\mathbf{y} \leq x} f(\mathbf{y}) \text{ Caso discreto,} \\ \int_{-\infty}^{\mathbf{x}} f(\mathbf{y})d\mathbf{y} \text{ Caso continuo.} \end{cases}$$

La **función de distribución conjunta** se define como $F(x_1, ..., x_n) = P(X_1 \leq x_1, ..., X_n \leq x_2)$, con $\mathbf{x} = (x_1, ..., x_n)$ y se utiliza para el cálculo de la probabilidad condicional y marginal. Si $F(x_1, ..., x_n)$ tiene derivadas parciales de n-ésimo orden y

$$\frac{\partial^n}{\partial x_1 \cdots \partial x_n} F(x_1, ..., x_n) = f(x_1, ..., x_n)$$

en los puntos de continuidad de f, donde $F_{X_1,...,X_n}$, la cual por simplicidad de ahora

en adelante se expresará como F, es la función de distribución de $\mathbf{X}$.

Definición 1.23. Sea $\mathbf{X} = (X_1, ..., X_n)$ un vector aleatorio con $f(x_1, ..., x_n)$ una función de densidad de probabilidad. La función $f(x_1, ..., x_n)$ se llama **función de densidad de probabilidad conjunta de las v.a.** $X_1, ..., X_n$, puede ser discreta o continua.

Definición 1.24. Sea $\mathbf{X} = (X_1, ..., X_n)$ un vector aleatorio y $f(x_1, ..., x_n)$ una función de densidad de probabilidad, si se suma (o integral) sobre t de las variables $x_1, ..., x_n$ manteniendo s variables fijas, donde $(t + s = n)$, la función resultante es la función de densidad de probabilidad conjunta de las s v.a.

$$f_{i_1,...,i_s}(x_{i_1}, ..., x_{i_s}) = \begin{cases} \sum_{x_{j_1},...,x_{j_t}} f(x_1, ..., x_n) \text{ Caso discreto} \\ \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(x_1, ..., x_n) dx_{j_1} ... dx_{j_t} \text{ Caso continuo.} \end{cases}$$

Tal f.d.p es llamada **f.d.p. marginal**.

Definición 1.25. Sea $\mathbf{X} = (X_1, ..., X_n)$ un vector aleatorio y $f(x_1, ..., x_n)$ una función de densidad de probabilidad, si $x_{j_1}, ..., x_{j_s}$ son tales que $f_{i_1,...,i_t}(x_{i_1,...,x_{i_s}}) > 0$, entonces la función de $x_{j_1,...,j_t}$ es

$$f\left(x_{j_1,...,j_t} \middle| x_{i_1,...,x_{i_s}}\right) = \frac{f(x_1, ..., x_n)}{f_{i_1,...,i_t}(x_{i_1}, ..., x_{i_s})}.$$

Esta función es llamada la **f.d.p condicional conjunta de las v.a**. $X_{j_1}, ..., X_{j_t}$, dado $X_{i_1}, ..., X_{i_s}$.

La **función de distribución condicional** está definida similarmente:

$$F(x_{j_1}, ..., x_{j_t} \mid x_{i_1}, ..., x_{i_s}) = \begin{cases} \sum_{(y_{j_1},...,y_{j_t}) \leq (x_{i_1},...,x_{i_t})} f(y_{j_1}, ..., y_{j_t} \mid x_{i_1}, ..., x_{i_s}) \\ \int_{-\infty}^{x_{j_1}} \cdots \int_{-\infty}^{x_{j_t}} f(y_{j_1}, ..., y_{j_t} \mid x_{i_1}, ..., x_{i_s}) dy_{j_1} \dots dy_{j_t} \end{cases}$$

1.3.4. Esperanza y varianza

Para resumir las características de distribuciones de probabilidad, se necesitan algunas medidas. La esperanza (o promedio) de una variable aleatoria X, para el caso discreto se calcula por

$$E[X] = \sum_x xP(X = x).$$

Cuando la entrada de una función $f(x)$ es una variable de una distribución P con diferentes valores x, la expectación de $f(x)$ es calculada como

$$E[f(x)] = \sum_x f(x)P(x).$$

En muchos casos se quiere medir que tanto una variable aleatoria X se desvía de su valor de expectación. Es posible cuantificar esto utilizando la **varianza** definida como

$$\text{Var}[X] = E[(X - E[X])^2]$$

La raíz cuadrada de la varianza es llamada **desviación estándar**. La varianza de una función de una variable aleatoria mide que tanto la función se desvía del valor de expectación de la función, dependiendo del valor x.

$$\text{Var}[f(x)] = E\left[(f(x) - E[f(x)])^2\right].$$

Si X es un vector aleatorio dado por

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix}$$

tal que la i-ésima entrada del vector X es una variable aleatoria con varianza finita, entonces la matriz de covarianza Σ es una matriz de dimensión $n \times n$ cuya entrada (i, j) es la covarianza entre la variable X_i, X_j, es decir

$$\Sigma_{ij} = \text{Cov}(X_i, X_j).$$

1.4. Optimización

Los problemas de optimización consisten en encontrar los puntos donde se alcanzan los máximos y mínimos de una función objetivo f sobre un dominio $\mathcal{D}$, conjunto sobre el cual se define la función. Dichos puntos se denominan puntos óptimos y pueden ser locales, máximos o mínimos en una vecindad contenida en el conjunto $\mathcal{D}$, o globales, es decir, para todo el dominio. En teoría de la optimización, uno de los principales objetivos es determinar las condiciones necesarias y suficientes en general, para que un punto dado sea un punto óptimo.

El conjunto $\mathcal{D}$ también se denomina conjunto de restricción de la función objetivo y suele definirse como

$$\mathcal{D} = U \cap \{x \in \mathbb{R}^n \mid g(x) = 0, h(x) \geq 0\},$$

donde $U \in \mathbb{R}^n$ es un abierto, $g\colon \mathbb{R}^n \to \mathbb{R}^k$, y $h\colon \mathbb{R}^n \to \mathbb{R}^l$. Las funciones se entenderán como $g = (g_1, ..., g_k)$ **restricciones de igualdad**, y las funciones $h = (h_1, ..., h_l)$ como **restricciones de desigualdad**. Los problemas de optimización que no presentan restricciones de igualdad ni desigualdad se conocen como problemas de optimización sin restricciones, problemas de optimización con restricciones de igualdad si únicamente está g, problemas de optimización con restricciones de desigualdad si únicamente está h y mixtos cuando ambos tipos de restricciones están presentes.

1.4.1. Convexidad y optimización

La convexidad es importante en el diseño de algoritmos de optimización. Se debe ampliamente al hecho que es mucho más fácil analizar y probar algoritmos en este contexto [4].

Definición 1.26. Sea $\mathcal{D} \subset \mathbb{R}^n$, también sean $x, y \in \mathcal{D}$ y toda $\lambda \in (0, 1)$, entonces $\mathcal{D}$ es un **conjunto convexo** si

$$\lambda x + (1 - \lambda)y \in \mathcal{D}.$$

En lo que queda del capítulo, se asumirá que $\mathcal{D}$ es un conjunto convexo. Ahora que ya se definió conjunto convexo, se introduce la definición de función convexa y función cóncava .

Definición 1.27. Una función $f\colon \mathcal{D} \to \mathbb{R}$ es **cóncava** si para todo $x, y \in \mathcal{D}$ y para todo $\lambda \in [0, 1]$, se tiene que

$$f[\lambda x + (1 - \lambda)y] \geq \lambda f(x) + (1 - \lambda)f(y).$$

Por otra parte, $f\colon \mathcal{D} \to \mathbb{R}$ es una **función convexa** si

$$f[\lambda x + (1 - \lambda)y] \leq \lambda f(x) + (1 - \lambda)f(y).$$

Las funciones cóncavas y convexas juegan un rol importante en problemas de optimización. Su significado aparece en problemas con un conjunto de restricción

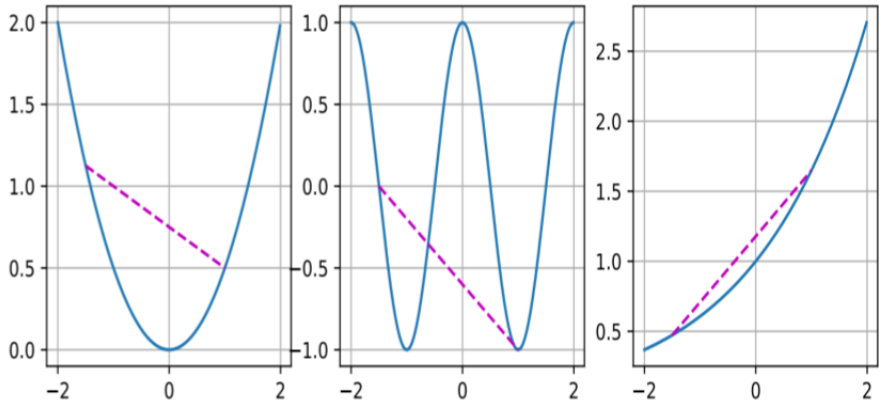


Figura 1.1. Ilustración funciones convexas y no convexas (curvas celestes). Las lineas punteadas permiten visualizar fácilmente las curvas que no son convexas. Tomado de [4].

convexo y una función objetivo cóncava, las condiciones de primer orden son necesarias y suficientes para identificar un máximo global; mientras en problemas con un conjunto de restricción convexo y una función objetivo convexa, las condiciones de primer orden son necesarias y suficientes para identificar un mínimo global [8].

1.4.2. Derivadas y convexidad

El siguiente teorema provee una caracterización completa de la concavidad o convexidad de una función diferenciable en todas partes f usando su primera derivada.

Teorema 1.28. *Sea $\mathcal{D}$ un conjunto abierto y convexo en $\mathbb{R}^n$, y sea $f : \mathcal{D} \to \mathbb{R}$ diferenciable en $\mathcal{D}$. Entonces, f es cóncava en $\mathcal{D}$ si y solo si*

$$Df(x)(y - x) \geq f(y) - f(x) \text{ para cada } x, y \in \mathcal{D},$$

mientras f es convexa en $\mathcal{D}$ si y solo si

$$Df(x) \leq f(y) - f(x) \text{ para cada } x, y \in \mathcal{D}.$$

Demostración. Suponga que f es cóncava, se tiene

$$Df(x)(y-x) = \lim_{t\to 0_+} \frac{f(x+t(y-x))-f(x)}{t} = \lim_{t\to 0_+} \frac{f(ty+(1-t)x)-f(x)}{t}.$$

Cuando $t \in (0,1)$, la concavidad de f implica $f(ty+(1-t)x) \geq tf(y)+(1-t)f(x)$, así escogiendo $t>0$, $t\to 0$, se tiene

$$Df(x)(y-x) \geq \lim_{t\to 0_+} \frac{tf(y)+(1-t)f(x)-f(x)}{t} = f(y)-f(x).$$

Ahora suponga que para cada $x_1, x_2 \in \mathcal{D}$ se define

$$Df(x_1)(x_2-x_1) \geq f(x_2)-f(x_1).$$

Ahora se toman x e y en $\mathcal{D}$, y cualquier $\lambda \in (0,1)$. Por convención se define

$$z = \lambda x + (1-\lambda)y$$

y

$$w = x - z = (1-\lambda)(x-y).$$

De esto, se deduce

$$y = z - \left(\frac{\lambda}{1-\lambda}\right) w.$$

Por hipótesis, también se tiene

$$f(x)-f(z) \leq Df(z)(x-z) = Df(z)w,$$

y

$$f(y)-f(z) \geq Df(z)(y-z) = \left(\frac{\lambda}{1-\lambda}\right) Df(z)w.$$

Multiplicando la primera ecuación por $\lambda/(1-\lambda)$, y sumando las dos ecuaciones, se obtiene

$$\left(\frac{\lambda}{1-\lambda}\right) f(x) + f(y) - \left(\frac{\lambda}{1-\lambda}\right) f(z) \leq 0.$$

Reordenando términos después de multiplicar por $1-\lambda$

$$\lambda f(x) + (1-\lambda)f(y) \leq f(z) = f[\lambda x + (1-\lambda)y],$$

esto completa la prueba. □

1.4.3. Mínimo local de una función convexa

Finalmente se presenta un resultado de la optimización convexa que establece que todos los óptimos deben ser globales.

Teorema 1.29. *Suponga $\mathcal{D} \subset \mathbb{R}^n$ es convexa, y $f\colon \mathcal{D} \to \mathbb{R}$ es cóncava. Entonces,*

1. *Cualquier máximo local de f es un máximo global de f.*
2. *El conjunto* $\arg\max\{f(x) \mid x \in \mathcal{D}\}$ *de maximizadores de f en $\mathcal{D}$ es o vacío o convexo.*

Demostración. Suponga que f tiene un máximo local x que no es un máximo global. Ya que x es un máximo local, hay $r > 0$ tal que $f(x) \geq f(y)$ para cada $y \in B(x, r) \cap \mathcal{D}$. Como x no es un máximo global, hay un $z \in \mathcal{D}$ tal que $f(z) > f(x)$. Al ser $\mathcal{D}$ convexa, $(\lambda x + (1 - \lambda)z) \in \mathcal{D}$ para cada $\lambda \in (0, 1)$. Si se toma un λ lo suficientemente cercano a la unidad para que $(\lambda x + (1 - \lambda)z) \in B(x, r)$. Por concavidad de f

$$f[\lambda x + (1 - \lambda)z] \geq \lambda f(x) + (1 - \lambda)f(z) > f(x),$$

pero por construcción se debe tener $f(x) \geq f[\lambda x+(1-\lambda)z]$, esto es una contradicción. Así se prueba la parte 1.

Para la parte 2, sean x_1 y x_2 maximizadores de f en $\mathcal{D}$. Entonces, se tiene $f(x_1) = f(x_2)$. Además, para $\lambda \in (0, 1)$, se tiene

$$f[\lambda x_1 + (1 - \lambda)x_2] \geq \lambda f(x_1) + (1 - \lambda)f(x_2) = f(x_1),$$

y esto debe ser cierto con la igualdad o x_1 y x_2 podrían no ser maximizadores. □

1.5. Algoritmos de optimización

Los algoritmos de optimización son importantes para el aprendizaje profundo. Por un lado, entrenar un modelo complejo de aprendizaje profundo puede llevar horas, días o incluso semanas. El rendimiento del algoritmo de optimización afecta directamente la eficiencia de entrenamiento del modelo. Por otro lado, comprender los principios de los diferentes algoritmos de optimización y la importancia de sus hiperparámetros permitirá ajustarlos de manera específica para mejorar el rendimiento de los modelos de aprendizaje profundo.

Para un problema de aprendizaje profundo, normalmente se define una función de pérdida. Dicha función se usa en un algoritmo de optimización para intentar minimizar el error para acercarse al óptimo. En optimización, una función de éstas características a menudo se denomina función objetivo del problema de optimización. Por tradición y convención, la mayoría de los algoritmos de optimización se ocupan de la minimización, ver [4].

1.5.1. Descenso del gradiente

En esta sección se introducen los conceptos básicos que subyacen al descenso de gradiente. Aunque este último rara vez se usa directamente en el aprendizaje profundo, su comprensión es clave para comprender los algoritmos de descenso de gradiente estocástico. Asimismo, el preacondicionamiento es una técnica común en el descenso de gradientes y se traslada a algoritmos más avanzados, ver [4].

1.5.1.1. Descenso del gradiente en una dimensión

Trabajar en una dimensión permite ilustrar bastante bien por que el algoritmo del descenso del gradiente se puede reducir al valor de la función objetivo. Considere alguna función de valor real continuamente diferenciable $f: \mathbb{R} \to \mathbb{R}$. Usando una expansión de Taylor. se obtiene que

$$f(x + \epsilon) = f(x) + \epsilon f'(x) + \mathcal{O}(\epsilon^2),$$

donde $\mathcal{O}(\epsilon^2)$, notación O-grande, indica que el valor absoluto del resto de términos en la expansión están acotados por ϵ^2. En primera aproximación $f(x + \epsilon)$ está dado por una función de valor real $f(x)$ y su primera derivada. Recordando que el vector gradiente apunta hacia la dirección en la que el valor de f incrementa más rápidamente, es de esperar que para ϵ moviendose en la dirección del gradiente negativo, f decrezca. Se escoge un tamaño de paso fijo $\eta > 0$ y $\epsilon = -\eta f'(x)$. Reemplazando esto en la expansión de Taylor de arriba

$$f(x - \eta f'(x)) \leq f(x).$$

Quiere decir que, si se sustituye x con $x - \eta f'(x)$, denotado por

$$x \leftarrow x - \eta f'(x),$$

el valor de la función puede decrecer. Por tanto, en el descenso de gradiente, primero se elige un valor inicial x y una constante $\eta > 0$ y luego se utiliza para iterar sobre x hasta que se alcanza la condición de parada, por ejemplo, cuando la magnitud del gradiente $|f'(x)|$ es suficientemente pequeña o el número de iteraciones alcanza cierto valor.

1.5.1.2. Descenso del gradiente multivariante

Ahora que se tiene una mejor intuición del caso univariante, considérese el caso donde $\mathbf{x} \in \mathbb{R}^d$. La función objetivo $f\colon \mathbb{R}^d \to \mathbb{R}$ mapea los vectores en escalares. El gradiente es:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, ..., \frac{\partial f(\mathbf{x})}{\partial x_d}\right]^T.$$

Cada derivada parcial $\partial f(\mathbf{x})/\partial x_i$ en el gradiente indica que la tasa de cambio de f en $\mathbf{x}$ con respecto a la entrada x_i. La correspondiente aproximación de Taylor para las funciones multivariantes es

$$f(\mathbf{x} + \epsilon) = f(\mathbf{x}) + \epsilon^T \nabla f(\mathbf{x}) + \mathcal{O}(\|x\|^2).$$

Para el término de segundo orden en ϵ la dirección de descenso más empinada está dada por el gradiente negativo $-\nabla f(\mathbf{x})$. Escogiendo un paso de aprendizaje adecuado $\eta > 0$ produce el algoritmo de descenso de gradiente prototípico:

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x}).$$

Es importante resaltar que el descenso del gradiente necesita que $f(x)$ sea una función convexa, ya que el mínimo o máximo alcanzado por el algoritmo podría ser un mínimo o máximo local y no global, esto se soluciona utilizando el descenso del gradiente estocástico.

1.5.2. Descenso del gradiente estocástico (DGE)

En aprendizaje profundo, la función objetivo es usualmente promediada de las funciones de pérdida para cada ejemplo en el conjunto de datos de entrenamiento, ver [4]. Se asume que $f_i(\mathbf{x})$ es la función de pérdida del conjunto de entrenamiento con n ejemplos, un índice de i, y vector de parámetro de $\mathbf{x}$, entonces la función

objetivo

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}).$$

El gradiente de la función objetivo en $\mathbf{x}$ es calculado como

$$\nabla f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}).$$

Si el descenso del gradiente es usado, el costo de cómputo para cada iteración de la variable independiente es $\mathcal{O}(n)$, ver [4]. Por lo tanto, cuando el conjunto de datos de entrenamiento del modelo es grande, el costo del descenso de gradiente para cada iteración será muy alto.

El descenso del gradiente estocástico (DGE) reduce el costo computacional en cada iteración. Se muestrea uniformemente un índice $i \in \{1, ..., n\}$ para ejemplos de datos aleatorios, y calcula el gradiente $\nabla f_i(\mathbf{x})$ para actualizar $\mathbf{x}$:

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f_i(\mathbf{x}),$$

η es la tasa de aprendizaje, un hiperparámetro que se presento en los algoritmos de optimización, este hiperparámetro junto al de regularización se explican al final del capítulo 2. Se debe mencionar que el gradiente estocástico $\nabla f_i(\mathbf{x})$ es la estimación imparcial del gradiente $\nabla f(\mathbf{x})$.

$$\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x}). \tag{1.5}$$

En promedio, el gradiente estocástico es una buena estimación del gradiente, ver [4].

2. Elementos de aprendizaje profundo

En este capítulo se introducen los conceptos de aprendizaje de máquina y aprendizaje profundo que con más frecuencia se utilizan. Se hace énfasis en el funcionamiento de una red neuronal multicapa y su definición formal, como actúan las funciones de activación y las funciones de pérdida en la red neuronal. También se presentan los algoritmos de descenso del gradiente y gradiente estocástico (que es una mejora del descenso del gradiente) que junto al algoritmo de propagación hacia atrás permite construir un optimizador para minimizar el valor de la función de pérdida usando una taza de aprendizaje adecuada.

2.1. Conceptos principales de aprendizaje de máquina

Los conceptos presentados a continuación son tomados de [5]. El interés en el aprendizaje de máquina se ha disparado en la última década. El aprendizaje de máquina se ve en programas de ciencias de la computación y conferencias de la industria casi a diario. Fundamentalmente, el aprendizaje de máquina utiliza algoritmos para extraer información de datos sin procesar y representarla en algún tipo de modelo. Dicho modelo se usa para inferir cosas sobre otros datos que aún no se han modelado.

Para definir cómo pueden aprender las máquinas, se define qué se entiende por «aprendizaje». En el lenguaje cotidiano, el término aprender, se refiere a «adquirir conocimientos mediante el estudio, la experiencia o la enseñanza». Afinando un poco el enfoque, se puede pensar en el aprendizaje de máquina como el uso de algoritmos para adquirir descripciones estructurales a partir de ejemplos de datos. Una computadora aprende algo sobre las estructuras que representan la información en los datos sin procesar. Las descripciones estructurales son otro término para los modelos construidos para contener la información extraída de los datos sin procesar, y se pueden usar esas estructuras o modelos para predecir datos desconocidos. Las

descripciones (o modelos) estructurales pueden adoptar muchas formas, incluidas las siguientes:

- Árboles de decisión.
- Regresión lineal.
- Redes neuronales con pesos.

Cada tipo de modelo tiene una forma diferente de aplicar reglas a datos conocidos para predecir datos desconocidos. Los árboles de decisión crean un conjunto de reglas en forma de estructura de árbol y los modelos lineales crean un conjunto de parámetros para representar los datos de entrada.

Las redes neuronales tienen lo que se denomina un vector de parámetros que representa los pesos de las conexiones entre los nodos de la red.

2.2. Funcionamiento del aprendizaje de máquina

En el campo del álgebra lineal, es de interés la resolución de sistemas de ecuaciones lineales de la forma: $Ax = b$ donde A es la matriz de coeficientes o pesos, x el vector de parámetros y b un vector columna de términos constantes. Fundamentalmente, el aprendizaje de máquina se basa en técnicas algorítmicas para minimizar el error, la diferencia entre un valor calculado en comparación a otro de referencia, de la ecuación $Ax = b$ a través de la optimización.

La optimización se enfoca en cambiar los números en el vector de columna x (vector de parámetros) hasta encontrar un buen conjunto de valores que brinde los resultados más cercanos a los valores reales. Cada peso en la matriz de peso se ajustará después de que la función de pérdida, como las presentadas en la sección 2.6, calcule el error producido por la red. Una matriz de error que atribuya una parte de la pérdida a cada peso se multiplicará por los pesos mismos.

2.2.1. Aprendizaje supervisado y no supervisado

Hay dos grandes paradigmas de aprendizaje en que las máquinas pueden entender un conjunto de datos, aprendizaje supervisado y aprendizaje no supervisado. El **aprendizaje supervisado** es el más utilizado e incluye algoritmos como regresión lineal y logístico, clasificación de clases múltiples, máquinas de vectores de soporte, entre otros. Se denomina así, porque el desarrollador actúa como una guía para

enseñar al algoritmo las conclusiones a las que debe llegar, es decir, la salida del algoritmo ya es conocida. Es similar a la forma en que un niño podría aprender de un maestro. Requiere que los posibles resultados del algoritmo ya sean conocidos y los datos utilizados para entrenar el algoritmos ya estén etiquetados con las respuestas correctas.

Por otra parte, el **aprendizaje no supervisado** son el conjunto de técnicas que tratan de inferir modelos a partir de datos, como *k-means* y reglas de asociación, de los cuales se desconoce la salida deseada, es decir, tratan de encontrar características similares entre ellos para realizar una clasificación correcta usando patrones que a simple viste no se perciben. Un ejemplo de esto se da en el mercadeo, donde se han conseguido buenos resultados en la segmentación de clientes, es decir, separación de clientes por hábitos o características para determinar la publicidad más apropiada.

2.2.2. Regresión

La regresión se refiere a funciones que intentan predecir una salida de valor real. Este tipo de función estima la variable dependiente conociendo la variable independiente. La clase de regresión más común es la regresión lineal, en el modelado de sistemas de ecuaciones lineales. La regresión lineal intenta generar una función que describa la relación entre x e y y, para valores conocidos de x, predice valores de y que sean precisos.

La predicción de un modelo de regresión lineal es la combinación lineal de coeficientes (del vector de parámetros x) y luego variables de entrada (características del vector de entrada). Se puede modelar usando la siguiente ecuación:

$$y = a + Bx \tag{2.1}$$

donde a es el intercepto en y, B es la característica de entrada, y x el vector de parámetro.

La ecuación expandida es

$$y = a + b_0^* x_0 + b_1^* x_1 + \ldots + b_n^* x_n.$$

Un ejemplo simple de un problema que resuelve la regresión lineal sería predecir gastos mensuales en gasolina en función de la duración de un viaje. Aquí, lo que paga por llenar el tanque es una función de la distancia que conduce. El costo de la gasolina es la variable dependiente y la duración del viaje es la variable independiente. Es

razonable hacer un seguimiento de estas dos cantidades y luego definir una función de la forma:

$$costo = f(distancia).$$

La relación permite predecir razonablemente el gasto de gasolina en función del kilometraje. En este ejemplo, la distancia es la variable independiente y el costo la variable dependiente del modelo f.

Aquí hay algunos otros ejemplos de modelos de regresión lineal:

- Predicción del peso en función de la altura
- Predecir el precio de venta de una casa en función de sus pies cuadrados.

Se puede representar la regresión lineal como encontrar una línea que mejor se ajuste a tantos puntos como sea posible en un diagrama de dispersión de datos. Ajustar es definir una función $f(x)$ que produce valores de y cercanos a los valores de y medidos del mundo real.

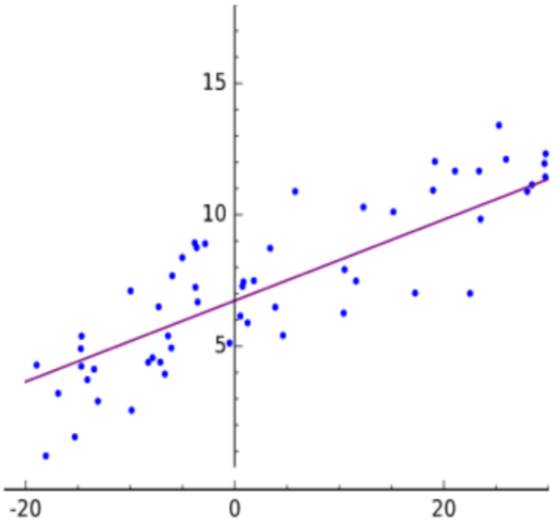


Figura 2.1. Ilustración de una regresión lineal (linea roja) para un conjunto de datos (puntos azules) en el plano cartesiano. Tomado de [5].

2.2.3. Clasificación

La clasificación es un modelo basado en la delineación de clases de salida en función de algún conjunto de características de entrada. Si la regresión da un resultado

de «cuánto», la clasificación da un resultado de «qué tipo». La variable dependiente y es categórica en lugar de numérica. La forma más básica de clasificación es un clasificador binario que tiene una única salida con dos etiquetas (dos clases: 0 y 1, respectivamente). Ejemplos de clasificación binaria incluyen: clasificaciones.

- Clasificación de una persona enferma o no enferma.
- Clasificación de un correo electrónico como spam o no spam.
- Clasificación de una transacción como fraudulenta o nominal.

Más allá de dos etiquetas, se puede tener modelos de clasificación que tengan N etiquetas donde se clasificaría cada una de las etiquetas de salida, y luego la etiqueta con la puntuación más alta es la etiqueta de salida. Se discutirá más adelante cuando se aborden las redes neuronales con múltiples salidas y su arquitectura.

2.2.4. Modelos de evaluación

La evaluación de modelos es el proceso de comprender qué tan bien dan la clasificación correcta y luego medir el valor de la predicción en un contexto determinado. A veces, solo importa la frecuencia con la que un modelo obtiene una predicción correcta; otras veces, es importante que el modelo obtenga cierto tipo de predicción correcta con más frecuencia que las demás. La herramienta básica para evaluar modelos: la matriz de confusión.

2.2.4.1. Matriz de confusión

La **matriz de confusión** es una tabla que representa las predicciones y las salidas actuales para un clasificador. Se usa esta tabla para entender mejor como el modelo o clasificador está rindiendo basado en dar la respuesta correcta en el momento apropiado.

Las medidas de las respuestas se obtienen contando lo siguiente:

- Verdaderos positivos (VP)

  Predichos positivos.
  Etiqueta positiva.

- Falsos positivos (FP)

  Predichos negativos.
  Etiqueta negativa.

- Verdaderos negativos (VN)

  Predichos positivo.
  Etiqueta negativa.

- Falsos negativos (FN)

  Predicho negativo.
  Etiqueta positiva.

En estadística tradicional un falso positivo también se conoce como error tipo I y un falso negativo se conoce como error tipo II, consultar [11]. Mediante el seguimiento de estas métricas, se lograr un mayor análisis detallado sobre el rendimiento del modelo más allá del porcentaje básico de conjeturas que eran correctas, ver [2]. Se pueden calcular diferentes evaluaciones del modelo en base a combinaciones de los cuatro recuentos de la matriz de confusión.

Sensibilidad y especificidad. La sensibilidad y especifidad son dos medidas diferentes de un modelo de clasificación binaria. La taza de verdaderos positivos mide como se clasifican las entradas en la clase positiva y su correcta clasificación. Esto también es llamado **sensibilidad**, o *recall*; un ejemplo sería clasificar a un paciente con una condición de la cual actualmente está enfermo. La sensibilidad cuantifica como el modelo evita falsos negativos

$$\text{sensibilidad} = VP/(VP + FN).$$

Si el modelo clasificara a un paciente del ejemplo anterior como que no tiene la afección y en realidad no la tenía, entonces esto se consideraría un verdadero negativo (también llamado especificidad). La **especificidad** cuantifica qué tan bien el modelo evita los falsos positivos.

$$\text{especificidad} = VN/(VN + FP).$$

Muchas veces es necesario evaluar el compromiso entre sensibilidad y especificidad. Un ejemplo sería tener un modelo que detecte enfermedades graves en pacientes con mayor frecuencia debido al alto costo de diagnosticar erróneamente a un paciente realmente enfermo. Si ahora este modelo tiene baja especificidad. Una enfermedad grave puede suponer un peligro para la vida del paciente y de los que le rodean, por lo que se consideraría que el modelo tiene una alta sensibilidad a esta situación

y sus implicaciones. En un mundo perfecto, el modelo sería 100 % sensible y 100 % específico.

Exactitud y precisión La **precisión** es el grado de cercanía de las mediciones de una cantidad al verdadero valor de dicha cantidad.

$$\text{exactitud} = (VP + VN)/(VP + FP + FN + VN),$$

la precisión puede ser engañosa en la calidad del modelo cuando el desequilibrio de clases es alto. Si simplemente se clasifica todo como la clase más grande, el modelo automáticamente obtendrá una gran cantidad de sus conjeturas correctas y proporcionará una puntuación de alta precisión pero una indicación de valor engañosa basada en una aplicación real del modelo [5].

El grado en que las mediciones repetidas bajo las mismas condiciones dan los mismos resultados se llama precisión en el contexto de la ciencia y la estadística. La **precisión** es también conocida como el valor de predicción positivo. Aunque a veces se usa indistintamente con «exactitud» en el uso coloquial, los términos se definen de manera diferente en el marco del método científico,

$$\text{precisión} = VP/(VP + FP).$$

Una medición puede ser exacta pero no precisa, no exacta pero aún así precisa, ni exacto ni preciso, o tanto exacto como preciso. Se considera una medida como válido si es exacta y precisa.

2.3. Principios generales de redes neuronales

Las redes neuronales son un modelo computacional que comparte algunas propiedades con el cerebro animal en el que muchas unidades simples funcionan en paralelo sin una unidad de control centralizada. Los pesos entre las unidades son el principal medio de almacenamiento de información a largo plazo en las redes neuronales. La actualización de los pesos es la forma principal en que la red neuronal aprende nueva información.

Cuando se mencionó el modelo $Ax = b$. En el contexto de las redes neuronales, la matriz A sigue siendo los datos de entrada y el vector de la columna b sigue siendo las etiquetas o los resultados de cada fila de la matriz A. Los pesos en las conexiones de la red neuronal se convierten en x (el vector de parámetros).

El comportamiento de las redes neuronales está determinado por su arquitectura de red, la que puede ser definida (en parte) por:

- Número de neuronas.
- Número de capas.
- Tipos de conexiones entre capas.

La red neuronal más conocida y más simple de entender es la red neuronal multicapa *dirigida hacia adelante*. Tiene una capa de entrada, una o varias capas ocultas y una sola capa de salida. Cada capa puede tener un número diferente de neuronas y cada capa está completamente conectada a la capa adyacente, en la subsección 2.3.3 se definirán con mayor formalidad.

Una red neuronal multicapa dirigida hacia adelante puede representar cualquier función, dadas suficientes unidades de neuronas artificiales, [5]. Por lo general, se entrena mediante un algoritmo de aprendizaje llamado aprendizaje propagación hacia atrás. La propagación hacia atrás utiliza un descenso de gradiente en los pesos de las conexiones en una red neuronal para minimizar el error en la salida de la red, [4].

Históricamente, la propagación hacia atrás se ha considerado lenta, pero los avances recientes en el poder computacional a través del paralelismo y las unidades de procesamiento de gráficos (GPU) han renovado el interés en las redes neuronales, [5].

2.3.1. El perceptrón

Las redes neuronales son una caja de herramientas muy popular para abordar problemas de aprendizaje supervisado, desde tareas de procesamiento de lenguaje natural hasta tareas de visión artificial, [5, 4]. El perceptrón es un modelo que imita ampliamente el funcionamiento de una neurona biológica: un perceptrón recibe una señal, realiza un conjunto de operaciones, generando un resultado, y finalmente hay un discriminador que decide si se activa o no en base a dicho resultado. El modelo del perceptrón se ilustra en la figura 2.2.

El perceptrón toma un vector de parámetros como entradas $x = (x_1, ..., x_n)$ y las salidas son 0 o 1. La activación se determina usando un conjunto de pesos $\{\omega_1, ..., \omega_n\}$ y un sesgo denotado por b, acorde a

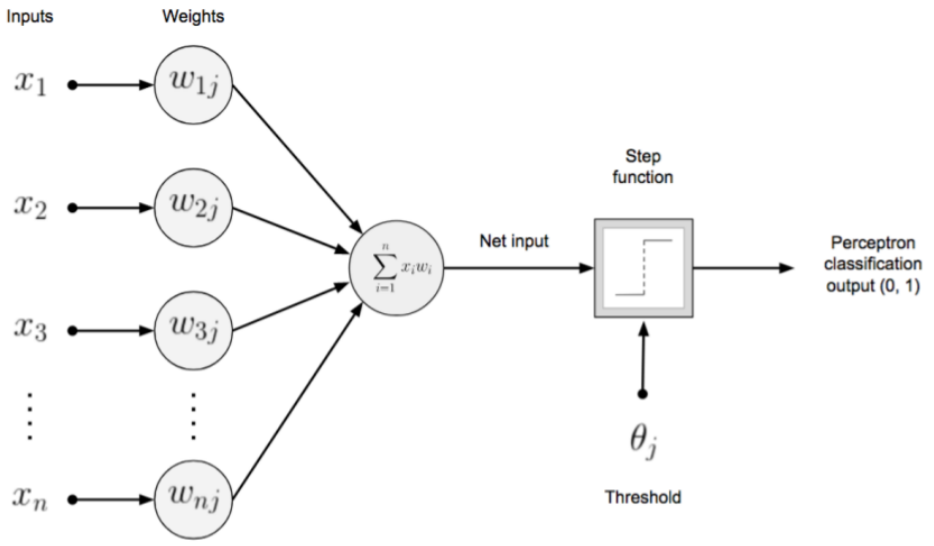


Figura 2.2. Estructura del perceptrón, red neuronal de una sola capa. Tomado de [5].

$$\begin{cases} 0, \text{ si } \omega \cdot x + b \leq 0 \\ 1, \text{ si } \omega \cdot x + b > 0. \end{cases}$$

Al ajustar cuidadosamente los pesos y el sesgo, el perceptrón atribuye a un conjunto dado de valores de entrada 0 o 1. Esta explicación resalta del hecho que un perceptrón puede entenderse como un clasificador lineal, y que el límite de decisión correspondiente viene dado por la ecuación

$$\langle \omega, x \rangle + b = 0.$$

A un lado de este límite, la salida del perceptrón es 1; en el otro lado, incluido el propio límite, la salida es 0. En $\mathbb{R}^n$, el límite de decisión corresponde a un hiperplano. Siguiendo con esto, se puede notar que los perceptrones son capaces de resolver problemas linealmente separables [5, 11]. Pero los perceptrones se pueden conectar fácilmente al considerar las salidas de algunos como las entradas de otros, creando así una red de perceptrones, una red neuronal. La esperanza es entonces que tal red sea capaz de resolver problemas más difíciles que los linealmente separables.

2.3.2. Perceptrón multicapa

Los perceptrones tienen salidas binarias, como se vio anteriormente. Sin embargo, es muy dudoso que cualquier aprendizaje pueda ocurrir en tal entorno. Supóngase

que se tiene un proceso de aprendizaje capaz de ajustar cuidadosamente los pesos y sesgos de una red neuronal, si los pesos y sesgos de un perceptrón dado en la red son tales que $\omega \cdot x + b \approx 0$, incluso un pequeño cambio en w o b es muy probable que haga que el perceptrón invierta su decisión y, por lo tanto, cambie completamente la decisión de toda la red. En consecuencia, es necesario introducir el concepto de funciones de activación. En lugar de salidas binarias, considere ahora que las salidas de las neuronas se calculan como $f(\omega \cdot x + b)$ donde f es una función no lineal dada; f necesita ser no lineal, de lo contrario, una red neuronal de tales unidades se reduciría a una sola capa de neuronas, ya que el conjunto de funciones lineales se cierra bajo composición [5, 11]. El diferencial de la función de activación muestra que esta nueva salida es tal que

$$df = \sum_j \frac{\partial f}{\partial \omega_j} d\omega_j + \frac{\partial f}{\partial b} db.$$

En consecuencia, un pequeño cambio en los pesos o el sesgo provoca un pequeño cambio en la salida, proporcionalmente a la derivada parcial de la cantidad modificada. Históricamente, la primera función de activación considerada fue la función **sigmoidea** $\sigma(x) := (1 + e^{-x})^{-1}$, [11]. Ya que un perceptrón es una neurona cuya función de activación es la función salto de Heaviside, una *neurona sigmoidea* realiza una aproximación suave del perceptrón, [4, 5, 11]. Las funciones de activación clásicas incluyen funciones con una forma que se aproxima a la función de Heaviside, como las funciones de tangente hiperbólica y sigmoidea, o funciones con derivadas interesantes, como la función ReLU definida por $f(x) = \text{máx}(0, x)$ cuyo gradiente es 1 para todas las entradas positivas, [11]. Las funciones de activación y de pérdida se detallarán más en siguientes secciones.

Los perceptrones multicapa tiene tres tipos de capas: la capa de entrada, las capas ocultas y la capa de salida.

- **Capa de entrada:** recibe los datos de entrada (vectores) de la red. El número de neuronas en una capa de entrada suele ser el mismo número que la entidad de entrada a la red. Estas capas van seguidas de una o más capas ocultas. En las redes neuronales dirigidas hacia adelante clásicas están completamente conectadas a la siguiente capa oculta, sin embargo, en otras arquitecturas de red, la capa de entrada podría no estar completamente conectada.

- **Capa oculta:** los valores de peso en las conexiones entre las capas son cómo las redes neuronales, codifican la información aprendida extraída de los datos

de entrenamiento sin procesar. Son la clave para permitir que las redes neuronales modelen funciones no lineales, como en las limitaciones de las redes de perceptrones de una sola capa.

- **Capa de salida:** da la respuesta o predicción del modelo de la capa de salida. Dado que se asigna un espacio de entrada a un espacio de salida con el modelo de red neuronal, la capa de salida brinda una salida basada en la entrada de la capa de entrada, [5]. Dependiendo de la configuración de la red neuronal, la salida final puede ser una salida de valor real (regresión) o un conjunto de probabilidades (clasificación), esto está controlado por el tipo de función de activación usada en las neuronas en la capa de salida, [5]. La capa de salida suele utilizar una función de activación **softmax** o sigmoidea para la clasificación, [5].

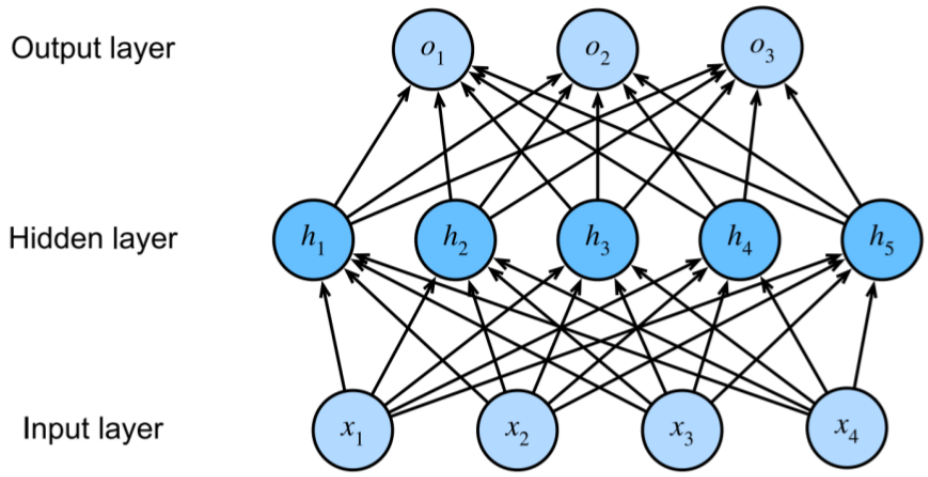


Figura 2.3. Perceptrón multicapa con cinco neuronas en su capa oculta. Tomado de [5].

2.3.3. Redes neuronales profundas

En esta sección se presenta la breve formulación matemática para redes neuronales profundas, tomada de [11]. Una arquitectura de red neuronal es un conjunto (V, E, f), donde V denota el conjunto de neuronas, E el conjunto de aristas dirigidas, y f la función de activación. Se representan los pesos y sesgos de la red con una función de peso $\omega\colon E \to \mathbb{R}$, los sesgos están incluidos con la suposición que cada capa incluye una neurona de sesgo, vinculado a todas las neuronas solo de la

siguiente capa. Siguiendo con esto, es posible clasificar redes neuronales acorde a su arquitectura (V, E, f), definiendo la clase $\mathcal{C}_{(V,E,f)}$ como el conjunto de todas las redes neuronales cuya arquitectura es (V, E, f).

Las neuronas están organizadas en capas disjuntas $L_0, ..., L_N$, tal que $V = \cup_{i=0}^{N} L_i$. El número de neuronas en una capa dada, $|L_i|$, se llama el **tamaño de la capa** L_i. Usando la terminología de grafos, se habla de capas totalmente conectadas cuando, para cada $i \in \{0, ..., N-1\}$, todas las neuronas en la capa L_i están conectados a todas las neuronas en la capa L_{i+1}. Una red neuronal dirigida hacia adelante es una red neuronal cuyo grafo no contiene ciclos dirigidos, [11]. Usando estas notaciones, se puede establecer que la clase de perceptrones multicapa es el conjunto de todas las clases $\mathcal{C}_{(V,E,f)}$ tal que la red resultante sea dirigida hacia adelante, con capas totalmente conectadas.

Se llama a una red neuronal *profunda* cuando tiene más de una capa oculta, es decir, cuando $N > 2$. De otra forma, se denomina poco profunda. Uno de los principales problemas del aprendizaje profundo es el diseño de tales redes, es decir, elegir la clase $\mathcal{C}_{(V,E,f)}$ adecuada de la red neuronal para resolver un problema dado. Los tamaños de las capas de entrada y salida dependen completamente del problema en cuestión y, en consecuencia, se encuentran fácilmente.

No hay una teoría matemática que pueda guiar hacia lo que se podría considerar un diseño bueno o eficiente, a menudo solo son útiles para determinados conjuntos de problemas. La cantidad de capas y neuronas ocultas en ellas son parte del hiperparámetro de la red, parámetros que se ajustan y diseñan cuidadosamente para que la red resuelva el problema de interés.

2.4. Algoritmo de propagación hacia atrás

El DGE provee una forma de actualizar los pesos y sesgos de una red neuronal usando gradientes, pero asume que los gradientes se pueden calcular efectivamente. Como se mencionó, los gradientes se calculan mediante el algoritmo de propagación hacia atrás. Primero se definen las notaciones a usar para referirse a una red neuronal. Se toma una red neuronal de una clase dada $\mathcal{C}_{(V,E,f)}$ con L capas, para $l = 1, ..., L$, se denota como $\omega^l_{j,k}$ los pesos para la conexión de la neurona k en la capa $(l-1)$ a la neurona j en la capa l, y b^l_j el sesgo para la neurona j en la capa l. Para cada capa, se puede por tanto definir una matriz de pesos W^l y un vector de sesgo b^l. También se introduce a^l, el vector de activación de la capa l, cuyas componentes

están definidas tal que

$$a_j^l = f\left(\sum_k \omega_{jk} a_k^{l-1} + b_j^l\right).$$

Usando estas notaciones, los cálculos realizados por una red neuronal se pueden resumir en la siguiente forma vectorizada:

$$a^l = f\left(W^l a^{l-1} + b^l\right).$$

Finalmente, se introduce la entrada ponderada z^l

$$z^l := W^l a^{l-1} + b^l,$$

por tanto verificando $a^l = f(z^l)$.

La propagación hacia atrás prevé una manera de calcular las derivadas parciales de la función de pérdida con respecto a cualquier peso $\omega_{j,k}^l$ y cualquier sesgo b_j^l, es decir,

$$\frac{\partial \mathcal{L}_p}{\partial \omega_{j,k}^l} \quad y \quad \frac{\partial \mathcal{L}_p}{\partial b_j^l},$$

de la red. La propagación hacia atrás se basa en dos suposiciones en la función de pérdida $\mathcal{L}_p$. La primera es la hipótesis de separabilidad que es requerida por el DGE, que la función de pérdida se puede separar en dos piezas que solo dependen en una muestra del conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1,\dots,n}$, la ecuación (1.5) es una consecuencia de esto. La segunda es que las pérdidas parciales obtenidas a través de la separabilidad pueden escribirse como funciones de las salidas de la red neuronal únicamente, es decir, tales que $\mathcal{L}_{p_i} = \mathcal{L}_{p_i}(a^L)$.

Nótese que

$$\delta_j^i := \frac{\partial \mathcal{L}_p}{\partial z_j^l},$$

el error de la neurona j en la capa l, y δ^l el vector asociado. El algoritmo permite calcular estos errores, y relacionarlos a las derivadas parciales de interés

$$\frac{\partial \mathcal{L}_p}{\partial \omega_{j,k}^l} \quad y \quad \frac{\partial \mathcal{L}_p}{\partial b_j^l},$$

necesarias para el DGE. Este se basa en las cuatro ecuaciones de la proposición 2.1.

Proposición 2.1. *Se tienen las siguientes ecuaciones donde $\odot$ designa el producto de Hadamard [4, 5, 11].*

$$\delta^L = \nabla_{a^L} \mathcal{L}_p \odot f'(z^L),$$

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot f'(z^l),$$
$$\frac{\partial \mathcal{L}_p}{\partial b_j^l} = \delta_j^l \quad y \quad \frac{\partial \mathcal{L}_p}{\partial \omega_{j,k}^l} = a_k^{l-1} \delta_j^l,$$

Demostración.

Usando regla de la cadena

$$\delta_j^L = \sum_k \frac{\partial \mathcal{L}_p}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L},$$

donde $\delta_j^L = \frac{\partial \mathcal{L}_p}{\partial z_j^L}$. Pero

$$\frac{\partial a_k^L}{\partial z_j^L} = 0$$

para todo $k \neq j$, y así

$$\delta_j^L = \frac{\partial \mathcal{L}_p}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}.$$

Ahora, usando el hecho que $a_j^L = f(z_j^L)$, se obtiene que

$$\delta_j^L = \frac{\partial \mathcal{L}_p}{\partial a_j^L} f'(z_j^L).$$

b) Para la segunda ecuación se tiene

$$\delta_j^L = \sum_k \frac{\partial \mathcal{L}_p}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1},$$

insertando la definición de δ. De igual forma $z_k^{l+1} = \sum_j \omega_{k,l}^{l+1} a_j^l + b_k^{l+1} = \sum_j \omega_{k,l}^{l+1} f(z_j^l) + b_k^{l+1}$. Así se obtiene, diferenciando esta expresión respecto a z_j^l,

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \omega_{k,j} f'(z_j^l).$$

Reemplazando esta expresión en la suma anterior

$$\delta_j^l = \sum_k \omega_{k,j}^{l+1} \delta_k^{l+1} f'(z_j^l) = ((W^{l+1})^T \delta^{l+1})_j f'(z_j^l).$$

c) Nuevamente usando regla de la cadena, para la tercera ecuación de la proposición

$$\delta_j^l = \sum_k \frac{\partial \mathcal{L}_p}{\partial b_k^l} \frac{\partial b_k^l}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{L}_p}{\partial b_k^l} 1_{\{j=k\}}$$

ya que $b_j^l = z_j^l - \sum_k \omega_{j,k}^l a_k^{l-1}$, esto prueba el resultado.

d) Finalmente si se toma $\frac{\partial \mathcal{L}_p}{\partial \omega_{j,k}^l}$ y usando la regla de la cadena

$$\frac{\partial \mathcal{L}_p}{\partial \omega_{j,k}^l} = \frac{\partial \mathcal{L}_p}{\partial z_j^l} \frac{\partial z_j^l}{\partial \omega_{j,k}^l} = \delta_j^l a_k^{l-1},$$

ya que $z_j^l = \sum_k \omega_{j,k}^l a_k^{l-1} + b_j^l$. Con esto se prueban todas las ecuaciones.

□

Las cuatro ecuaciones dan una forma para calcular el error de salida δ^L, y entonces propagarlo en la red hacia atrás para encontrar los gradientes necesarios para actualizar los pesos y los sesgos. Ahora ya se tiene lo necesario para construir un proceso de aprendizaje, u optimizador, como se detalla en [11].

Si se toma $\{(x_i, y_i)\}_{i=1,...,n}$ como el conjunto de entrenamiento. Para cada i, el conjunto de entradas de activación $a^{i,1}$, y se usa el DGE

$$W^l \leftarrow W^l - \frac{\eta}{m} \sum_{i \in B} \delta^{i,l} (a^{i,l-1})^T$$

$$b^l \leftarrow b^l - \frac{\eta}{m} \sum_{i \in B} \delta^{i,l}.$$

donde

- B es un subconjunto escogido aleatoriamente del conjunto de entrenamiento.
- $\delta^{i,l}$ con $l = L-1, ..., 1$ son los errores de propagación hacia atrás y $\delta^{i,L}$ el error de salida.
- $W^l a^{i,l-1} + b^l$ el peso de entrada y la correspondiente función de activación $a^{i,l} = f(W^l a^{i,l-1} + b^l)$ para cada capa $l = 2, ..., L$.

Con este optimizador, en la clase de red $\mathcal{C}_{V,E,f}$, se encuentra el conjunto de pesos y sesgos minimizando la función de pérdida $\mathcal{L}_p$ en el conjunto de entrenamiento [11].

2.5. Funciones de activación

Las funciones de activación toman escalares y dan escalares, y producen la activación de la neurona. Muchas funciones de activación pertenecen a la clase logística de transformadas que se asemejan a una S, ésta clase de función se llama sigmoidal [5]. La familia de funciones sigmoideas contiene variaciones, una de las cuales se conoce como la función sigmoidea, [5]. Algunas funciones de activación útiles en las redes neuronales son:

2.5.1. Lineal

Una transformación lineal, en la práctica, es parecida a una función identidad para redes neuronales, y $f(x) = Wx$ es un caso particular, donde la variable dependiente tiene una relación directamente proporcional con la variable independiente.

2.5.2. Sigmoide

Como todas las transformaciones logísticas, la función sigmoide puede reducir valores extremos o valores atípicos en los datos sin eliminarlos [5]. Una función sigmoidea convierte variables independientes de rango casi infinito en probabilidades simples entre 0 y 1, y la mayor parte de su salida estará muy cerca de 0 o 1.

2.5.3. Tangente hiperbólica

Así como la tangente representa una relación entre los lados opuestos y adyacentes de un triángulo rectángulo, tanh representa la relación entre el seno hiperbólico y el coseno hiperbólico: $\tanh(x) = \sinh(x)/\cosh(x)$. El rango normalizado es -1 a 1. La ventaja de tanh es que tiene la facilidad con números negativos.

2.5.4. Softmax

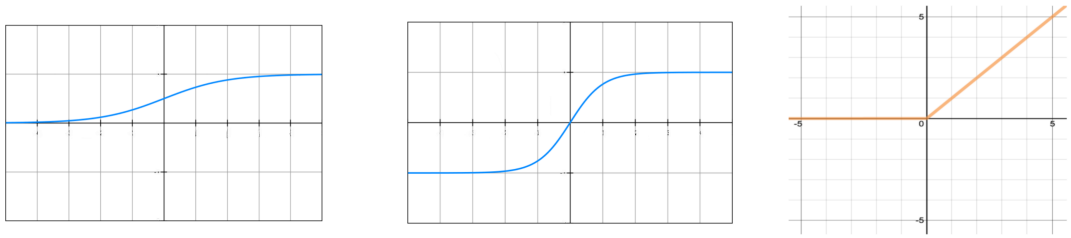
La softmax es una generalización de la regresión logística en la medida en que se puede aplicar a datos continuos (en lugar de clasificar binarios) y puede contener múltiples límites de decisión. Maneja sistemas de etiquetado multinomial. Se encuentra a menudo en la capa de salida de un clasificador.

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^j}.$$

Cuando se trabajan miles de etiquetas, se utiliza la variante **llamada función de activación jerárquica de softmax**, ésta variante descompone las etiquetas en una estructura de árbol y el clasificador softmax se entrena en cada nodo del árbol para dirigir la ramificación para la clasificación [5].

2.5.5. ReLu

La **función lineal rectificada** es una transformación más interesante que activa un nodo solo si la entrada está por encima de cero. Mientras, si la entrada está por debajo de cero, la salida es cero, pero cuando la entrada sube por encima de cierto umbral, tiene una relación lineal con la variable dependiente $f(x) = \max(0, x)$. ReLu ha demostrado entrenar mejor en la práctica que las funciones de activación sigmoidea, [5].



(a) **(b)** **(c)**

Figura 2.4. Funciones de pérdida más utilizadas en aprendizaje profundo. En 2.4a se muestra la gráfica de la función sigmoide, en 2.4b la gráfica de la tangente hiperbólica y en 2.4c la función ReLu. Tomado de [4].

2.6. Funciones de pérdida

Las **funciones de pérdida** cuantifican qué tan cerca está una red neuronal dada del ideal hacia el cual se está entrenando. La idea es sencilla, se calcula una métrica en función del error observado en las predicciones de la red. Luego los errores se agregan en todo el conjunto de datos y se promedian para tener un único número representativo de qué tan cerca está la red neuronal de su ideal.

Buscar este estado ideal es equivalente a encontrar los parámetros (pesos y sesgos) que minimizarán la «pérdida» incurrida por los errores. De esta forma, las funciones de pérdida ayudan a replantear el entrenamiento de redes neuronales como un problema de optimización. En la mayoría de los casos, estos parámetros no

se pueden resolver analíticamente, pero, en la mayoría de los casos, se pueden aproximar bien con algoritmos de optimización iterativos como el descenso de gradiente. La siguiente sección proporciona una descripción general de las funciones de pérdida comúnmente vistas.

- Sea N el número de muestras que se han recopilado.
- Cada punto de datos registra un conjunto de características de entrada y características de salida únicas. Sea P denote la cantidad de características de entrada recopiladas y M denote la cantidad de características de salida que se han observado.
- (X, Y) denota los datos de entrada y salida recopilados. El i-ésimo par en el conjunto de datos será X_i y Y_i.
- Se usará $\hat{Y}$ para denotar la salida de la red neuronal. $\hat{Y}$ es la conjetura de la red en Y y, por lo tanto, también tendrá M características.
- Se usará la notación $h(X_i) = \hat{Y}_i$ para denotar la red neuronal transformando la entrada X_i para dar la salida $\hat{Y}$.
- $y_{i,j}$ se refiere a la j-ésima característica observada en la i-ésima muestra recolectada.
- La función de pérdida se representará igual que en la sección 2.4, $\mathcal{L}_p(W, b)$.

La notación $h(X) = \hat{Y}$ debe ser condicionada en un conjunto de pesos y sesgos, así, se modifica la notación para decir $h_{w,h}(X) = \hat{Y}$.

2.6.1. Funciones de pérdida para regresión

2.6.1.1. Error cuadrático medio

Un modelo de regresión que requiere una salida de valor real, se utiliza la función de pérdida al cuadrado, como en el caso de los mínimos cuadrados ordinarios en la regresión lineal. Considere el caso donde se desea predecir solo una característica de salida ($M = 1$). El **error cuadrático medio (ECM)** en una predicción se eleva al cuadrado y se promedia sobre el número de datos en la siguiente ecuación para la pérdida:

$$\mathcal{L}_p(W, b) = \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2.$$

Si M es mayor que uno y se busca predecir múltiples características de salida para un conjunto dado de características de entrada en este caso, las entradas deseadas y predichas, Y y $\hat{Y}$, respectivamente, son una lista ordenada de números, es decir, vectores. Una variación de la función de pérdida ECM es

$$\mathcal{L}_p(W, b) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{M} \sum_{j=1}^{M} (\hat{y}_{ij} - y_{ij})^2.$$

Téngase en cuenta que N, el tamaño del conjunto de datos, y M, la cantidad de características que la red tiene que predecir, son constantes.

2.6.1.2. Error medio absoluto

De manera similar, una alternativa a la pérdida ECM es la pérdida del **error absoluto medio (EAM)**, como se muestra en la siguiente ecuación:

$$\mathcal{L}_p(W, b) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{M} |\hat{y}_{ij} - y_{ij}|$$

Esto simplemente promedia el error absoluto sobre todo el conjunto de datos.

2.6.1.3. Error logarítmico cuadrático medio

Otra función de pérdida utilizada para la regresión es el **error logarítmico cuadrático medio (ELCM)**:

$$\mathcal{L}_p(W, b) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\log \hat{y}_{ij} - \log y_{ij}\right)^2.$$

2.6.2. Funciones de pérdida para clasificación

También es posible construir redes neuronales para agrupar puntos de datos en diferentes categorías.

2.6.2.1. Pérdida de bisagra

La pérdida de bisagra es la función de pérdida más utilizada cuando la red debe optimizarse para una clasificación estricta. La siguiente es la ecuación para la

pérdida de bisagra cuando los puntos de datos deben clasificarse como -1 o 1:

$$\mathcal{L}_p(W, b) = \frac{1}{N} \sum_{i=1}^{N} \max\left(0, 1 - y_{ij}\hat{y}_{ij}\right).$$

La pérdida de bisagra se usa principalmente para clasificaciones binarias.

2.6.2.2. Pérdida logística

Las funciones de pérdida logística se utilizan cuando las probabilidades son de mayor interés que las clasificaciones estrictas. Predecir probabilidades válidas significa generar números entre 0 y 1, [5]. Predecir probabilidades válidas también significa asegurarse de que la probabilidad de resultados mutuamente excluyentes sume uno. Por esta razón, es esencial que la última capa de una red neuronal utilizada en la clasificación sea un softmax. La pérdida logística se expresa como sigue

$$\mathcal{L}_p(W, b) = \prod_{i=1}^{N} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}.$$

2.6.2.3. Logaritmo de probabilidad negativo.

Cuando se trata del producto de probabilidades, se acostumbra convertirlas al logaritmo de las probabilidades; por lo tanto, el producto de las probabilidades se transforma en la suma del logaritmo de las probabilidades. La función de pérdida de M clases da la siguiente ecuación para la pérdida logística

$$\mathcal{L}_p(W, b) = -\sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} \log \hat{y}_{i,j}.$$

Esto es matemáticamente equivalente a lo que se llama la **entropía cruzada** entre dos distribuciones de probabilidad, [5].

2.7. Hiperparámetros

En el aprendizaje de máquina, hay parámetros de modelo y parámetros que se ajustan para que las redes se entrenen mejor. Estos parámetros de ajuste se denominan **hiperparámetros** y se ocupan del control de las funciones de optimización y la selección de modelos durante el entrenamiento con el algoritmo de aprendizaje, [5]. La selección de hiperparámetros se enfoca en garantizar que el modelo se ajus-

te de forma adecuada al conjunto de datos de entrenamiento, mientras aprende la estructura de los datos lo más rápido posible.

Taza de aprendizaje: la tasa de aprendizaje afecta la cantidad en la que ajusta los parámetros durante la optimización para minimizar el error de las conjeturas de la red neuronal. Es un coeficiente que escala el tamaño de los pasos (actualizaciones) que toma una red neuronal a su vector de parámetros x cuando cruza el espacio de la función de pérdida.

En la propagación hacia atrás, se multiplica el gradiente de error por la tasa de aprendizaje y luego actualiza la última iteración de un peso de conexión con el producto para alcanzar un nuevo peso. La tasa de aprendizaje determina cuánto del gradiente se usará para el siguiente paso del algoritmo, [5]. Un gran error y una pendiente pronunciada se combinan con la tasa de aprendizaje para producir un gran paso. En la práctica a medida el error disminuye y el gradiente se aplana, el tamaño del paso tiende a acortarse, [4].

Un coeficiente de tasa de aprendizaje grande (p. ej., 1) hará que sus parámetros den saltos, y los pequeños (p. ej., 0.00001) harán que avance lentamente. Los grandes saltos ahorrarán tiempo inicialmente, pero pueden ser desastrosos si se sobrepasa el mínimo. Una tasa de aprendizaje demasiado grande hace que el algoritmo rebote hacia adelante y hacia atrás a ambos lados del mínimo sin detenerse nunca. Por el contrario, las tasas de aprendizaje pequeñas deberían llevarlo eventualmente a un mínimo de error (podría ser un mínimo local en lugar de uno global), pero pueden llevar mucho tiempo y aumentar la carga de un proceso que ya es computacionalmente intensivo.

Regularización: la regularización ayuda con los efectos de los parámetros fuera de control mediante el uso de diferentes métodos para minimizar el tamaño del parámetro con el tiempo.

En notación matemática, la regularización se representa por el coeficiente λ, que controla el equilibrio entre encontrar un buen ajuste y mantener bajo el valor de ciertas ponderaciones de características a medida que aumentan los exponentes de las características, [5]. Los coeficientes de regularización L_1 y L_2 ayudan a combatir el sobreajuste, resultado de un entrenamiento que terminó memorizando los datos y no predice nada fuera de éste, al reducir ciertos pesos, [5]. Los pesos de menor valor conducen a hipótesis más simples, y las hipótesis más simples son las más generalizables. Los pesos no regularizados con varios

polinomios de orden superior en el conjunto de características tienden a sobreajustarse al conjunto de entrenamiento. A medida que crece el tamaño del conjunto de entrenamiento de entrada, el efecto de la regularización disminuye y los parámetros tienden a aumentar en magnitud, [5]. Esto es apropiado, porque un exceso de características en relación con los ejemplos de conjuntos de entrenamiento conduce a un sobreajuste en primer lugar.

2.8. Otros métodos de optimización

Existen más métodos de optimización basados en el DGE que consisten en mejoras cuando se trabaja sobre conjuntos no convexos. A continuación se mencionan brevemente.

2.8.1. DGE con momento

La mejora que introduce es reemplazar los gradientes con un promedio sobre gradientes anteriores acelerando la convergencia y evitando el estancamiento en el proceso de optimización, que es mucho más probable de ocurrir para el DGE, [4].

2.8.2. Gradiente adaptivo

Reduce la tasa de aprendizaje dinámicamente por coordenadas y utiliza la magnitud del gradiente como un medio para ajustar la rapidez del progreso. A diferencia del DGE, éste utiliza utiliza la expansión de Taylor hasta segundo orden, [4].

2.8.3. Propagación de la raíz cuadrada media

Una limitación del gradiente adaptivo es que puede dar como resultado un tamaño de paso muy pequeño para cada parámetro al final de la búsqueda, lo que puede ralentizar demasiado el progreso de la búsqueda y puede significar no ubicar los valores óptimos, [4].

Es una extensión del descenso de gradiente y la versión **AdaGrad** del descenso de gradiente que utiliza un promedio decreciente de gradientes parciales en la adaptación del tamaño de paso para cada parámetro, [4]. El uso de un promedio móvil decreciente permite que el algoritmo olvide los gradientes iniciales y se concentre en los gradientes parciales observados más recientemente durante el progreso de la búsqueda, superando la limitación del gradiente adaptivo.

2.8.4. Adam

Combina todas estas técnicas en un algoritmo de aprendizaje eficiente. Este es un algoritmo que se ha vuelto bastante popular como uno de los algoritmos de optimización más robustos y efectivos para usar en el aprendizaje profundo. Sin embargo, no está exento de problemas. Hay situaciones en las que Adam puede divergir debido a un control de varianza deficiente, como se muestra en [12]. Aunque en el trabajo [14], Zaheer propuso una revisión para Adam, llamada Yogi, que aborda estos problemas. Además utiliza una corrección de sesgo para ajustarse a un inicio lento al estimar el impulso y un segundo momento.

3. Factorización de matrices para filtrado colaborativo

Primero se verán los aspectos generales sobre **sistemas de recomendación (SR)** y **filtrado colaborativo (FC)**. El modelo central a tratar es un modelo de factorización de matrices sin restricciones al cual se introduce un factor de regularización y dos términos de sesgos para usuario y elemento, aunque el punto de partida son los modelos de reducción de la dimensionalidad para comparar sus ventajas respecto a un método de reducción de dimensionalidad estándar. La factorización de matrices es un método basado en el modelo usado en filtrado colaborativo.

3.1. Sistemas de recomendación

Los modelos básicos para los SR funcionan con dos tipos de datos, que son (i) las interacciones usuario-elemento, como calificaciones o comportamiento de compra, y (ii) la información de atributos sobre los usuarios y elementos, como perfiles textuales o palabras clave relevantes [1]. Los métodos que usan el primero se conocen como métodos de filtrado colaborativo, mientras que los métodos que usan el último se denominan métodos de recomendación basados en el contenido [1]. Los sistemas basados en contenido también usan las matrices de calificación, matrices cuya entrada (i, j) es la calificación asignada por el usuario i al producto (elemento) j, en la mayoría de los casos, aunque el modelo generalmente se enfoca en las calificaciones de un solo usuario en lugar de las de todos los usuarios, [1].

Los SR basados en contenido y contexto también son útiles para incorporar descripciones de contenido de elementos/usuarios y señales contextuales como marcas de tiempo y ubicaciones [2]. En los SR basados en el conocimiento, las recomendaciones se basan en requisitos de usuario especificados explícitamente en lugar de utilizar calificaciones históricas o datos de compra y se utilizan restricciones y bases de conocimiento externas para crear la recomendación, [1]. Algunos SR combinan estos diferentes aspectos para crear sistemas híbridos. Los sistemas híbridos pue-

den combinar las fortalezas de varios tipos de SR para crear técnicas que pueden funcionar de manera más sólida en una amplia variedad de entornos, [1].

3.2. Filtrado colaborativo

En general, las técnicas de FC se pueden clasificar en: FC basada en memoria, FC basada en modelo y su híbrido. Las técnicas representativas de FC basadas en memoria son FC basadas en vecinos más cercanos, como FC basadas en usuarios y FC basadas en elementos. Los modelos de factores latentes, como la factorización de matrices, son ejemplos de FC basados en modelos. Muchos enfoques de FC basados en modelos se pueden ampliar con redes neuronales, lo que da lugar a modelos más flexibles y escalables con la aceleración de la computación en el aprendizaje profundo [4]. En general, FC solo usa los datos de interacción usuario-elemento para hacer predicciones y recomendaciones.

La idea principal de los enfoques de recomendación colaborativa es explotar la información sobre el comportamiento pasado o las opiniones de una comunidad de usuarios existente para predecir qué elementos probablemente le gustarán o le interesarán al usuario actual del sistema, [1].

Los enfoques colaborativos puros toman una matriz de calificaciones dadas de usuario-elemento como la única entrada y normalmente producen los siguientes tipos de salida: (a) una predicción (numérica) que indica en qué medida le gustará o disgustará al usuario actual un determinado elemento y (b) una lista de n artículos recomendados, [1].

3.2.1. Métodos basados en memoria

Los métodos basados en la memoria también se denominan algoritmos de filtrado colaborativo basados en la vecindad, estos se encontraban entre los primeros algoritmos de filtrado colaborativo, en los que las calificaciones de las combinaciones de elementos de usuario se predicen en función de sus vecindarios. Las ventajas de las técnicas basadas en la memoria yace en su facilidad de implementar y las recomendaciones resultantes suelen ser fáciles de explicar. Por otro lado, los algoritmos basados en memoria no funcionan muy bien con matrices de calificación dispersas. En otras palabras, dichos métodos pueden carecer de una cobertura completa de las predicciones de calificación, [1].

3.2.1.1. Filtrado colaborativo basado en usuario

Las calificaciones proporcionadas por usuarios afines a un usuario objetivo A se utilizan para hacer las recomendaciones para A. Por lo tanto, la idea básica es determinar los usuarios que son similares al usuario objetivo A y recomendar calificaciones. Para las calificaciones no observadas de A calculando promedios ponderados de las calificaciones de este grupo de pares. Las funciones de similitud se calculan entre las filas de la matriz de calificaciones para descubrir usuarios similares [1].

3.2.1.2. Filtrado colaborativo basado en elemento

Para hacer las predicciones de calificación para el elemento de destino B por parte del usuario A, el primer paso es determinar un conjunto S de elementos que sean más similares al elemento de destino B. Se utilizan las calificaciones en el conjunto de elementos S, que son especificados por A para predecir si al usuario A le gustará el artículo B. Las funciones de similitud se calculan entre las columnas de la matriz de calificaciones para descubrir elementos similares, [1].

3.2.2. Métodos basados en el modelo

En los métodos basados en modelos, los métodos de aprendizaje de máquina y minería de datos se utilizan en el contexto de modelos predictivos. En los casos en que el modelo está parametrizado, los parámetros de este modelo se aprenden dentro del contexto de un marco de optimización. Algunos ejemplos de tales métodos basados en modelos incluyen árboles de decisión, modelos basados en reglas, métodos bayesianos y modelos de factores latentes, [1]. Muchos de estos métodos, como los modelos de factores latentes, tienen un alto nivel de cobertura incluso para matrices de calificaciones escasas.

Los SR basados en modelos a menudo tienen una serie de ventajas sobre los métodos basados en vecinos (memoria):

1 Eficiencia espacial: por lo general, el tamaño del modelo aprendido es mucho más pequeño que la matriz de calificaciones original. Por lo tanto, los requisitos de espacio son a menudo bastante bajos. Por otro lado, un método de vecindad basado en el usuario podría tener una complejidad de espacio $\mathcal{O}(m^2)$, donde m es el número de usuarios. Un método basado en elementos tendrá una complejidad de espacio $\mathcal{O}(n^2)$, para más detalles sobre la notación puede referirse a [13].

2 Velocidad de entrenamiento y velocidad de predicción: un problema con los métodos basados en vecindarios es que la etapa de preprocesamiento es cuadrática en el número de usuarios o en el número de elementos. Los sistemas basados en modelos suelen ser mucho más rápidos en la fase de preprocesamiento de la construcción del modelo entrenado. En la mayoría de los casos, el modelo compacto y resumido se puede usar para hacer predicciones de manera eficiente, [1].

3 Evitar el sobreajuste: el sobreajuste es un problema grave en muchos algoritmos de aprendizaje de máquina, en los que la predicción está demasiado influenciada por artefactos aleatorios en los datos, este problema también se encuentra en los modelos de clasificación y regresión. El enfoque de resumen de los métodos basados en modelos a menudo puede ayudar a evitar el sobreajuste. Además, los métodos de regularización se pueden utilizar para hacer que estos modelos sean robustos, [1].

3.2.3. Reacciones explícitas e implícitas

Para conocer las preferencias de los usuarios, el sistema recopilará sus comentarios. La retroalimentación puede ser explícita o implícita. Por ejemplo, recopila clasificaciones de estrellas que van de una a diez estrellas para películas. Es evidente que recopilar comentarios explícitos requiere que los usuarios indiquen sus intereses de manera proactiva. No obstante, los comentarios explícitos no siempre están disponibles, ya que muchos usuarios pueden ser reacios a calificar los productos. En las aplicaciones del mundo real las matrices de calificación tienden a ser muy escasas, ya que los clientes generalmente brindan calificaciones para (o han comprado) solo una pequeña fracción de los elementos del catálogo. En general, el desafío en ese contexto es calcular buenas predicciones cuando hay relativamente pocas calificaciones disponibles. En términos relativos, la retroalimentación implícita a menudo está fácilmente disponible, ya que se ocupa principalmente de modelar el comportamiento implícito, como los clicks de los usuarios, etcétera. También es posible diferenciar las tareas según los tipos de comentarios y datos de entrada, [4].

3.3. Métodos de reducción de dimensionalidad

La escasez es un problema recurrente en los SR, incluso en la configuración más simple, es probable que una matriz dispersa con miles de filas y columnas (es decir,

usuarios y elementos), la mayoría de los cuales son ceros. Por lo tanto, la reducción de la dimensionalidad se produce de forma natural. Su aplicación marca una gran diferencia y sus resultados son directamente aplicables al cálculo del valor predicho, que ahora se considera un enfoque para el diseño de SR, en lugar de una técnica de preprocesamiento, [2].

Los métodos de reducción de dimensionalidad se usan comúnmente en otras áreas de análisis de datos para representar los datos subyacentes en una pequeña cantidad de dimensiones. La idea básica de los métodos de reducción de dimensionalidad es rotar el sistema de ejes, de modo que se eliminen las correlaciones por pares entre las dimensiones. Aprovechan las correlaciones de filas y columnas para crear una representación reducida y completamente especificada. Los métodos de factorización de matrices proporcionan una forma ordenada de aprovechar todas las correlaciones de filas y columnas de una sola vez para estimar la matriz de datos completa, [1]. Esta sofisticación del enfoque es una de las razones por las que los modelos de factores latentes se han convertido en lo último en filtrado colaborativo, [2].

A continuación se abordan los algoritmos de reducción de dimensionalidad más relevantes en el contexto de SR: análisis de componentes principales (ACP) y los modelos de factorización de matrices. Primero se desarrollará el método de ACP, luego los modelos de factores latentes y factorización de matrices y las ventajas que estos presentan sobre el método de ACP.

3.3.1. Análisis de componentes principales

Es un método estadístico clásico para encontrar patrones en conjuntos de datos de alta dimensionalidad. ACP permite obtener una lista ordenada de componentes que representan la mayor cantidad de varianza de los datos en términos de errores de mínimos cuadrados: la cantidad de varianza capturada por el primer componente es mayor que la cantidad de varianza en el segundo componente. Finalmente, es posible reducir la dimensionalidad de los datos ignorando aquellos componentes con una pequeña contribución a la varianza, [2].

Como ejemplo ilustrativo del ACP considere una nube de puntos bidimensional generada por una combinación de distribuciones. Después de centrar los datos, se obtienen los componentes principales y se denotan por u_1 y u_2. Tenga en cuenta que la longitud de las nuevas coordenadas es relativa a la energía contenida en sus vectores propios. Por lo tanto, para el ejemplo particular, si el primer componente u_1 representa el 83.5 % de la energía, lo que significa que eliminar el segundo compo-

nente u_2 implicaría perder solo el 16.5 % de la información. La regla general es elegir un m' de modo que la información acumulada esté por encima de cierto umbral, normalmente el 90 %. El ACP permite recuperar la matriz de datos original al proyectar los datos en el nuevo sistema de coordenadas $X'_{n\times m'} = X_{n\times m}W'_{m\times m'}$, donde $W'_{m\times m'}$ es la matriz que se encarga de proyectar la información contenida en $n \times m$ dimensiones a $n \times m'$ dimensiones. La nueva matriz de datos X' contiene la mayor parte de la información de la X original con una reducción de dimensionalidad de $m - m'$, [2].

ACP es una técnica poderosa, pero tiene limitaciones importantes. Este método se basa en que el conjunto de datos empíricos es una combinación lineal de una determinada base, aunque se han propuesto generalizaciones de ACP para datos no lineales [2]. Otra suposición importante de ACP es que el conjunto de datos original se extrajo de una distribución gaussiana. Cuando esta suposición no es cierta, no hay garantía de que los componentes principales sean significativos [2].

3.3.1.1. Desarrollo del método

Se tiene una serie de variables $(x_1, ..., x_m)$ y $(y_1, ..., y_m)$. Entonces se tiene

$$y_i = \sum_{j=1}^{m} a_{ij}x_j = a_i^t x = (a_{i1}, ..., a_{im}) \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix},$$

con a_{ij} la entrada en la fila i y columna j de la matriz de datos. Con el conjunto de restricciones de igualdad $\|a_i\| = 1$ para cada $i = 1, ..., m$, o expresado de otra forma $g(a_i) = a_i^t a_i - 1 = 0$. Se pretende maximizar la varianza de y_i, que es

$$\mathrm{Var}(y_i) = \mathrm{Var}(a_1^t x) = a_i^t \Sigma a_i$$

donde Σ es la matriz de covarianza. El método que se suele utilizar para funciones de varias variables es el método de los multiplicadores de Lagrange, para más detalles sobre el teorema y su demostración puede referirse a [8]. La función objetivo o lagrangiana es

$$L_g(a_i) = a_i^t \Sigma a_i - \lambda(a_i^t a_i - 1).$$

Se encuentran los puntos críticos calculando la derivada respecto a a_i e igualando a cero

$$\frac{\partial L_g}{\partial a_i} = \Sigma a_i + a_i^t \Sigma - \lambda a_i - \lambda a_i^t = 0,$$

Ahora usando la propiedad $a_i^t\Sigma = \Sigma a_i$, reescribiendo la ecuación se obtiene

$$\frac{\partial L_g}{\partial a_i} = 2\Sigma a_i - \lambda I a_i - \lambda I a_i = 0$$

o

$$\frac{\partial L_g}{\partial a_i} = (\Sigma - \lambda I)a_1 = 0.$$

Para que el sistema tenga solución no trivial, por el teorema Rouché-Fröbenius la matriz debe tener determinante cero, [15]. Así, una condición necesaria y suficiente es

$$|\Sigma - \lambda I| = 0. \tag{3.1}$$

De la ecuación ((3.1)), λ debe ser un autovalor de Σ. La matriz de covarianza, Σ es de orden m y definida positiva, por esto sabemos que sus valores propios son reales positivos. Con lo cual podemos ordenarlos de tal forma que $\lambda_1 > \lambda_2 > \cdots > \lambda_m$. Del resultado $\Sigma a_i = \lambda I a_i$ y la condición de restricción obtenemos

$$\mathrm{Var}(y_i) = \lambda a_i^t a_i = \lambda.$$

Se puede notar que para maximizar la varianza son necesarios los valores propios. Un segundo conjunto de restricciones se obtiene de exigir $\mathrm{Cov}(y_j, y_i) = 0$, para $j \neq i$, que es

$$\mathrm{Cov}(y_j, y_i) = \mathrm{Cov}(a_j x, a_i x) = a_j^t E((x-\mu)(x-\mu))a_i = a_j^t \Sigma a_i$$

donde μ es el valor esperado. De esto, $a_j^t \Sigma a_i = a_j^t(\lambda a_i) = \lambda a_j^t a_i = 0$. Quiere decir que a_j debe ser ortogonal a a_i. Por tanto resulta necesario volver a implementar el método de los multiplicadores de Lagrange, en este caso usando dos condiciones. La función objetivo es

$$L_g(a_i) = a_i^t \Sigma a_i - \lambda(a_j^t a_j - 1) - \delta(a_j^t a_i).$$

Repitiendo el procedimiento anterior se llega a las siguientes expresiones

$$\frac{\partial L_g}{\partial a_j} = 2\Sigma a_j - 2\lambda a_j - \delta a_i = 0,$$

multiplicando por a_i^t por la izquierda

$$\delta = 2a_i^t \Sigma a_j.$$

Y la derivada parcial respecto a a_i es

$$\frac{\partial L_g}{\partial a_i} = 2\Sigma a_2 - 2\lambda a_2 - 2a_1^t \Sigma a_2 a_1 = 0,$$

y el último término de esta ecuación es cero, así

$$2(\Sigma - \lambda I)a_j = 0.$$

Si

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad A = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \quad y \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix},$$

del sistema $y = Ax$ se tiene

$$\Delta = \mathrm{Var}(y) = A^t \mathrm{Var}(x) A = A^t \Sigma A$$

o

$$\Sigma = A\Delta A^t = A\Delta A^{-1}$$

con $A^t = A^{-1}$ por que A es ortogonal y $\Delta = \mathrm{diag}(\lambda_1, ..., \lambda_m)$. En resumen, para obtener las componentes principales es suficiente calcular los valores y vectores propios de la matriz de covarianzas y a partir de esto hacer las transformaciones de multiplicar por filas los resultados en forma de producto escalar para obtener las nuevas variables que no estarán correlacionadas. En base a eso se juzgará si será necesario eliminar alguna para reducir la dimensión del problema.

3.3.2. Modelos de factores latentes

Es una técnica de reducción de datos usada para métodos basados en el modelo, que utiliza algoritmos de clasificación como una subrutina. Por tanto, los métodos de reducción de la dimensionalidad únicamente permiten crear una representación más conveniente de los datos para métodos basados en el modelo, [1]. En el resto del capítulo se presentarán métodos más sofisticados, por que el objetivo es usar métodos de reducción de la dimensionalidad para estimar directamente la matriz de datos de una sola vez, a diferencia del ACP que no predice.

La idea básica usar el hecho que porciones significativas de las filas y columnas de las matrices de datos están altamente correlacionadas. Como resultado, los datos tienen redundancias incorporadas y la matriz de datos resultante a menudo

se aproxima bastante bien a una matriz de rango bajo. Debido a las redundancias inherentes a los datos, la aproximación de rango bajo completamente especificada se puede determinar incluso con un pequeño subconjunto de las entradas en la matriz original, esta aproximación de rango bajo totalmente especificada a menudo proporciona una estimación sólida de las entradas que faltan, [1].

Una forma de comprender la eficacia de los modelos de factores latentes es examinar el papel que desempeña la factorización en tales matrices. La factorización es, de hecho, una forma más general de aproximar una matriz cuando es propensa a la reducción de dimensionalidad debido a las correlaciones entre columnas (o filas), [1].

3.3.3. Principios básicos de factorización de matrices

Se asume la matriz de calificaciones que se denota por R, y es una matriz de $m \times n$ donde m es el número de usuarios y n el número de elementos. Por lo tanto, la calificación del usuario i para el elemento j se indica mediante r_{ij}. En el modelo básico de factorización de matrices, la matriz de calificaciones aproximadamente se factorizada dentro de una matriz U de $m \times k$ y otra V de $n \times k$ como

$$R \approx UV^T. \tag{3.2}$$

Cada columna de U (o V) se denomina vector latente o componente, mientras cada fila de U (o V) se denomina como factor latente. La i-ésima fila $\bar{u_i}$ de U es llamada factor de usuario, y contiene k entradas correspondientes a la afinidad del usuario i hacia los k conceptos en la matriz de calificaciones. Similarmente, cada fila $\bar{v_i}$ de V se entiende como un factor de elemento, y representa la afinidad del i-ésimo elemento hacia los k conceptos.

De la ecuación (3.2), se sigue que cada calificación r_{ij} en R se puede expresar aproximadamente como un producto punto del i-ésimo factor de usuario y el j-ésimo factor de elemento:

$$r_{ij} \approx \bar{u_i}\bar{v_j}. \tag{3.3}$$

Ya que los factores latentes $\bar{u_i} = (u_{i1}, ..., u_{ik})$ y $\bar{v} = (v_{j1}, ..., v_{jk})$ se pueden entender como las afinidades de los usuarios para k conceptos diferentes, una manera intuitiva de expresar:

$$r_{ij} \approx \sum_{s=1}^{k} u_{is}v_{js}.$$

Las diferencias clave entre varios métodos de factorización de matrices surgen en términos de las restricciones impuestas a U y V (por ejemplo, ortogonalidad o no negatividad de los vectores latentes) y la naturaleza de la función objetivo.

3.3.4. Familia de factorización de matrices

Existen diversas formas de factorización de matrices y la mayoría de formulaciones de optimización minimizan la norma de Frobenius de la matriz residual $(R-UV^T)$ sujeta a varias restricciones en las matrices factoriales U y V, [1]. Nótese que el objetivo de la función es hacer que UV^T se aproxime a la matriz de calificaciones R tanto como sea posible. Las restricciones sobre las matrices factoriales logran diferentes propiedades de interpretabilidad. De hecho, la familia más amplia de modelos de factorización de matrices puede usar cualquier otra función objetivo o restricción para forzar una buena aproximación. Esta familia más amplia se puede escribir de la siguiente manera:

Optimizar $J =$ [Función objetivo que cuantifica el emparejamiento entre R y UV^T]

sujeto a:

Restricciones en U y V.

La función objetivo de un método de factorización de matrices a veces se denomina función de pérdida, cuando está en forma de minimización. Tenga en cuenta que la formulación de optimización puede ser un problema de minimización o maximización, pero el objetivo de la función objetivo siempre es forzar a R a que coincida con UV^T lo más cerca posible. La norma de Frobenius es un ejemplo de un objetivo de minimización, y algunos métodos de factorización de matrices probabilísticas utilizan una formulación de maximización como la función objetivo de máxima verosimilitud [1, 7, 9].

En la mayoría de los casos, se agregan regularizadores a la función objetivo para evitar el sobreajuste. Las diversas restricciones a menudo imponen diferentes tipos de interpretabilidad de los factores. Dos ejemplos de dicha interpretabilidad son la ortogonalidad (que proporciona la interpretabilidad geométrica) y la no negatividad (que proporciona la interpretabilidad de la suma de las partes). Además, aunque las restricciones aumentan el error en las entradas observadas, a veces pueden mejorar los errores en las entradas no observadas cuando tienen una interpretación semántica

significativa [1]. Esto se debe a que las restricciones reducen la varianza de las entradas no observadas al tiempo que aumentan el sesgo. Como resultado, el modelo tiene una mejor generalización. Por ejemplo, fijar las entradas en una columna en cada uno de U y V a unos casi siempre da como resultado un mejor rendimiento. La selección de las restricciones correctas para usar a menudo depende de los datos y requiere conocimientos sobre el dominio de la aplicación en cuestión.

Existen otras formas de factorización en las que se puede asignar interpretabilidad probabilística a los factores. El **análisis semántico probabilístico latente (ASPL)** es un buen ejemplo, y puede verse como una variante probabilística de factorización de matriz no negativa. En este método la función objetivo aprende los parámetros de este proceso generativo para que la probabilidad de que el proceso generativo cree la matriz de calificaciones sea la mayor posible. Por lo tanto, la función objetivo está en forma de maximización. En muchas de estas formulaciones, las técnicas de optimización como el descenso (o ascenso) de gradiente son útiles. Por lo tanto, la mayoría de estos métodos utilizan ideas muy similares en cuanto a la formulación del problema de optimización y la metodología de solución subyacente. Para más detalles de este método puede referirse a [1].

3.3.5. Factorización de matrices sin restricciones

La forma más fundamental de factorización de matrices es el caso sin restricciones, en el que no se imponen restricciones a las matrices factoriales U y V. Gran parte de la literatura de recomendaciones se refiere a la factorización de matrices sin restricciones como descomposición en valores singulares (DVS). Estrictamente hablando, esto es técnicamente incorrecto; en DVS, las columnas de U y V deben ser ortogonales [1].

Antes de discutir la factorización de matrices incompletas, primero se considera el problema de factorización de matrices completamente especificadas. Para determinar las matrices U y V se puede formular un problema de optimización con respecto a dichas matrices:

$$\text{Minimizar } J = \frac{1}{2} \left\| R - UV^T \right\|^2$$

$$\text{sujeto a ninguna restricción en } U \text{ y } V.$$

Aquí, $\|\cdot\|^2$ representa la norma de Frobenius al cuadrado de la matriz, que es igual a la suma de los cuadrados de las entradas de la matriz (ECM). Así, la función objetivo es igual a la suma de los cuadrados de las entradas en la matriz residual $(R - UV^T)$.

Sin embargo, en el contexto de una matriz con entradas faltantes, solo se conoce un subconjunto de las entradas de R. Por lo tanto, la función objetivo, como se escribió anteriormente, tampoco está definida. Después de todo, no se puede calcular la norma de Frobenius de una matriz en la que faltan algunas de las entradas. Por lo tanto, la función objetivo necesita reescribirse solo en términos de las entradas observadas para aprender U y V. Lo bueno de este proceso es que una vez que se aprenden los factores latentes U y V, la matriz de calificaciones completa se puede reconstruir como UV^T directamente.

Sea (i, j) el par ordenado usuario-elemento, los cuales son observados en R, denotados por S. Aquí, $i \in \{1, ..., m\}$ es el índice de un usuario, y $j \in \{1, ..., n\}$ es el índice de un elemento. Por tanto, el conjunto observado de pares usuario-elemento es definido como

$$S = \{(i, j) \mid r_{ij} \text{ es observado}\}$$

Si de alguna manera se puede factorizar la matriz incompleta R como el producto aproximado UV^T de matrices completamente especificadas $U = [u_{i,s}]_{m\times k}$ y $V = [v_{js}]_{n\times k}$, entonces todas las entradas en R pueden ser predichas. Específicamente, la entrada (i, j) de la matriz R se puede predecir como:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is}\ v_{js}.$$

El sombrero en el lado izquierdo de la ecuación de la calificación indica que es un valor predictivo y no uno observado. La diferencia entre el valor observado y predicho de una entrada especificada (i, j) está dada por

$$e_{ij} = \left(r_{ij} - \hat{r}_{ij}) = (r_{ij}) = (r_{ij} - \sum_{s=1}^{k} u_{is}\ v_{js}\right).$$

Entonces, la función objetivo modificada, la cual solo funciona para matrices incompletas, se calcula solo para las entradas en S

$$\text{Minimizar } J = \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is}\ v_{js}\right)^2$$

sujeto a ninguna restricción en U y V.

Nótese que la función objetivo mencionada suma los errores solo sobre las entradas

observadas en S. Además, cada uno de los términos

$$\left(r_{ij} - \sum_{s=1}^{k} u_{is}\ v_{js}\right)^2$$

es el error cuadrático e_{ij}^2 entre los valores observado y predicho de la entrada (i, j). Aquí, u_{is} y v_{js} son las variables desconocidas, que es necesario aprender para minimizar la función objetivo. Entonces únicamente queda aplicar algún método de descenso del gradiente para minimizar el error, pero primero se introduce un término de regularización y dos de sesgos al modelo.

3.3.5.1. Introduciendo el término de regularización

Uno de los principales problemas con este enfoque surge cuando la matriz de calificaciones R es escasa y se observan relativamente pocas entradas. Este es casi siempre el caso en escenarios reales, [1].

En tales casos, el conjunto S observado de clasificaciones es pequeño, lo que puede provocar un sobreajuste. Tenga en cuenta que el sobreajuste también es un problema común en la clasificación cuando los datos de entrenamiento son limitados, de los mejores enfoque para abordar este problema es utilizar la regularización, como se mencionó al final del Capítulo 2.

La idea es desalentar valores muy grandes de los coeficientes en U y V para fomentar la estabilidad. Por lo tanto, se agrega un término de regularización, $\chi(\|U\|^2 + \|V\|^2)/2$, a la función objetivo, con $\chi > 0$. Este es un enfoque estándar, que se utiliza en muchas formas de clasificación y regresión, y también se aprovecha mediante el filtrado colaborativo, [1].

Como en el caso anterior, suponga que $e_{ij} = (r_{ij} - \sum_{s=1}^{k} u_{is}\ v_{js})$ representa la diferencia entre el valor observado y el valor predicho de la entrada especificada $(i, j) \in S$. La función objetivo regularizada es la siguiente

$$\text{Minimizar } J = \frac{\chi}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is}\ v_{js}\right)^2 + \frac{\chi}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\chi}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2.$$

3.3.5.2. Introduciendo sesgos de usuario y elemento

Ahora se construye una variación del modelo sin restricciones para incorporar variables que pueden aprender los sesgos del usuario y de los elementos. A cada usuario i, se le asocia una variable o_i, que indica el sesgo general de los usuarios

para calificar los elementos. Por ejemplo, si el usuario i es una persona generosa, que tiende a calificar muy bien todos los elementos, entonces la variable o_i será una cantidad positiva. Por otro lado, el valor de o_i será negativo para un usuario que califica negativamente la mayoría de los elementos. De manera similar, la variable p_j denota el sesgo en las calificaciones del elementos j. Los elementos que gustan mucho (por ejemplo con las películas, un éxito de taquilla) tenderán a tener valores mayores (positivos) de p_j, mientras que los elementos que no gustan tendrán valores negativos de p_j. El trabajo del modelo factorial es aprender los valores de o_i y p_j de una manera basada en datos. El principal cambio al modelo de factor latente original es que una parte de la (i, j)-ésima calificación se explica por $o_i + p_j$ y el resto por la (i, j)-ésima entrada del producto UV^T de las matrices de factores latentes [1]. Por lo tanto, el valor predicho de la calificación de la entrada (i, j) viene dado por lo siguiente:

$$\hat{r}_{ij} = o_i + p_j + \sum_{s=1}^{k} u_{is}\ v_{js}$$

Así, el error e_{ij} de una entrada observada $(i, j) \in S$ está dado por

$$e_{ij} = r_{ij} - \hat{r}_{ij} = r_{ij} - o_i - p_j - \sum_{s=1}^{k} u_{is}\ v_{js}$$

Tenga en cuenta que los valores o_i y p_j también son variables que deben aprenderse de manera basada en datos junto con las matrices de factores latentes U y V. Entonces, la función objetivo de minimización J se puede formular agregando los errores cuadráticos sobre las entradas observadas de la matriz de calificaciones (es decir, establecer S) de la siguiente manera:

$$\sum \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - o_i - p_j - \sum_{s=1}^{k} u_{is}\ v_{js} \right)^2$$
$$+ \frac{\chi}{2} \left(\sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 + \sum_{i=1}^{m} o_i^2 + \sum_{j=1}^{n} p_j^2 \right)$$

Resulta que este problema es diferente de la factorización de matrices sin restricciones solo en un grado menor. En lugar de tener variables de sesgo separadas o_i y p_j para usuarios y elementos, se puede aumentar el tamaño de las matrices de factores para incorporar estas variables de sesgo, [1]. Es necesario agregar dos columnas adicionales a cada matriz factorial U y V, para crear matrices factoriales más grandes

de tamaño $m \times (k+2)$ y $n \times (k+2)$, respectivamente. Las últimas dos columnas de cada matriz factorial son especiales, porque corresponden a los componentes de sesgo. Específicamente, se tiene

$$u_{i,k+1} = o_i \text{ para cada } i \in \{1, ..., m\} \quad \text{y} \quad u_{i,k+2} = 1 \text{ para cada } i \in \{1, ..., m\},$$

$$v_{j,k+1} = 1 \text{ para cada } j \in \{1, ..., n\} \quad \text{y} \quad v_{j,k+2} = p_j \text{ para cada } j \in \{1, ..., n\}.$$

Tómese un momento para meditar la estructura de las matrices factoriales con las columnas introducidas, se dará cuenta que aparecen como una consecuencia natural. Entonces, el problema de optimización modificado con estas matrices factoriales ampliadas es el siguiente:

$$\begin{aligned} \text{Minimice } J = \frac{1}{2} \sum_{i,j \in S} \left(r_{ij} - \sum_{s=1}^{k+2} u_{is} \cdot v_{js} \right)^2 \\ + \frac{\chi}{2} \sum_{s=1}^{k+2} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 \right) \end{aligned} \tag{3.4}$$

sujeto a

$$\begin{aligned} &\text{la columna } (k+2) \text{ de } U \text{ contiene solo } 1s \\ &\text{la columna } (k+1) \text{ de } V \text{ contiene solo } 1s \end{aligned}$$

El otro cambio es el incremento en el tamaño de las matrices factoriales al incorporar las variables de usuario y elemento. Ahora utilizando el método del descenso del gradiente la actualización en las entradas pueden ser ejecutadas ciclando sobre cada una de las entradas especificadas $(i, j) \in S$:

$$\begin{aligned} u_{iq} &\leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \chi \cdot u_{iq}) \text{ para cada } q \in \{1, ..., k+2\} \\ v_{jq} &\leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \chi \cdot v_{jq}) \text{ para cada } q \in \{1, ..., k+2\}. \end{aligned}$$

Donde u_{iq} y v_{jq} se obtienen calculando las derivadas parciales $\partial J / \partial u_{iq}$ y $\partial J / \partial v_{jq}$ en la expresión (3.4), respectivamente, y α denota la tasa de aprendizaje.

3.3.6. Resultados

El objetivo principal es realizar un recomendor Top-N, en este caso se dará un ejemplo creando un recomendador de películas utilizando movielens-100k. El tamaño de la matriz de calificaciones utilizada, para predecir calificaciones en sus entradas aún sin valorar por los usuarios, fue de 943 filas (usuarios) y 1682 columnas (elementos). El conjunto de datos movielens-100k puede ser cargados desde la librería dl2, una librería especializada para el diseño y entrenamiento de redes neuronales, puede consultar la documentación en [17]. Para hacer las predicciones de las entradas faltantes en la matriz se implementó el modelo (3.4) en google colaboratory utilizando la versión 3.7.15 de Python con una CPU Intel Xeon a 2.20 GHz y 13 GB de memoria RAM.

El modelo fue entrenado utilizando los algoritmos de optimización DGE y Adam, ya incorporados en la librería. Para el DGE se utilizó una tasa de aprendizaje $\alpha = 0.005$ y le tomó 51 minutos con 17 segundos reducir el error por debajo de 0.1 realizando 2307 iteraciones (epochs). Por otra parte, el entrenamiento con Adam demoró apenas 3 minutos y 43 segundos para reducir el error por debajo de 0.01 realizando 129 iteraciones con una tasa de aprendizaje de 0.001. La figura 3.1a muestra la evolución del error usando DGE y 3.1b la evolución del error con Adam. Para este caso particular, el método de Adam fue más rápido por un factor de 13.80 que DGE pese a reducir el error un orden más de magnitud y usar una tasa de aprendizaje 5 veces más pequeña.

El tiempo de ejecución de cada iteración fue de 1.33s y 1.73s para DGE y Adam, respectivamente. En este caso podemos visualizar el comportamiento de la pérdida de entrenamiento en función del tiempo para las figuras 3.1a y 3.1b con unidad de tiempo de 1.33s y 1.73s, respectivamente.

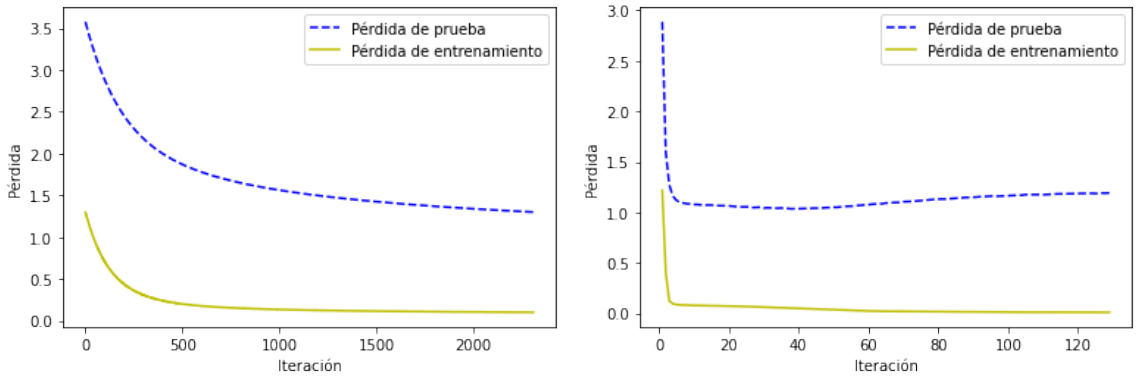
Una vez entrenada la red es bastante sencillo realizar un recomendador Top-N para usuarios con los valores predichos para las entradas ausentes de la matriz de calificaciones.

Entrada

- Identificador i de usuario, que es igual al número de fila que contiene las valoraciones predichas para el usuario en la matriz de calificaciones.

Proceso

- Se toma de la i-ésima fila de la matriz de calificaciones predichas todas las calificaciones y etiquetas de los elementos que el usuario i no ha calificado



(a) **(b)**

Figura 3.1. Comportamiento de la función de pérdida para el conjunto de entrenamiento y de prueba durante el entrenamiento de la red neuronal, en 3.1a utilizando DGE y en 3.1b Adam. Elaboración propia.

(interaccionado) en la matriz de calificaciones observadas.

- Reordenar las etiquetas de izquierda a derecha en un arreglo cumpliendo la siguiente condición: una etiqueta está a la izquierda de otra si su calificación correspondiente es mayor o igual a la calificación de la segunda etiqueta.

Salida

- Arreglo de etiquetas ordenado.

Al ser i arbitrario, ya se tiene un recomendador para un conjunto de usuarios.

CONCLUSIONES

1. La implementación y entrenamiento del modelo de factorización matricial con factor de regularización y términos de sesgo en Python permitío la creación de un sistema de recomendación Top-N debido a las predicciones, obtenidas durante el entrenamiento de la red neuronal, en la matriz de calificaciones.

2. El comportamiento de la función de pérdida se comparó para los algoritmos de optimización DGE y Adam. En ambos casos el valor de la función de pérdida tiendió a cero, como lo garantiza el método del descenso del gradiente que actualiza los pesos y sesgos en la red y el cálculo de los gradientes se debe al algoritmo de propagación hacia atrás, a medida que el número de iteraciones aumentaba. El algortimo de Adam fue más rápido que DGE necesitando una menor cantidad de iteraciones para pasar del umbral esperado, como se esperaba, pues el resto de algortimos como DGE con momento, gradiente adaptivo o propagación de la raíz cuadrada media consisten en diferentes técnicas que mejoran la convergencia de DGE y el algoritmo de Adam combina todas estas técnicas en un algoritmo de aprendizaje eficiente.

3. Los modelos de factores latentes, como la factorización matricial, es un método de reducción de la dimensionalidad que ofrece ventajas sobre otros métodos de reducción como el análisis de componentes principales (ACP). No solamente se encarga de reducir el tamaño de la dimensión, en este caso expresando la matriz de calificaciones R como un producto de dos matrices U y V de menor dimensión, sino también lidia con el problema de escasez de datos en la matriz R prediciendo las entradas faltantes. Por otra parte, el método trabaja directamente con la matriz R mientras (ACP) necesita trabajar con una matriz de covarianza y calcular los autovalores.

BIBLIOGRAFÍA

[1] C. C. Aggarwal, (2016). *Recommender Systems: The Textbook* . Springer. Nueva York.

[2] F. Ricci, L. Rokach, B. Shapira, P. B. Kantor, (2011). *Recommender Systems Handbook*. Springer, Nueva York.

[3] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich, (2011). *Recommender Systems An Introduction*. Cambridge University Press, Nueva York.

[4] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola, (2021). *Dive into Deep Learning*.

[5] J. Patterson, A. Gibson, (2017). *Deep Learning A Practitioner's Approach*. O'Reilly Media, Inc, Sebastopol.

[6] L. Rokach, (2019). *Ensamble Learning Pattern Classification Using Ensemble Methods*. 2.^{a} ed. World Scientific, Singapore.

[7] G. G. Roussas, (1997). *A Course in Mathematical Statistics*. 2.^{a} ed. Academic Press, San Diego.

[8] R. K. Sundaram, (1996). *A First Course in Optimization Theory*. Cambridge University Press, New York.

[9] G. G. Roussas, (2003). *An Introduction to Probability and Statistical Inference*. Academic Press, San Diego.

[10] S. Axler, (2015). *Linear Algebra Done Right*. 3.^{ra} ed. Springer, San Francisco.

[11] B. Barreau, (2020). *Machine Learning for Financial Products Recommendation*. Université Paris-Saclay.

[12] Reddi, S. J., Kale, S., & Kumar, S, (2019). *On the convergence of adam and beyond*. `https://arxiv.org/pdf/1904.09237.pdf`.

[13] H. Wilf, (2002). *Algorithms and Complexity*. 2.^{da} ed. Ak Peters, Ltd. USA, Massachusetts.

[14] Zaheer, M., Reddi, S., Sachan, D., Kale, S., & Kumar, S, (2018). *Adaptive methods for nonconvex optimization. Advances in Neural Information Processing Systems (pp. 9793–9803).*

[15] P-G. Ciarlet, (1989). *Introduction To Numerical Linear Algebra And Optimisation*. 1.^{ra} ed. Cambridge University Press, New York.

[16] Código de la implementación del modelo de factorización de matrices. `https://colab.research.google.com/drive/1CIKYCJ3VDGwWmYwfV51yAI98xQTxdDdJ`.

[17] Documenetación de mxnet. `https://mxnet.apache.org/versions/1.6/api/python/docs/api/`.

ANEXOS

Código de la implementación en el lenguaje Python

```
# -*- coding: utf-8 -*-
"""Factorizacion_Matrices.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1CIKYCJ3VDGwWmYwfV51yAI98xQTxdDdJ

#Instalamos la libreria mxnet

# Seccion nueva
"""

pip install mxnet==1.7.0.post1

"""#Instalacion de la libreria d2l"""

pip install matplotlib-inline

pip install -U d2l

#Importamos las librerias necesarias

from d2l import mxnet as d2l
from mxnet import autograd, gluon, np, npx
from mxnet.gluon import nn
import mxnet as mx
from mxnet import nd, context
npx.set_np()

#Implementamos el modelo de factorizacion de matrices con un termino de regularizacion
#y un termino de sesgos para usuario y elementos en una clase que hereda de mxnet.gluon
#es decir, una estructura de red neuronal
class MF(nn.Block):
  def __init__(self, num_factors, num_users, num_items, **kwargs):
    super(MF, self).__init__(**kwargs)
    self.U = nn.Embedding(input_dim=num_users, output_dim=num_factors)
    self.V = nn.Embedding(input_dim=num_items, output_dim=num_factors)
    self.user_bias = nn.Embedding(num_users, 1)
    self.item_bias = nn.Embedding(num_items, 1)
  def forward(self, user_id, item_id):
    U_u = self.U(user_id)
    V_i = self.V(item_id)
    o_u = self.user_bias(user_id)
    p_i = self.item_bias(item_id)
    outputs = (U_u * V_i).sum(axis=1) + np.squeeze(o_u) + np.squeeze(p_i)
    return outputs.flatten()

#Creamos una funcion para calcular el ERCM

def evaluator(net, test_iter, devices):

  # obtiene la Error de Raiz Cuadrada Media, esta variable es una instancia
  rmse = mx.metric.RMSE()
  rmse_list = []
  for idx, (users, items, ratings) in enumerate(test_iter):
      u = gluon.utils.split_and_load(users, devices, even_split=False)
```

```
        i = gluon.utils.split_and_load(items, devices, even_split=False)
        r_ui = gluon.utils.split_and_load(ratings, devices, even_split=False)
        r_hat = [net(u, i) for u, i in zip(u, i)]
        rmse.update(labels=r_ui, preds=r_hat)
        rmse_list.append(rmse.get()[1])
    return float(np.mean(np.array(rmse_list))) #retornamos el error

#Definimos una lista vacia donde se guardaran los datos del numero
#de iteracion (epoch) y la medicion de los errores de prueba y entrenamiento
y = []
def train_recsys_rating(net,train_iter,test_iter,loss,trainer,err,\
    devices=d2l.try_all_gpus(),evaluator=None,**kwargs):

    timer = d2l.Timer()
    aux = 9999
    epoch = 0
    #Hacemos un ciclo que termina hasta que el error de
    #entrenamiento quede por debajo del umbral indicado
    while (aux > err):
      metric, l = d2l.Accumulator(3), 0.
      for i, values in enumerate(train_iter):
        timer.start()
        input_data = []
        values = values if isinstance(values, list) else [values]
        for v in values:
          input_data.append(gluon.utils.split_and_load(v, devices))
        train_feat = input_data[0:-1] if len(values) > 1 else input_data
        train_label = input_data[-1]
        with autograd.record():
          preds = [net(*t) for t in zip(*train_feat)]
          ls = [loss(p, s) for p, s in zip(preds, train_label)]
        [l.backward() for l in ls]
        l += sum([l.asnumpy() for l in ls]).mean() / len(devices)
        trainer.step(values[0].shape[0])
        metric.add(l, values[0].shape[0], values[0].size)
        timer.stop()
      if len(kwargs) > 0:
        test_rmse = evaluator(net, test_iter, kwargs['inter_mat'], devices)
      else:
        test_rmse = evaluator(net, test_iter, devices)
      train_l = l / (i + 1)
      epoch +=1
      aux = metric[0] / metric[1]
      if epoch % 100 == 0:
        print(aux)
      y.append([epoch, test_rmse, aux])
    return y

#Empezamos el entrenamiendo del modelo

#Indicamos que se utilicen todas las gpus disponibles
#e importamos los datos de movie lens 100k


devices=d2l.try_all_gpus()
num_users,num_items,train_iter,test_iter=d2l.split_and_load_ml100k(test_ratio=0.1,batch_size=512)

#Imprimimos el numero de usuarios para verificar

print(num_users)
print(num_items)

#Creamos una instancia del modelo para iniciar el entrenamiento y calcular la perdida

net = MF(50, num_users, num_items)
net.initialize(ctx=devices, force_reinit=True, init=mx.init.Normal(0.01))

#definimos las variables de los parametros, hiperparametros y les asignamos los valores
#Tambien indicamos el optimizador a utilizar (DGS o Adam)
tasa_error, error, peso_caida, optimizador = 0.001, 0.01, 1e-5, 'Adam'

#definimos una instancia de la funcion de error L2 o ECM
perdida = gluon.loss.L2Loss()
```

```
entrenador = gluon.Trainer(net.collect_params(),optimizador,\
             {"learning_rate":tasa_error,'wd':peso_caida})
resultados = train_recsys_rating(net,train_iter,test_iter,\
               perdida,entrenador,error,devices,evaluator)

#SGD lr = 0.005 y error = 0.1
#adam lr = 0.001 y error = 0.01

#Guardamos en un .txt los valores de perdida para graficarlos posteriormente
archivo = open("adam", "w")
for epoch, y1, y2 in resultados:
  archivo.write(str(epoch) + "␣" + str(y1) + "␣" + str(y2))
  archivo.write("\n")
archivo.close()

#Creamos un recomendador Top-N

def recomendador(u_i,num_items, N):
  scores = []
  diccionario = {}

  #nos interesa la nota del usuario en la posicion u_i
  #obtenemos las calificaciones predichas para el usuario u_i a todos los elementos
  for i in range(0,num_items):
    temp = net(np.array([u_i],dtype='int',ctx=devices[0]),\
                np.array([i],dtype='int',ctx=devices[0]))
    scores.append( float(temp) )

  #Finalmente ordenamos la lista e imprimimos los resultados del recomendador Top-N
  scores = sorted(scores, reverse=True)[:N]
  print(scores)
  for i in range(0,num_items):
    calificacion = net(np.array([u_i],dtype='int',ctx=devices[0]),\
                       np.array([i],dtype='int',ctx=devices[0]))
    calificacion = float(calificacion)
    if calificacion in scores:
      diccionario[i] = calificacion
  print(diccionario)
recomendador(20,num_items, 15)
  # Link para documentacion de mxnet
  # https://mxnet.apache.org/versions/1.6/api/python/docs/api/
```