

# Agradecimientos

El desarrollo de este trabajo fue un gran reto, y no hubiese sido posible sin el apoyo de mis padres, por siempre creer en mí, y por estar ahí sin importar nada.

Agradezco mis directores de tesis, Dr Carlos Brito Loeza y Dr Ricardo Legarda Sáenz, por sus invaluable conocimientos, por sus aportes y mentoría en este trabajo.

A mis compañeros de Maestría y en particular a mi hermano gemelo Luis Ballote, no sólo por estar siempre presente sin importar la distancia, sino por sus apoyo en la discusión y revisión de mi trabajo.

Al amor de mi vida Adriana Tamayo, por sus palabras de aliento, por su compañía frente a las adversidades, y por su apoyo incondicional.

A la facultad de Matemáticas UADY, que ha sido mi casa de estudios durante varios años y donde me he formado profesionalmente.



# Índice general

<b>1. Preliminares</b>	<b>1</b>
1.1. Resultados de Cálculo y Álgebra lineal . . . . .	1
1.2. Visión Computacional . . . . .	5
1.3. Inteligencia artificial y Aprendizaje profundo . . . . .	5
1.3.1. Sobreajusto y Desajuste . . . . .	6
1.3.2. Regularización . . . . .	8
1.3.3. Función de activación. . . . .	9
1.3.4. Transferencia de aprendizaje . . . . .	10
1.3.5. Conjuntos desbalanceados . . . . .	11
<b>2. Redes Neuronales Convolucionales</b>	<b>15</b>
2.1. Convoluciones . . . . .	15
2.1.1. Stride . . . . .	17
2.1.2. Convoluciones con múltiples canales . . . . .	18
2.1.3. Padding . . . . .	19
2.1.4. Agrupación (Pooling) . . . . .	21
2.1.5. Convolución como una operación lineal . . . . .	23
2.2. Redes Neuronales Convolucionales . . . . .	23
2.2.1. Descenso del Gradiente Estocástico . . . . .	24
2.2.2. Optimizadores . . . . .	25
2.2.3. Propagación hacia atrás . . . . .	27

2.2.4.	Normalización por Lotes . . . . .	27
2.2.5.	Desvanecimiento del gradiente . . . . .	28
2.2.6.	CNNs en el Estado del Arte . . . . .	29
2.3.	ResNet . . . . .	29
2.3.1.	Inicialización de parámetros . . . . .	32
2.3.2.	Estabilidad en las redes Neuronales . . . . .	33
<b>3.</b>	<b>Ecuaciones diferenciales</b>	<b>35</b>
3.1.	Problemas de valor inicial . . . . .	36
3.1.1.	Existencia, unicidad y continuidad de soluciones . . . . .	36
3.2.	Estabilidad . . . . .	36
3.2.1.	Problema Lineal . . . . .	37
3.2.2.	IVP no lineales (El caso general) . . . . .	38
3.3.	Métodos de Runge-Kutta . . . . .	39
3.3.1.	Método de Euler . . . . .	39
3.3.2.	Método de Euler modificado . . . . .	41
3.3.3.	Método general . . . . .	41
3.3.4.	Runge Kutta de orden 4 (RK4) . . . . .	42
3.3.5.	Métodos de Runge Kutta simplécticos . . . . .	42
3.3.6.	Métodos implícitos . . . . .	43
<b>4.</b>	<b>Teoría de control óptimo</b>	<b>45</b>
4.1.	La ecuación de Hamilton-Jacobi-Bellman . . . . .	46
4.2.	Ecuación adjunta . . . . .	48
<b>5.</b>	<b>Relación entre las ecuaciones diferenciales y las ResNets</b>	<b>51</b>
5.1.	Control Óptimo para DL . . . . .	51
5.2.	DNNs como discretización de ODEs . . . . .	52
5.2.1.	Estabilidad de Redes por medio de ODEs . . . . .	53
5.2.2.	PolyNet . . . . .	54

5.2.3.	FractalNet . . . . .	55
5.2.4.	Midpoint Network . . . . .	57
5.2.5.	IMEXnet . . . . .	57
5.2.6.	Otros Avances . . . . .	58
<b>6.</b>	<b>Experimentos y Resultados</b>	<b>61</b>
6.1.	Descripción del problema . . . . .	61
6.2.	Métricas para clasificación Binaria . . . . .	62
6.2.1.	Matriz de confusión . . . . .	63
6.2.2.	Exactitud . . . . .	64
6.2.3.	Sensibilidad y Especificidad . . . . .	65
6.2.4.	Precisión . . . . .	65
6.2.5.	<b>F1-score</b> . . . . .	65
6.3.	Métricas para clasificación multiclase . . . . .	66
6.3.1.	Métricas Macro . . . . .	68
6.3.2.	Métricas Ponderadas . . . . .	68

# Capítulo 1

## Preliminares

Antes de poder hablar del Aprendizaje profundo, o de Redes Neuronales Convolucionales, es importante sentar las bases matemáticas y computacionales requeridas para su estudio. A continuación, se presentan algunas definiciones y resultados conocidos de temas tales como Cálculo, Álgebra y Ciencias de la Computación.

### 1.1. Resultados de Cálculo y Álgebra lineal

**Definición 1.1.** Sean  $A$  y  $B$  dos conjuntos. El producto cartesiano  $A \times B$  se define como

$$A \times B := \{(a, b) : a \in A \text{ y } b \in B\}. \quad (1.1)$$

De igual manera sean  $A_1, \dots, A_s$  son conjuntos, el producto cartesiano se define como

$$A_1 \times A_2 \times \dots \times A_s = \{(a_1, \dots, a_s) : a_i \in A_i, \quad i = 1, \dots, s\}. \quad (1.2)$$

De modo natural, se define  $A^n := A \times A \times \dots \times A$  donde  $A$  se repite  $n$  veces.

A su vez, es conveniente destacar otra notación

**Notación 1.2.** Sean  $a_1, \dots, a_n \in \mathbb{N}$ . Denotamos  $\mathbb{R}^{a_1 \times a_2 \times \dots \times a_s} := \mathbb{R}^{a_1} \times \dots \times \mathbb{R}^{a_s}$ .

**Definición 1.3.** Sean  $u, v \in \mathbb{R}^n$ , el producto punto de  $u$  con  $v$  denotado como  $u \cdot v$  es

$$u_1 v_1 + u_2 v_2 + \dots + u_n v_n, \quad (1.3)$$

donde  $u_i$  y  $v_i$  representan la  $i$ -ésima entrada de los vectores  $u$  y  $v$  respectivamente.

**Definición 1.4** (Polinomio de Taylor). Sea  $f$  una función  $n$  veces diferenciable, y sea

$$a_k = \frac{f^{(k)}(a)}{k!}.$$

Se define el Polinomio de Taylor de grado  $n$  para  $f$  en  $a$  como

$$P_{n,a}(x) = a_0 + a_1(x - a) + \cdots + a_n(x - a)^n.$$

dónde  $f^{(k)}$  representa la  $k$ -ésima derivada de  $f$ .

El polinomio  $P_{n,a}(x)$  ha sido definido de manera que  $P_{n,a}(a) = f(a)$  para  $0 \leq k \leq n$ .

**Definición 1.5.** Sea  $f$  una función tal que  $P_{n,a}(x)$  existe, definimos el término residual  $R_{n,a}(x)$  como

$$R_{n,a}(x) = f(x) - P_{n,a}(x). \quad (1.4)$$

**Teorema 1.6** (Teorema de Taylor). Supóngase que  $f^{(1)}, f^{(2)}, \dots, f^{(n+1)}$  están definidos en  $[a, x]$ . Entonces

$$R_{n,a}(x) = \frac{f^{(n+1)}(t)}{(n+1)!} (x - a)^{n+1} \quad (1.5)$$

para algún  $t \in (a, x)$ .

Para clasificar las imágenes, los algoritmos de inteligencia artificial extraen características importantes de una imagen. ¿Qué puede ser tomado en cuenta como importante? Podría pensarse que las esquinas, los bordes u otros detalles. Sin embargo, los algoritmos de aprendizaje automático suelen ser tomadas como cajas negras, dónde las características relevantes de una imagen, a simple vista no parezcan intuitivas bajo la percepción humana.

**Definición 1.7** (Característica). Una característica  $x$  es un elemento de  $\mathbb{R}^{w \times h \times d}$ , dónde  $w, h \in \mathbb{N}$  representan las dimensiones espaciales y  $d$  la cantidad de canales.

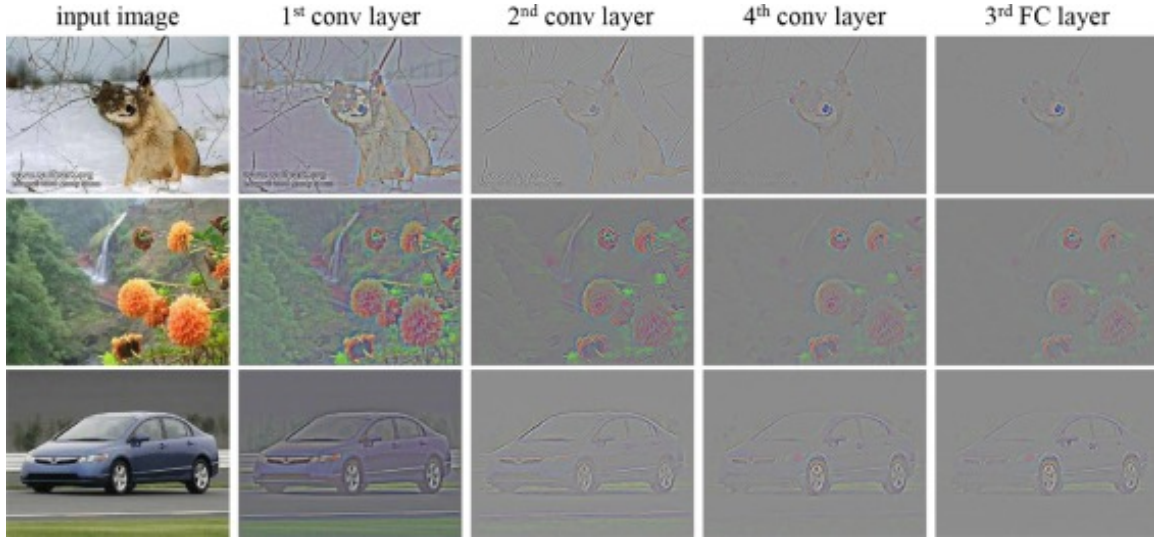


Figura 1.1: En las redes neuronales convolucionales, las imágenes pasan por múltiples filtros. El resultado de cada convolución, es una característica.

**Definición 1.8.** Sea  $x$  una característica en  $\mathbb{R}^{h \times w \times d}$ . La vectorización de  $x$ , denotada  $X = \text{vec}(x)$  es un vector en  $\mathbb{R}^{hwd}$  tal que

$$X_{(k-1)hw+(i-1)w+j} = x_{i,j,k}, \quad (1.6)$$

para todos  $i = 1, 2, \dots, h$ ,  $j = 1, 2, \dots, w$ ,  $k = 1, 2, \dots, d$ .

**Definición 1.9.** Una matriz  $H$  es llamada Circulante si es de la forma

$$H = \begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} \\ h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & & \vdots \\ h_1 & h_2 & \cdots & h_0 \end{bmatrix} \quad (1.7)$$

ya que  $H$  queda completamente determinada con la primera fila, es posible escribir  $H = \text{circ}(h_0, \dots, h_{N-1})$ .

**Definición 1.10.** Una matriz  $H$  es llamada Doblemente Circulante por Bloques si es de



la forma

$$H = \begin{bmatrix} H_0 & H_1 & \cdots & H_{N-1} \\ H_{N-1} & H_0 & \cdots & H_{N-2} \\ \vdots & \vdots & & \vdots \\ H_1 & H_2 & \cdots & H_0 \end{bmatrix} = \text{circ}(H_0, \dots, H_{N-1}) \in \mathbb{R}^{N^2 \times N^2} \quad (1.8)$$

dónde  $H_i = \text{circ}(h_{i,0}, \dots, h_{i,N-1}) \in \mathbb{R}^{N \times N}$ .

**Definición 1.11.** Una función vectorial es una función de la forma  $f : \mathbb{R} \rightarrow \mathbb{R}^n$ .

**Definición 1.12.** La derivada  $f'$  de una función vectorial  $f : \mathbb{R} \rightarrow \mathbb{R}^n$  se define como

$$f'(t) = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}. \quad (1.9)$$

Un teorema útil para encontrar derivadas vectoriales es el siguiente:

**Teorema 1.13.** La derivada  $f'$  de una función vectorial  $f = (f_1, f_2, \dots, f_n)$ , con  $f_i : \mathbb{R} \rightarrow \mathbb{R}$  está dada por

$$f'(t) = (f'_1(t), f'_2(t), \dots, f'_n(t)). \quad (1.10)$$

**Definición 1.14** (Gradiente). Sea  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . El gradiente de  $f$  es el vector conformado por las derivadas parciales en cada una de las  $n$  variables:

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right). \quad (1.11)$$

**Proposición 1.15** (Regla de la cadena). Sea  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  y  $v : \mathbb{R} \rightarrow \mathbb{R}^n$ . Es posible hallar la derivada de la composición.

$$\frac{d}{dt}(f \circ v) = \nabla f(v(t)) \cdot v'(t) \quad (1.12)$$

**Definición 1.16** (Notación O-grande (Big O)). Sean  $f$  y  $g$  funciones  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ . Decimos que  $f(n) = \mathcal{O}(g(n))$  si existen enteros positivos  $c$  y  $n_0$  tales que para cualquier entero  $n \geq n_0$ ,

$$f(n) \leq cg(n). \quad (1.13)$$

Cuando  $f(n) = \mathcal{O}(n)$ , decimos que  $g(n)$  es una cota asintótica superior de  $f(n)$ .

## 1.2. Visión Computacional

El problema de clasificación de imágenes, pertenece a la rama de la visión computacional. La siguiente definición de imagen fue extraída del libro [1].

**Definición 1.17.** *Una imagen digital es una función  $I : D \subset \mathbb{Z}^2 \rightarrow U$ . Dónde el dominio es un rectángulo con coordenadas enteras  $D = [1, M] \cap \mathbb{Z} \times [1, N] \cap \mathbb{Z}$ . La terna  $(x, y, u)$  se conoce como pixel.*

- *Cuando las imágenes son a escala de grises se tiene que  $U = [0, 255] \cap \mathbb{Z}$ . Es decir que  $u \in U$  es un valor de 0 a 255.*
- *Si las imágenes son a color, consideramos  $U = ([0, 255] \cap \mathbb{Z})^3$ . Es decir que  $u \in U$  es una terna de valores de 0 a 255. A cada entrada de la terna se conoce como un canal de color.*

Por otro lado, las imágenes, son almacenadas en estructuras de datos conocidas como tensores. Los tensores son arreglos multidimensionales, utilizados en Aprendizaje de Máquina (ML por sus siglas en inglés) para almacenar datos. Es posible representar una imagen digital  $I$  como un Tensor de tres dimensiones.

$$T_{i,j,k} = u_k, \quad (u_1, u_2, u_3) = I(i, j), \quad k = 1, 2, 3 \quad (1.14)$$

## 1.3. Inteligencia artificial y Aprendizaje profundo

El Aprendizaje de Máquina es un caso particular de la inteligencia artificial, en donde la máquina aprende de los datos proporcionados. Para entender este concepto es importante definir qué significa aprender.

**Definición 1.18.** *Se dice que un programa de computadora aprende de la experiencia  $E$  respecto a un conjunto de tareas  $T$  y medida de rendimiento  $P$  si el rendimiento en las tareas en  $T$ , medido con  $P$  mejora gracias a la experiencia  $E$  [2].*

Ya que aprender, implica mejorar el rendimiento en una tarea específica, es pertinente remarcar algunas de las tareas más comunes en el campo del aprendizaje automático.

1. **Clasificación.** Consiste en que el modelo reconozca que elementos pertenecen a ciertas clases, teniendo en cuenta sus características. Por ejemplo, distinguir perros de gatos, o en el caso de este proyecto, diferenciar, tipos de polen y polvo.
2. **Regresión.** Aquí lo que se busca es usar la información existente para tener una predicción aproximada de algún escalar. Por ejemplo, predecir el precio de una casa dadas sus características (ubicación, número de cuartos, antigüedad, etc).
3. **Agrupamiento (Clustering).** Este es un método de aprendizaje que tiene como objetivo detectar similitudes entre las instancias, y separarlas en grupos de elementos similares entre sí.

### 1.3.1. Sobreajusto y Desajuste

En la mayoría de las tareas que puede realizar un modelo de aprendizaje automático, lo que se busca es encontrar una función  $f : D \rightarrow R$ . donde nuestro dominio  $D$  pueden ser imágenes, tensores, o sólo una tabla con datos relevantes. Por ejemplo, nuestra función  $f$  podría ser la temperatura  $f(x)$  de mañana donde  $x$  es un vector con información sobre la presión atmosférica, la temperatura de ayer, y la velocidad del viento. Por supuesto, la función  $f$  es desconocida, y el aprendizaje automático sólo pretende aproximarse lo mejor posible.  $D$  podría ser un conjunto infinito, muy grande, o simplemente poco explorado.

Asumiendo que se tiene un conjunto finito  $\mathcal{X}_{all} = \{(x, f(x)) : x \in \mathcal{X}_{all} \subseteq D\}$ . Nuestro objetivo será aproximar  $f$  haciendo uso de los datos conocidos. Un buen intento inicial es  $\hat{f} : D \rightarrow R$ , dado por  $\hat{f}(x) = f(x)$ . Sin embargo, esto no nos da información con respecto a los valores fuera de  $\mathcal{X}_{all}$ . Claro que si  $D$  fuesen los reales, bastaría con hacer una interpolación, o incluso se puede definir

$$\hat{f}(x) = \begin{cases} f(x) & \text{si } x \in X \\ 0 & \text{si } x \notin X \end{cases} \quad (1.15)$$

Esto por supuesto sería una función que predice perfectamente los valores  $f(x)$ . Sin embargo, es claro que cualquier  $x \notin X$  generará un resultado incorrecto. Cuando ocurre esto, decimos que el modelo no generaliza bien. Es decir, tiene *sobreajuste*.

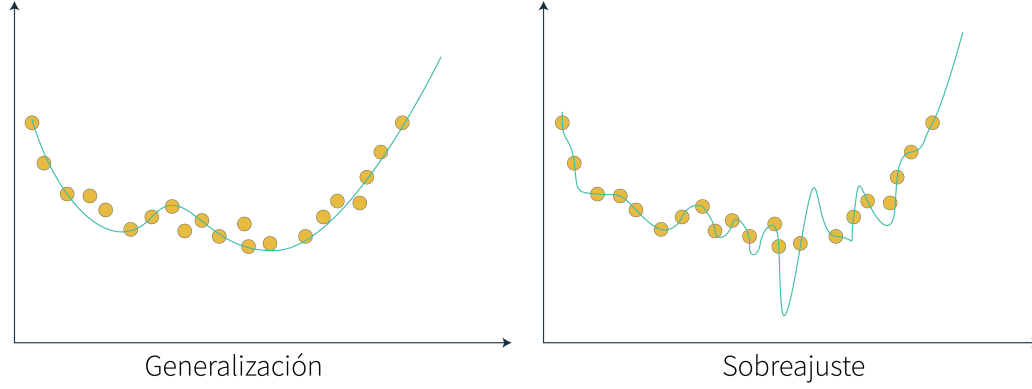


Figura 1.2: Cuando un modelo tiene sobreajuste se dice que memorizó las etiquetas de los datos, en vez de encontrar patrones para generalizar.

Para evitar el sobreajuste, es importante que nuestro modelo entrene utilizando un conjunto  $\mathcal{X}$  y sea evaluado con un conjunto de prueba  $\mathcal{X}'$ , cuyos datos no hayan influido en la aproximación de  $\hat{f}$ . Para determinar cuán buena fue nuestra aproximación, se utiliza una función denominada *función de costo*. Una de las funciones de costo más comunes es la raíz cuadrada media RMSE.

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}} \quad (1.16)$$

dónde  $\hat{y}_i = \hat{f}(x_i)$  y  $\{(x_i, y_i)\}_{i=1}^N$  puede ser el conjunto de prueba o el de entrenamiento. Los modelos suelen minimizar la función de costo en el conjunto de entrenamiento, con la esperanza de a su vez minimizar el error de prueba.

Sin embargo, el conjunto de funciones que puede adoptar un modelo, depende de la cantidad de parámetros que este tenga, lo cual se conoce como *capacidad*. Con una

capacidad suficiente, siempre es posible hacer un mapeo perfecto entre entradas y etiquetas. Sin embargo, no es la función que se adapte mejor a los datos de entrenamiento la que es más conveniente, sino la que generalice mejor. Cuando la capacidad de un modelo es muy reducida, puede ocurrir que no sea posible reducir el error de entrenamiento y con ello se dice que el modelo sufre de un *desajuste*.

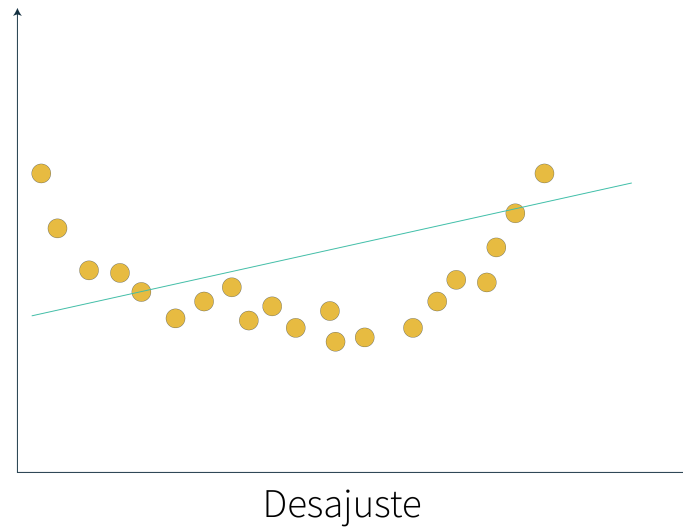


Figura 1.3: Cuando el error de entrenamiento es muy grande, la causa podría ser un desajuste.

### 1.3.2. Regularización

Cuando se entrena un modelo, son necesarias dos partes:

1. Optimización: en donde, se busca minimizar la función de costo.
2. Generalización: es importante que las predicciones de nuestro modelo sean aplicables con datos que no hayan sido vistos en el entrenamiento.

Sin embargo, no es fácil conseguir ambos objetivos simultáneamente. En vez de priorizar alguno de los dos, la *regularización* pretende lograr que se satisfagan ambos de la mejor ma-

nera. Es decir, *la regularización es cualquier modificación en el algoritmo de entrenamiento cuyo objetivo sea reducir el error de generalización, pero no el error de entrenamiento.*

Una de las formas más comunes de regularizar un modelo que aprende la función  $f(x, \theta)$ , es agregando una penalización  $R(\theta)$  a la función de costo, donde  $\theta$  son los parámetros del modelo.

### 1.3.3. Función de activación.

Con la intención de ampliar el conjunto de funciones que un modelo puede predecir, en cada capa de las redes neuronales se incluye una función no lineal denominada *función de activación*. En caso de no incluirla, el conjunto de funciones disponibles para nuestro modelo serían sólo funciones lineales, debido a que la composición de dos funciones lineales  $L_1$  y  $L_2$  da lugar a otra función lineal  $L_1 \circ L_2$ .

Existen muchos ejemplos de funciones no lineales. En un principio se usaban funciones suaves como la función *sigmoidal*  $(1 + \exp(x))^{-1}$ , la tangente hiperbólica  $\tanh(x)$ , o la función *Softplus*  $\frac{1}{\beta} \log(1 + \exp(\beta x))$ . Sin embargo, los modelos recientes utilizan funciones no suaves como es el caso de la más popular, *ReLU*  $\max(x, 0)$ .

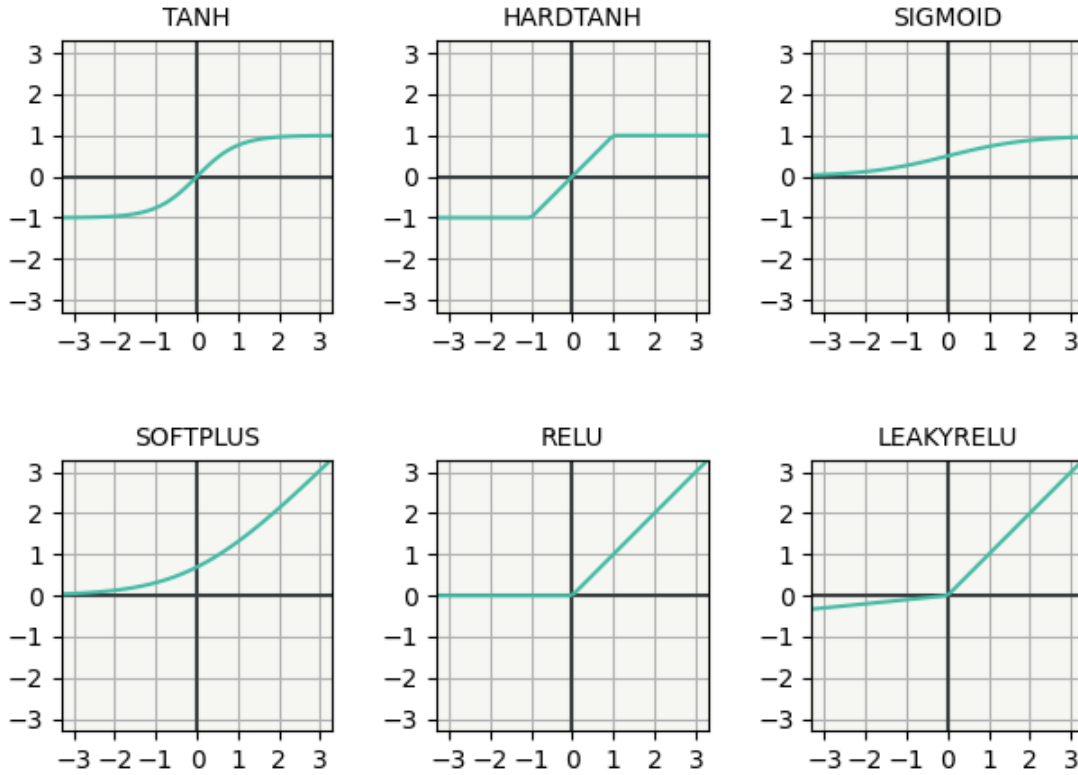


Figura 1.4: Funciones de activación comunmente utilizadas

### 1.3.4. Transferencia de aprendizaje

La transferencia de aprendizaje en el campo del aprendizaje profundo, es un concepto motivado por la psicología, que se basa en usar conocimientos adquiridos para cierta tarea  $A$ , con el objetivo de aprender alguna tarea  $B$  [3]. Por ejemplo, si una persona sabe bailar salsa, posiblemente pueda usar esos conocimientos para aprender a bailar bachata.

En la mayoría de los casos, los conjuntos de datos tienen una cantidad limitada de imágenes etiquetadas. Una posible solución a este problema, es usar *aprendizaje semisupervisado*, el cuál requiere algunas imágenes etiquetadas, pero también se pueden usar varias imágenes no etiquetadas. Sin embargo, incluso los conjuntos de datos no etiquetados, pueden ser insuficientes. Cuando se tienen conjuntos de datos muy pequeños, es

posible apoyarse de modelos pre-entrenados con millones de imágenes, con la esperanza de que las características aprendidas anteriormente, sean de utilidad en el nuevo conjunto de datos.

### 1.3.5. Conjuntos desbalanceados

Cuando se enfrenta el problema de clasificación, digamos con  $m$  clases, uno esperaría que tengamos suficientes ejemplos de cada clase. Más aún, para que nuestro modelo no tenga ningún sesgo por ninguna clase, es preferible que en nuestro conjunto de entrenamiento, todas las clases tengan una cantidad similar de ejemplos. De lo contrario, nuestro modelo podría ser entrenado con demasiados elementos de una clase particular  $\mathcal{C}$ , y tendería a clasificar muchos elementos en la clase  $\mathcal{C}$ . Además de esto, si nuestro conjunto de prueba también está desbalanceado, no se vería reflejado el sesgo en métricas comunes como la exactitud y la precisión.

**Definición 1.19.** Sea  $\mathcal{C}$  un conjunto de datos, cuyas clases son  $\mathcal{C}_1, \dots, \mathcal{C}_m$ . Un conjunto de datos se dice balanceado con tolerancia de  $\epsilon$  si

$$1 - \epsilon \leq \frac{|\mathcal{C}_i|}{|\mathcal{C}_j|} \leq 1 + \epsilon, \quad i, j = 1, 2, \dots, m. \quad (1.17)$$

Un conjunto es desbalanceado bajo la tolerancia  $\epsilon$  si no es balanceado.

Para efectos prácticos, diremos que un conjunto balanceado con tolerancia  $\epsilon = 0,1$

**Ejemplo 1.20.** Supóngase ahora que dada una lista de características, se quiere clasificar a las personas infectadas de COVID-19. En México el índice de positividad en la semana 46 fue del 17% [4]. Por lo que si se toman todas las pruebas realizadas se tendría la siguiente razón:

$$\frac{|\mathcal{C}_{Neg}|}{|\mathcal{C}_{Pos}|} = 4,8823 > 1,1. \quad (1.18)$$

El tamaño de la clase de pruebas negativas es casi 5 veces mayor al tamaño de la clase de pruebas positivas. Por consiguiente, estaríamos en presencia de un conjunto de datos desbalanceado.



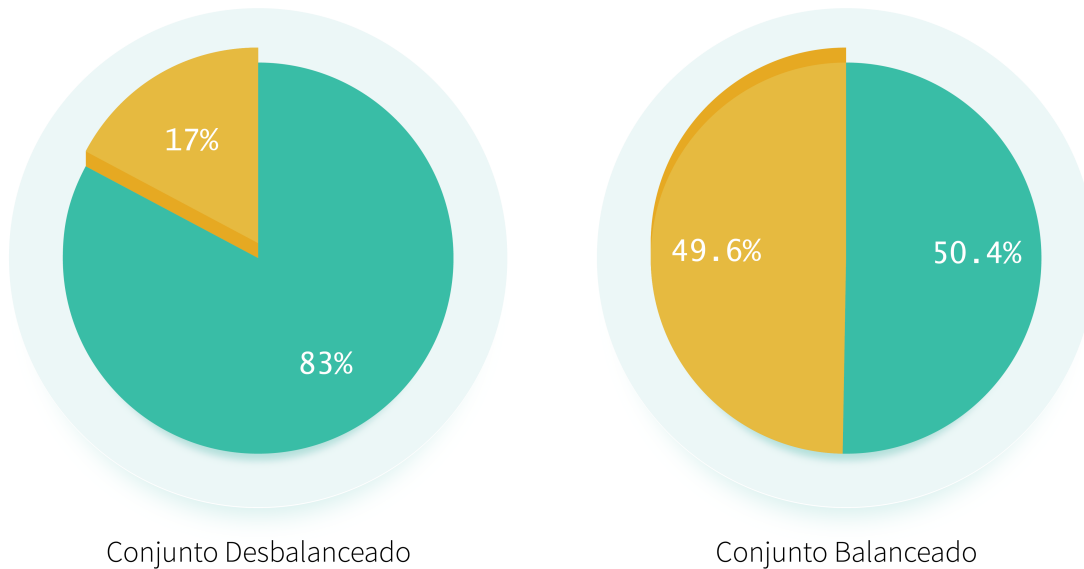


Figura 1.5: Ejemplos de conjuntos balanceados y desbalanceados.

### Afrontar un problema con un conjunto desbalanceado

La forma principal de lidiar con un conjunto desbalanceado es, valga la redundancia *balancear el conjunto*. Es decir, forzar al conjunto a tener clases cuyos tamaños coincidan. Para conseguir esto, es posible seguir distintas estrategias.

- **Submuestreo.** Consiste en tomar la clase mayoritaria (la clase con menor número de elementos) y eliminar algunas instancias, de modo que el conjunto quede balanceado. La manera más sencilla, es tomando elementos aleatorios del conjunto. Sin embargo, esto puede retirar instancias que contengan información esencial, por lo que sólo es recomendable en caso de tener un conjunto de datos muy grande.

Otros algoritmos, heurísticos se basan en dos diferentes modelos de teoría de ruido. Algunos investigadores consideran que las instancias cercanas a los márgenes de clasificación de dos clases, son consideradas ruido. Por otro lado, algunos investigadores consideran que las instancias presentes en vecindades de varias etiquetas distintas, se

pueden considerar como ruido. De modo que en lugar de remover elementos aleatorios, estas estrategias implican hacer el submuestreo tomando en cuenta únicamente las instancias que no sean ruido.

- **Sobremuestreo.** Al igual que en el submuestreo, existe el sobremuestreo aleatorio, el cual se basa en crear copias idénticas de las instancias de la clase minoritaria, de modo que la nuestra clase minoritaria alcance la magnitud del resto de clases. Al aplicar este método, se incrementa el riesgo de sobreajustar el modelo.

Otros algoritmos de sobremuestreo, generan instancias sintéticas para incrementar la cantidad de datos en una clase. Existen diversas formas de conseguir esto. Una posibilidad, es usando el aumento de datos únicamente en la clase minoritaria, permitiendo rotaciones, traslaciones y otras transformaciones. Existe también algoritmos como el VAE [5], SMOTE [6], MSMOTE [7].



Figura 1.6: Representación gráfica del submuestreo y el sobremuestreo.

### Métricas distintas a la exactitud

Como hemos mencionado antes, la exactitud no debe ser la única métrica a considerar en un problema con conjunto de datos desbalanceado. La razón es simple: es posible obtener

una muy buena exactitud sin realmente hacer predicciones de nuestra clase minoritaria.

**Ejemplo 1.21.** *Si tenemos un conjunto de datos con dos clases, tal que una clase  $C_1$  representa el 99% de las instancias, y la clase  $C_2$  el 1% restante. Si tomamos el clasificador  $f: \mathcal{C} \rightarrow \{1, 2\}$ ,  $f(x) = 1$ . Claramente nuestra exactitud sería del 99%, pero las predicciones de nuestra clase minoritaria aciertan un 0% de las veces.*

Una forma de sobrellevar el efecto visto en el ejemplo 1.21 es analizando el desempeño del clasificador para cada clase por separado y luego hacer un promedio. En el caso de nuestro ejemplo todas las instancias que en verdad pertenecen a la clase 1, fueron clasificadas correctamente (1.0), y las instancias que pertenecen a la clase 2, fueron clasificadas todas incorrectamente (0.0). Con lo con esta nueva métrica tendríamos una puntuación de  $\frac{1,0+0,0}{2} = 0,5$ . En el capítulo 6 se definen formalmente las métricas relevantes para este trabajo. Sin embargo, las métricas más utilizadas para este tipo de problemas son las siguientes:

- Confussion matrix: De ésta matriz, se pueden obtener métricas útiles tales como Especificidad, Sensitividad, Precisión, Recall
- F1-score: La media armónica de precisión y recall
- ROC curves
- Logloss
- Kappa: Exactitud de la clasificación, normalizada por el desbalance de clases en nuestros datos.

## Capítulo 2

# Redes Neuronales Convolucionales

Las primeras Redes Neuronales Convolucionales (CNN por sus siglas en inglés) datan de 1980 por Fukushima [8] y 1998 desarrollada por Yann Lecun [9]. Sin embargo, no fue hasta el 2012 con la introducción de la AlexNet [10] que la comunidad científica empezó a destinar recursos considerables a su estudio. Con el propósito de entender a profundidad a las CNNs, definiremos lo que es una convolución. Para un estudio más detallado es posible consultar [11, 12].

### 2.1. Convoluciones

La convolución es una operación binaria que tiene muchas aplicaciones en los campos de matemáticas y física [13], tales como álgebra lineal y procesamiento de señales. En cuanto a Inteligencia Artificial se refiere, las convoluciones son utilizadas para construir arquitecturas invariantes ante las traslaciones [14]. A continuación se definen matemáticamente las convoluciones y en la Sección 2.2 se definen las CNNs.

**Definición 2.1** (Convolución). *Sean  $f, g : \mathbb{R} \rightarrow \mathbb{R}$ . La convolución de  $f$  con  $g$ , denotada como  $f * g$  se define como:*

$$(f * g)(t) = \int_0^t f(\tau)g(t - \tau)d\tau$$

Uno de las propiedades de las convoluciones es que son invariantes ante las traslaciones [14].

**Proposición 2.2.** *Sea  $T_a f$  el operador de traslación definido por  $T_a f(t) = f(t + a)$ . Se tiene que*

$$T_a(I * K) = (T_a f) * K. \quad (2.1)$$

Sin embargo la Definición 2.1 no es muy práctica para los paradigmas computacionales, pues las limitaciones físicas nos obligan discretizar la integral a modo de suma. Además, las imágenes son funciones cuyo dominio es subconjunto de  $\mathbb{R}^2$ . Por lo que es necesario extender la definición 2.1 a más de una dimensión.

**Definición 2.3** (Convolución de una imagen con un kernel). *Sea  $I$  una imagen y  $K$  un kernel bidimensional. La convolución de  $I$  con  $K$  es la imagen  $(I * K)$  definida como:*

$$(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n).$$

En varias bibliotecas de aprendizaje automático se implementa una correlación cruzada en lugar de una convolución. La única diferencia entre una correlación cruzada y una convolución es la orientación del Kernel. En este trabajo, adoptaremos la convención usual de referirnos a las correlaciones como convoluciones.

**Definición 2.4.** *Sea  $I$  una imagen y  $K$  un kernel bidimensional. La correlación cruzada de  $I$  con  $K$  es la imagen definida por*

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n).$$

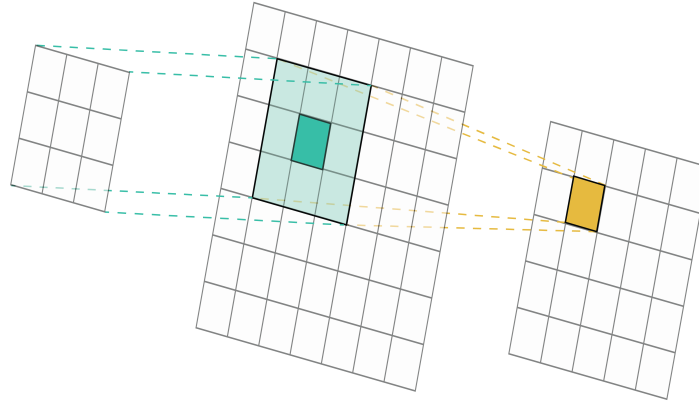


Figura 2.1: Representación gráfica de una convolución. Para cada posición del Kernel sobre la imagen, se realiza una multiplicación *entrada por entrada* del Kernel y la un conjunto de pixeles de la imagen, del mismo tamaño que el Kernel.

Cuando calculamos la correlación cruzada, desplazamos el kernel 1 pixel a la vez. Sin embargo, es posible aumentar la cantidad de pixeles que avanza el kernel en cada iteración. A esto se le conoce como tamaño de paso

### 2.1.1. Stride

**Definición 2.5.** Sea  $x$  una característica, y  $K$  un kernel. Denotamos la correlación cruzada con tamaño de paso  $a$ , de la siguiente manera

$$(I * K)|_a = (I * K)(i, j) = \sum_m \sum_n I(i + m + a \cdot i, j + n + a \cdot j) K(m, n) \quad (2.2)$$

Debido a las convenciones en la notación, a la correlación cruzada con tamaño de paso  $a$ , también se le llamará *convolución con tamaño de paso  $a$* .

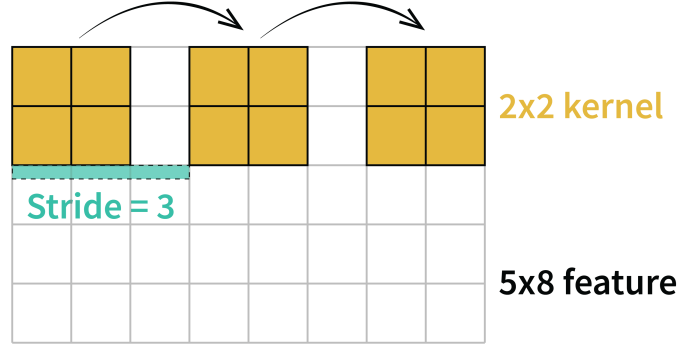


Figura 2.2: Convolución con tamaño de paso 2

### 2.1.2. Convoluciones con múltiples canales

Las imágenes RGB suelen tener 3 canales, cada uno representa la intensidad de cada color: rojo, verde y azul respectivamente. Más aún, en las redes modernas, se suelen hallar características con múltiples canales, en cada capa. Es por esto, que es imperativo definir convoluciones para características con más de un canal. Antes de definir lo que es una convolución para múltiples canales, definiremos un concepto más general de *kernel*.

**Definición 2.6.** Decimos que un kernel  $K$  multicanal tiene  $d_1$  canales de entrada y  $d_2$  canales de salida cuando  $K \in \mathbb{R}^{k \times k \times d_1 \times d_2}$ .

$$K = \begin{pmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,d_2} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,d_2} \\ \vdots & \vdots & \ddots & \vdots \\ K_{d_1,1} & K_{d_1,2} & \cdots & K_{d_1,d_2} \end{pmatrix}$$

Cada  $K_{i,j} \in \mathbb{R}^{k \times k}$  representa un subfiltro de  $K$ .

Cuando no exista ambigüedad, los kernel multicanal también pueden ser denotados como kernel.

**Definición 2.7.** Sean  $x \in \mathbb{R}^{h \times w \times d_1}$  una característica y  $K \in \mathbb{R}^{k \times k \times d_1 \times d_2}$  un kernel con  $d_1$  canales de entrada y  $d_2$  canales de salida. Denotamos la convolución 2D de  $x$  y  $K$  como:

$$y_j := \sum_{i=1}^{d_1} K_{i,j} \otimes x_i, \quad j = 1, \dots, d_2. \quad (2.3)$$

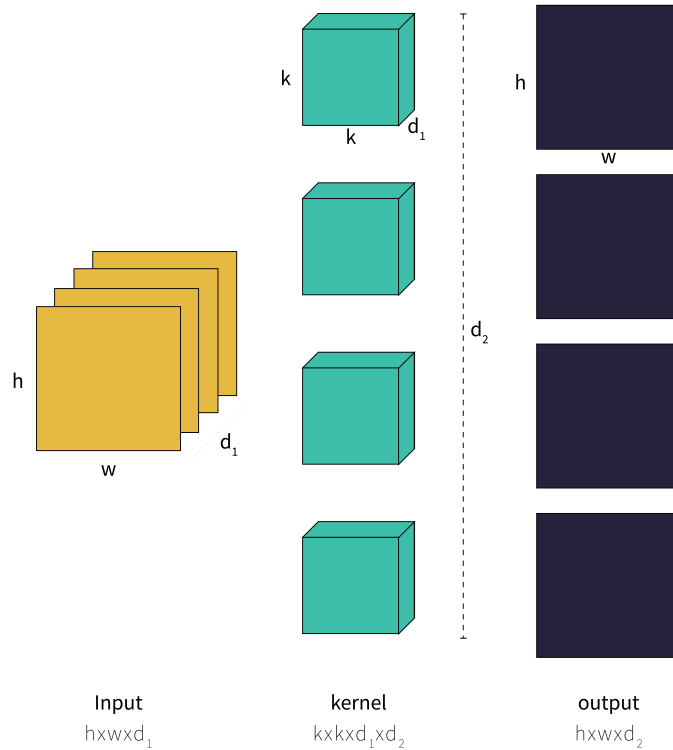


Figura 2.3: La convolución 2D puede ser usada con características de  $d_1$  canales y devolver una característica de  $d_2$  canales.

### 2.1.3. Padding

Debido a que la operación de convolución utiliza píxeles adyacentes, no es posible calcular los bordes de la imagen resultante. Suponiendo que se tiene una imagen de  $n \times n$  y un kernel de  $k \times k$  con  $k$  impar. Sólo es posible calcular la parte interior de la imagen resultante de tamaño  $n - k + 1$ .



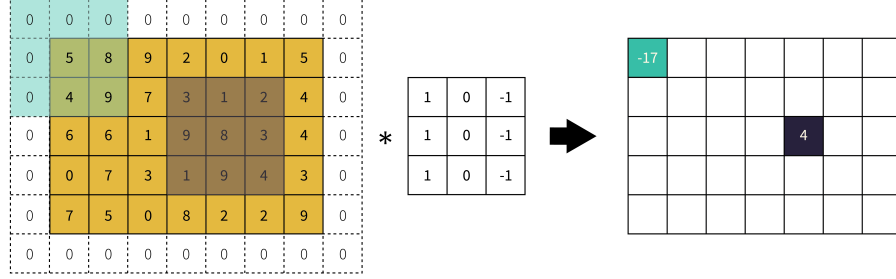


Figura 2.4: Usar una imagen propia

Para solucionar este problema, en vez de hacer la convolución con la imagen original  $I$ , extendemos la imagen hacia todos lados  $\frac{k-1}{2}$  pixeles y realizamos una nueva convolución con la imagen extendida  $\tilde{I}$ . La nueva imagen  $\tilde{I}$  contiene la misma información que la imagen original  $I$ , por tanto no importa la información que se añada en los nuevos pixeles. Sin embargo varias estrategias heurísticas han sido desarrolladas con el paso de los años:

- **Constant Padding.** Cuando se agrega un valor  $c$  en todo el borde. Cuando  $c = 0$  se conoce como **Zero Padding**.
- **Zero Padding.** Una posible opción y de las más utilizadas, es agregar ceros en el borde. En [15] se descubrió que utilizando este tipo de **padding** codifica cierto grado de información de las posiciones absolutas. Efecto que no ocurre cuando no se utiliza ningún tipo de **padding**.
- **Reflection Padding.** Este padding se basa en reflejar los valores con respecto al borde [16].
- **Symmetric Padding.** Este padding es muy similar al **reflection padding**. En cada fila toma todos los valores y los invierte. Siendo esto lo que aparece en los bordes.

**Definición 2.8.** Sea  $x \in \mathbb{R}^{h \times w}$  una característica. Sea  $\tilde{x} \in \mathbb{R}^{(h+2s) \times (w+2r)}$  con  $s < h$  y  $r < w$  la característica extendida. Definimos algunos de los **padding**s como sigue:

1. *Zero Padding:*

$$\tilde{x}_{i,j} = \begin{cases} x_{i-s,j-r} & \text{si } s < i < h+s \quad \text{y} \quad r < j < w+r \\ 0 & \text{en otro caso} \end{cases}. \quad (2.4)$$

2. *Reflection Padding:*

$$\tilde{x}_{i,j} = \begin{cases} x_{i-s,j-r} & \text{si } s < i < h+s \quad \text{y} \quad r < j < w+r \\ x_{i-s,r-j+2} & \text{si } j < r \\ x_{s-i+2,j-r} & \text{si } i < s \\ x_{2h+s-i,j-r} & \text{si } i > h+s \\ x_{i-s,2w+r-j} & \text{si } j > w+r \end{cases}. \quad (2.5)$$

En [16] tras un estudio exhaustivo en el Image Net concluyen que el **symmetric padding** y el **reflection padding** obtienen peores resultados que el **Zero padding**.

## 2.1.4. Agrupación (Pooling)

Una vez que la red encuentra características en una imagen, es posible que algunas secciones relevantes sean más grandes que otras. Por eso mismo, se implementa el **Down-sampling**, es decir reducir el tamaño de las imágenes, para que nuestro kernel pueda encontrar cosas de distintos tamaños conforme las capas van reduciendo el tamaño de la imagen. La pregunta que surge es, ¿Cómo reducimos el tamaño de una imagen? Sea como sea, siempre algo de información se pierde. Sin embargo existen métodos para resumir la información de varios píxeles en un sólo pixel. Algunos de estos métodos son

- **Max pooling.** Dada una vecindad rectangular de píxeles, la **Max pooling** consiste en tomar el máximo valor dentro de esta vecindad.
- **Average pooling.** Al igual que en el anterior caso, se toma una vecindad rectangular de píxeles. Sin embargo, en este caso se toma el promedio de todos los píxeles.
- **Un punto intermedio.** Considérese  $v$  el vector de píxeles que debe reducirse a un sólo pixel . En [17] se analiza el uso de diferentes agrupaciones, y se obtienen distintas

parametrizaciones para hallar el valor del pixel  $f_P(v)$  que sintetiza la información de  $v$ . Una posible parametrización es  $f_P(v) = \left(\frac{1}{d} \sum_{i=1}^d v_i^P\right)^{\frac{1}{P}}$ . Donde  $P = 1$  es el **average pooling** y si  $P \rightarrow \infty$  es el **max pooling**. Por tanto, es posible utilizar otros valores de  $P$  para obtener **poolings** diferentes.

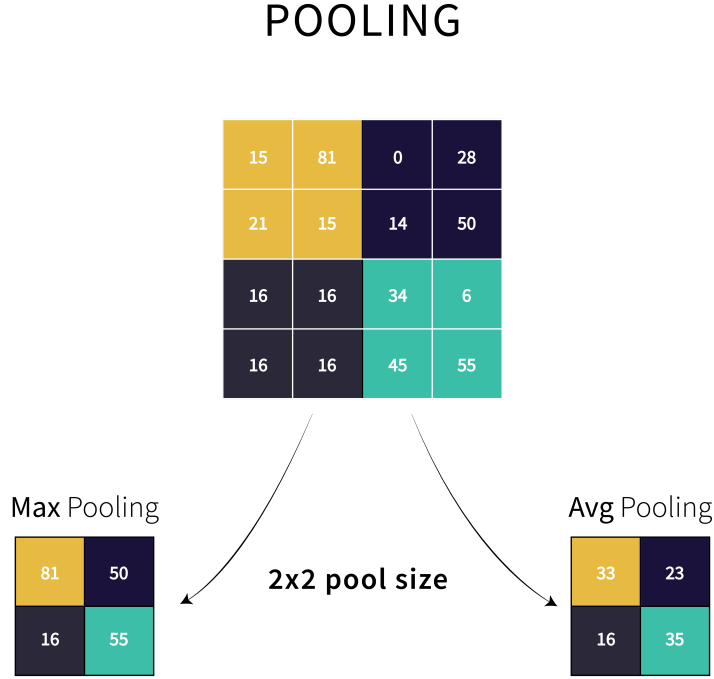


Figura 2.5: EJEMPLO DE MAXPOOLING Y AVERAGEPOOLING CON REGIONES RECTANGULARES DE  $2 \times 2$ .

También es posible en vez de hacer un **pooling** hacer una convolución con cierto tamaño de paso, con la intención de reducir el tamaño de la imagen. Esto no ha remplazado al **maxpooling** pero en la literatura se observan resultados prometedores.

### Global pooling layers

Además de hacer un **downsampling**, también es común que en algún punto de la red, se reduzca una característica a un valor escalar, o en caso de una imagen con  $d$  canales es posible obtener un vector en  $\mathbb{R}^d$  [11].

**Definición 2.9.** Sea  $x \in \mathbb{R}^{h \times w}$  una característica. El operador *global average pooling*,  $P_g : \mathbb{R}^{hw} \rightarrow \mathbb{R}$  se define como:

$$P_g(x) = \frac{1}{hw} \sum_{i=1}^h \sum_{j=1}^w x_{i,j}. \quad (2.6)$$

Más aún, sea  $x \in \mathbb{R}^{h \times w \times d}$  una característica. El operador *global average pooling*,  $P_g : \mathbb{R}^{hw} \rightarrow \mathbb{R}$  se define como:

$$P_g(x) = (P_g(x_1), \dots, P_g(x_d)). \quad (2.7)$$

### 2.1.5. Convolución como una operación lineal

Es posible ver una convolución como una multiplicación por una matriz poco densa, en donde varios elementos de la matriz están restringidos a ser iguales a otros. Para las convoluciones de una variable, tiene que ser una *matriz de toeplitz*. En lo que refiere a dos dimensiones, una convolución es equivalente a una *Doble Matriz Circulante por Bloques* [Definición 1.10]. Es decir, se tiene lo siguiente

**Corolario 2.10.** Sea  $I$  una imagen, y  $K$  un kernel. Existe una matriz  $\hat{K}$

$$I * K = \text{vec}(I) \hat{K}. \quad (2.8)$$

## 2.2. Redes Neuronales Convolucionales

Consideremos el problema de clasificación. Sea  $m$  la cantidad de clases posibles. Dados  $y_1, y_2, \dots, y_s \in \mathbb{R}^d$  y sus etiquetas  $c_1, c_2, \dots, c_s \in \mathbb{R}^m$ , en donde la  $k$ -ésima entrada de  $c_i$  puede ser 1 o 0. Cada vector  $c_i$  tiene únicamente una entrada con valor 1, que determina exactamente a que clase pertenece  $y_i$ .

Sean

$$Y_0 = [y_1, y_2, \dots, y_s]^T \quad \text{y} \quad C = [c_1, \dots, c_s]^T. \quad (2.9)$$

El objetivo es poder clasificar correctamente datos no etiquetados. Para ello se han desarrollado diferentes arquitecturas de redes neuronales, y en el 2015 las Redes Neuronales

Convolucionales obtuvieron el primer lugar en el concurso Image Net [18], consiguiendo resultados sin precedentes [19].

### 2.2.1. Descenso del Gradiente Estocástico

Ya que el problema de aprendizaje, es un problema de optimización, podemos recurrir a la teoría de optimización conocida. El algoritmo más común en el aprendizaje automático es el *descenso de gradiente estocástico* (SGD por sus siglas en inglés), el cuál es un algoritmo numérico para encontrar mínimos locales de una función. La intuición se basa en que si se tiene un punto  $x \in D_f$ , la dirección en dónde más decrece es  $-\nabla f$ . Éste método consiste en dos pasos

1. Se selecciona un punto inicial  $x_0$  en el dominio.
2. Se actualiza nuestro valor  $x_{i+1} = x_i - \alpha \nabla f(x_i)$  para  $i = 1, \dots, N$

donde  $N$  es el número de iteraciones totales y  $\alpha$  es un escalar, el cuál se conoce como razón de aprendizaje (learning rate). Es importante seleccionar una razón de aprendizaje apropiado, pues en caso de seleccionar una muy grande, el algoritmo podría diverger, por el contrario si se selecciona una muy pequeño, entonces el entrenamiento podría ser demasiado lento.

Para calcular el gradiente de manera exacta, es necesario utilizar todo el conjunto de datos, lo cuál puede traducirse a tiempos elevados de cómputo. Es por eso, que en la práctica, se utiliza únicamente un subconjunto aleatorio de datos, calculando así, una aproximación al gradiente  $\nabla J$  de la función de costo. Debido a este artificio que reduce el tiempo de entrenamiento, es que nuestro algoritmo es considerado estocástico.

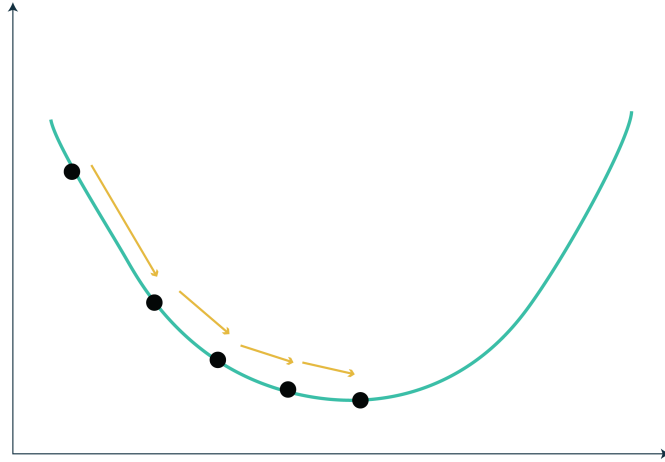


Figura 2.6: **Seleccionar su propia imagen**

### 2.2.2. Optimizadores

Existe una amplia literatura en el área de optimización de funciones. Por lo que es natural pensar que el SGD no es el único método para optimizar la función de costo. Debido a que la evaluación de dicha función, involucra millones de parámetros en las redes profundas actuales, no todos los algoritmos de optimización son adecuados para esta tarea. Aquí se presentan los métodos más efectivos investigados en los últimos años.

#### AdaGrad

A diferencia del SGD, el Algoritmo de Gradiente Adaptativo o AdaGrad [20], actualiza la razón de aprendizaje con cada iteración. El algoritmo tiene un mejor desempeño en presencia de gradientes dispersos, es decir cuando el vector del gradiente contiene varios ceros.

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{b_t} + \epsilon} g_{t-1} \quad (2.10)$$

dónde  $b_{t-1}$  es la suma del cuadrado de los gradientes.

$$b_t = \sum_{i=1}^t g_{t-1}^2 \quad (2.11)$$

### **RMSPProp**

El algoritmo RMSPProp fue propuesto por Geoff Hinton motivado en resolver el problema de las razones de aprendizaje sumamente pequeñas que se obtenían con el algoritmo AdaGrad y por consiguiente, evitar el desvanecimiento del gradiente.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)g_t^2 \quad (2.12)$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (2.13)$$

### **Adam**

El Estimador de momento Adaptativo (Adam por sus siglas en inglés) [21] se basa en momentos de primer y segundo orden del gradiente. Este método fue creado con la intención de obtener los beneficios de 2 optimizadores anteriores: el AdaGrad y el RMSPProp.

Para calcular los parámetros  $\theta_t$ , es necesario calcular el gradiente  $g_t = \nabla f(\theta_{t-1})$ . Denotamos  $m_t$  a la media móvil exponencial, y  $v_t$  al gradiente cuadrado. Los cuales se calculan de la siguiente manera:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t, \quad (2.14)$$

$$g_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2, \quad (2.15)$$

dónde  $\beta_1, \beta_2 \in [0, 1)$  son hiperparámetros que controlan el decrecimiento exponencial.

Los momentos  $\hat{m}_t$  y  $\hat{v}_t$  son la media y la varianza no centrada respectivamente.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad (2.16)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \quad (2.17)$$

El optimizador Adam, actualiza los parámetros de la siguiente manera:

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}. \quad (2.18)$$

Dónde  $\alpha$  es el tamaño de paso y  $\epsilon$  es un valor cercano a cero.

### 2.2.3. Propagación hacia atrás

Ahora que se tiene un algoritmo para optimizar una función de costo, la pregunta natural es ¿Cómo obtenemos el gradiente? Siendo la propagación hacia adelante de una red, una función tan complicada y con tantos parámetros, es muy difícil calcular de manera analítica nuestro gradiente. Por buena suerte para nosotros, existe una técnica conocida como *propagación hacia atrás*, encargada de calcular numéricamente el gradiente de la función de costo.

### 2.2.4. Normalización por Lotes

Entre las técnicas de regularización más utilizadas se encuentra la *normalización por lotes* (BN por sus siglas en inglés). En el 2015 en un artículo de google [22] se describe esta técnica y se constata que tiene múltiples beneficios tales como incrementar la razón de aprendizaje, conseguir que el entrenamiento sea más independiente de la inicialización y además actuar como regularizador.

**Definición 2.11.** Sea  $\mathcal{B} = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  un lote de características. La normalización por lotes de  $\mathcal{B}$  se define como:

$$B(x^{(i)}; \gamma, \beta) := \frac{\gamma(x^{(i)} - \mu)}{\sigma} + \beta, \quad i = 1, 2, \dots, m, \quad (2.19)$$

donde  $\mu := \sum_{i=1}^m x^{(i)} / m$  y  $\sigma^2 = \sum_{i=1}^m (x^{(i)} - \mu)^2 / m$ .

Los parámetros  $\gamma$  y  $\beta$  usualmente se aprenden en la optimización. Cuando se usa la normalización por lotes, no es necesario agregar **biases** pues son calculados explícitamente a través de  $\beta$ .



Ahora que conocemos lo que es una convolución, se define matemáticamente una Red Neuronal Convolutacional.

**Definición 2.12.** Sea  $x_0 \in \mathbb{R}^{h \times w \times d}$  una característica. La propagación hacia adelante de una CNN está determinada por el siguiente esquema recursivo

$$x_{n+1} = \sigma(x_n * K_n + b_n). \quad (2.20)$$

donde  $\sigma$  es una función de activación,  $K_n$  representa un Kernel y  $b_n \in \mathbb{R}^d$  es el *bias*.

En las primeras arquitecturas se usaban Redes Completamente Conectadas (FCN por sus siglas en inglés) tal como se aprecia en la Figura 2.7. Sin embargo en los últimos años se ha visto un incremento en el uso de Redes Completamente Convolucionales.

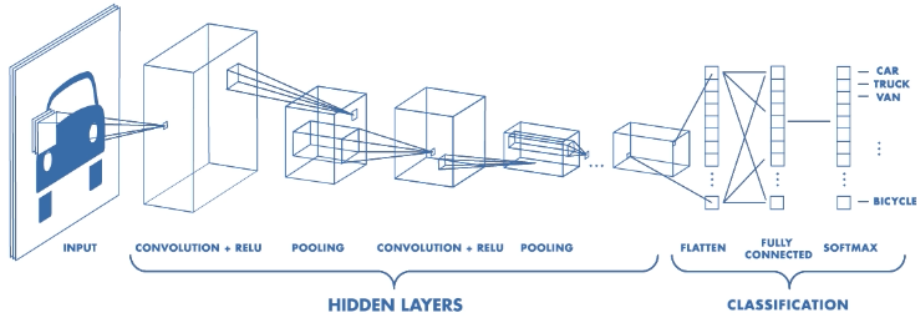


Figura 2.7: Ejemplo de arquitectura de una Red Neuronal Convolutacional. En la figura se aprecia la imagen de entrada, las capas convolucionales y la FCN que determina la clase [23]

### 2.2.5. Desvanecimiento del gradiente

Los mayoría de los algoritmos de optimización, dependen del gradiente. Y los algoritmos que utilizan la propagación hacia atrás, en ocasiones presentan el problema conocido como *desvanecimiento del gradiente*, que consiste en que los gradientes se vuelven tan pequeños que previenen al modelo de continuar con el proceso de aprendizaje.

### 2.2.6. CNNs en el Estado del Arte

En 2012 en el ImageNet ocurrió algo sin precedentes en dicha competencia. El ganador del concurso se había llevado el primer lugar absoluto, obteniendo una ventaja de 10.9 % de error por debajo del segundo lugar. Es así como la *AlexNet* [10] consiguió reputación para las CNNs. Para 2013, la red ganadora del ImageNet(2013) fue la Zf-net [24], la cuál era una modificación de los hiperparámetros de la AlexNet.

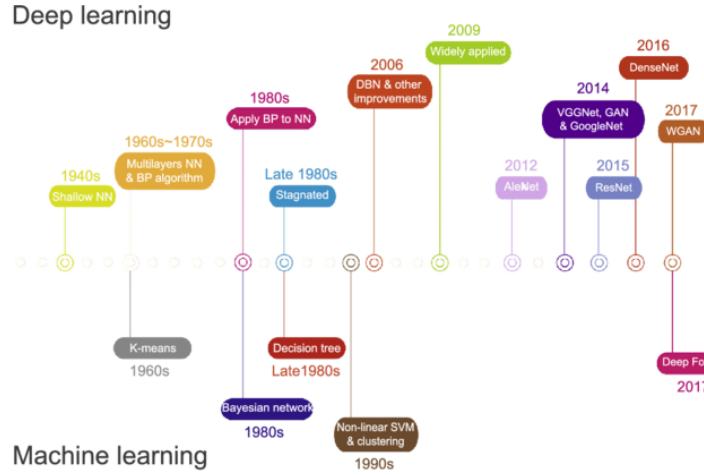


Figura 2.8: Línea de tiempo de las Redes en el Estado del Arte

## 2.3. ResNet

Las *Redes Neuronales Residuales* fueron presentadas en 2015 por Kaiming He et. al. en [19]. Uno de los mayores retos en el diseño de redes profundas, es el *desvanecimiento del gradiente*. Es decir, por la naturaleza de la propagación hacia atrás, las redes muy profundas, implican el cálculo de gradientes con entradas muy pequeñas, y eventualmente el gradiente se desvanece y previene el aprendizaje de la red. Es decir,  $\|\nabla_{\theta} L\| = 0$  dónde  $L$  es la función de pérdida y  $\theta$  son los parámetros.

Antes del 2015 existían dificultades para alcanzar redes con profundidades mayores a 20 capas. Para solucionar este problema, se crearon las *conexiones de salto*.

**Definición 2.13.** Consideremos el problema de clasificación (2.9). Sea  $x_0 \in \mathbb{R}^h \times w \times d$

una característica. La propagación hacia adelante de la ResNet está determinada por el siguiente esquema recursivo

$$x_{n+1} = x_n + \sigma(x_n * K_n + b_n). \quad (2.21)$$

donde  $\sigma$  es una función de activación,  $K_n$  representa un Kernel y  $b_n \in \mathbb{R}^d$  es el *bias*.

Ya que la operación de convolución puede ser expresada como una multiplicación de matrices, es posible reescribir la ecuación (2.21) como:

$$X_{n+1} = X_n + \sigma(X_n A_n + b_n). \quad (2.22)$$

dónde  $X_0 = \text{vec } x_0$  y  $A_n$  es la matriz que induce el kernel  $K_n$ .

La diferencia entre una CNN convencional es el término  $X_n$  que se agrega a  $\sigma(\cdot)$ , conocida como conexión de salto, cuyo propósito es estabilizar el gradiente, para evitar su explosión o desvanecimiento.

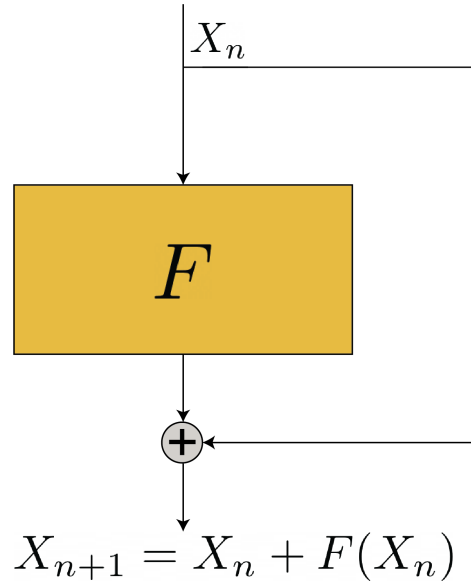


Figura 2.9: Diagrama de un bloque de ResNnet.

Pese a que la ecuación (2.22) es la forma canónica de definir una ResNet, también es posible usar otras funciones en vez de sólo la composición de la función de activación con

una función lineal. Por lo que una definición más general de la ResNet es

$$X_{n+1} = X_n + F_n(X_n) \quad (2.23)$$

En la ResNet original se observa que  $F_n(x) = \sigma(xA_n + b_n)$ . Sin embargo se han usado otras funciones. En [25] se toma

$$F_n(x) = \text{BN}(\hat{K}_n * \text{ReLU}(\text{BN}(K_n * x))) \quad (2.24)$$

que en la versión de multiplicación de matrices es:

$$F_n(x) = \text{BN}(\text{ReLU}(\text{BN}(x \cdot A_n)) \cdot B_n), \quad (2.25)$$

dónde  $A_n$  y  $B_n$  son las matrices correspondientes a los filtros  $K_n$  y  $\hat{K}_n$  de manera respectiva, BN y ReLU son las funciones de Normalización por lotes y Unión lineal Rectificada y la operación  $M \cdot N$  es la multiplicación de matrices convencional colocada para mayor claridad.

La función (2.24) es también la utilizada en la biblioteca de pyTorch [26] en el repositorio:

<https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py>.

Además de los bloques con conexiones de salto, las arquitecturas modernas cuentan con otras capas, incluyendo capas convolucionales, normalizaciones e incluso una Red Completamente Conectada (FCN por sus siglas en inglés) en la parte final de la propagación. Una de las ResNet más populares, es la ResNet-50

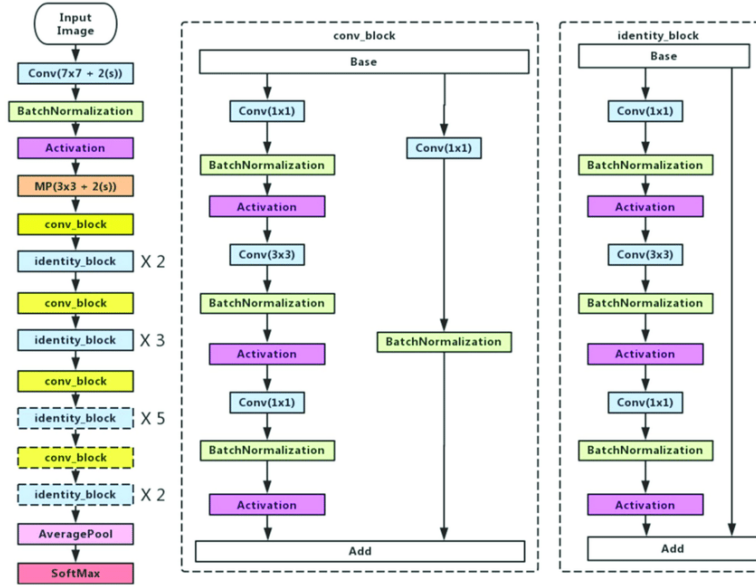


Figura 2.10: Diagrama de una ResNet-50

### 2.3.1. Inicialización de parámetros

Los algoritmos de optimización (Sección 2.2.2) requieren de un valor inicial  $\theta_0$ . En la mayoría de los casos, una selección adecuada de  $\theta_0$  puede conseguir que el número de iteraciones se reduzca en comparación con un  $\theta_0$  arbitrario. El Artículo [27] contiene un análisis dedicado a la inicialización de parámetros para aprendizaje profundo.

#### Inicialización con Ceros

Inicializar los parámetros con ceros es un acercamiento natural. Sin embargo, es conocido que si dos parámetros con la misma función de activación, están conectados con la misma entrada, entonces se modificarán de igual forma en el proceso de entrenamiento. De modo que se dice que la inicialización deber *romper la simetría* [12]. El caso de la inicialización con ceros, o con cualquier constante, no satisface esta propiedad, por lo que no es recomendable usarlas.

### Inicialización Aleatoria

Una manera sencilla de romper la simetría es inicializando los parámetros de manera aleatoria. Una heurística común es inicializar la matriz de pesos  $W$  con una distribución uniforme

$$W_{i,j} = U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right), \quad (2.26)$$

donde  $m$  es la cantidad de entradas, y  $U[-a, a]$  la distribución uniforme de  $-a$  hasta  $a$ . Sin embargo, al no prestar atención al comportamiento de los gradientes, esta forma de inicializar nuestros parámetros puede provocar desvanecimiento o explosión del gradiente.

### Inicialización de Glorot

Para *Inicialización de Glorot*, también conocida como la *inicialización de Xavier* [28] se asume que los pesos son inicializados independientemente y que las varianzas de las características de entrada son iguales. Además se pretende que las varianzas de los gradientes en todas las capas sean iguales. Para ello se propone la siguiente inicialización :

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \quad (2.27)$$

donde se tienen  $m$  valores de entrada y  $n$  valores de salida.

### Inicialización con Kaiming

La *inicialización de Kaiming* o *Inicialización de He* [29] asume funciones de activación del tipo Rectificador, tales como la ReLu o la PReLU.

$$W_l \sim \mathcal{N}\left(0, \frac{2}{n^l}\right) \quad (2.28)$$

dónde  $n_l$  es la cantidad de elementos en la capa  $l$ .

## 2.3.2. Estabilidad en las redes Neuronales

En clasificación de imágenes, una red debe ser robusta contra el ruido, o pequeñas modificaciones en la imagen. En el artículo [11] se establecen definiciones formales de

estabilidad, cuyo principio básico es que la salida debe ser continua con respecto a la entrada.

**Definición 2.14.** *Supóngase  $f$  la propagación hacia adelante de una red neuronal. Sea  $x$  una imagen y  $\hat{x}$  la misma imagen perturbada. Decimos que  $f$  es estable con respecto a epsilon cuando*

$$\|f(x) - f(\hat{x})\| < \epsilon \quad (2.29)$$

En la práctica se pretende reducir lo más posible el valor de  $\epsilon$ . En [30] se agrega un término  $h \in \mathbb{R}$  a la ResNet con la intención de añadirle estabilidad a la red

$$X_{n+1} = X_n + h\sigma(X_n A_n + b_n). \quad (2.30)$$

# Capítulo 3

## Ecuaciones diferenciales

Las ResNets y las Ecuaciones Diferenciales Ordinarias guardan cierta relación que será estudiada a detalle en el Capítulo 5. En este capítulo se presenta una introducción a las Ecuaciones Diferenciales Ordinarias (ODEs por sus siglas en inglés) y a los métodos numéricos necesarios para su resolución. Para un estudio más extensivo se puede consultar [31, 32].

Desde su implementación en el siglo XVII [33] las ecuaciones diferenciales han sido una herramienta de mucha utilidad para las ciencias, con aplicaciones en geometría [34], mecánica [35], electromagnetismo [36], etc.

Una ecuación diferencial ordinaria se representa de la siguiente manera:

$$\frac{dx}{dt} = f(x(t), t), \quad (3.1)$$

donde  $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ . En caso de que no exista ambigüedad, es posible escribir  $x$  en lugar de  $x(t)$  y  $x'$  para referirse a la derivada de  $x$  con respecto a  $t$ , con lo que la ecuación anterior se puede escribir de la siguiente manera:

$$x' = f(x, t) \quad (3.2)$$

En este caso, ya que la variable dependiente es el tiempo  $t$ , la derivada con respecto al tiempo se puede escribir con un punto, es decir:  $\dot{x} = \frac{dx}{dt}$ . En este trabajo, la variable dependiente será el tiempo, pero la teoría se puede extender para distintos dominios.



### 3.1. Problemas de valor inicial

Considérese el siguiente problema de valor inicial (IVP por sus siglas en inglés)

$$\dot{x}(t) = f(x, t), \quad t > 0 \quad (3.3)$$

$$x(t_0) = \nu, \quad (3.4)$$

donde  $t \in [t_0, t_f]$ .

#### 3.1.1. Existencia, unicidad y continuidad de soluciones

**Teorema 3.1.** *Sea  $f : [a_1, b_1] \times [a_2, b_2]$  una función Lipschitz continua en la segunda entrada y  $a_2 < x_a < b_2$ . Entonces existe un  $c \in [a_1, b_1]$  tal que el IVP*

$$\begin{cases} \dot{x} = f(x, t) \\ x(a_1) = x_a \\ t \in [a_1, c] \end{cases} \quad (3.5)$$

*tiene exactamente una solución  $x(t)$ . Aún más, si  $f$  es Lipschitz continua en  $[a_1, b_1] \times (-\infty, \infty)$ , entonces existe exactamente una solución en  $[a_1, b_1]$ .*

La demostración se puede encontrar en [31]

### 3.2. Estabilidad

En la literatura, no existe una convención al hablar de estabilidad de ecuaciones diferenciales [37, 38], sin embargo adoptaremos las definiciones de Ascher [39].

Cuando se hacen cambios mínimos en el valor inicial del IVP (3.3 - 3.4), uno se pregunta ¿qué ocurre con la solución? ¿Los cambios también serán pequeños? Para el análisis de estabilidad esto, se requieren algunas definiciones

**Definición 3.2.** *Una solución  $x(t)$  es llamada **estable** si para todo  $\varepsilon > 0$  existe un  $\delta > 0$  tal que cualquier otra solución  $\hat{x}(t)$  de 3.3 que satisfaga*

$$|x(t_0) - \hat{x}(t_0)| \leq \delta \quad (3.6)$$

también satisface

$$|x(t) - \hat{x}(t)| \leq \varepsilon, \quad \forall t > t_0. \quad (3.7)$$

La solución  $x(t)$  es **asintóticamente estable** si en adición a (3.7) se cumple que

$$|x(t) - \hat{x}(t)| \rightarrow 0, \quad \text{si } t \rightarrow \infty. \quad (3.8)$$

Por otro lado,  $x(t)$  es **relativamente estable** si en vez de (3.7) se satisface que

$$|x(t) - \hat{x}(t)| \leq \varepsilon |x(t)|, \quad \forall t > t_0. \quad (3.9)$$

**Definición 3.3.** Una solución es llamada **uniformemente estable** si, para cada  $\varepsilon > 0$ , existe  $\delta > 0$  tal que cualquier otra solución  $\hat{x}(t)$  de (3.3) que satisfaga

$$|x(c) - \hat{x}(c)| \leq \delta \quad (3.10)$$

para algún punto  $c \geq t_0$ , también se satisface que

$$|x(t) - \hat{x}(t)| \leq \varepsilon, \quad \forall t > c. \quad (3.11)$$

La estabilidad asintótica uniforme y la estabilidad relativa uniforme se definen de manera análoga. Es evidente que la estabilidad uniforme, implica estabilidad.

### 3.2.1. Problema Lineal

Considérese el IVP homogéneo

$$\dot{x} = A(t)x, \quad t > t_0 \quad (3.12)$$

$$x(t_0) = \alpha \quad (3.13)$$

**Teorema 3.4.** La solución y IVP (3.12- 3.13) con  $A$  constante es uniformemente estable si y sólo si todos los eigenvalores de  $A$  statisfacen que  $\text{Re}(\lambda) < 0$  o si  $\text{Re}(\lambda) = 0$  con  $\lambda$  simple.

Sin embargo,  $A(t)$  no tiene por qué ser constante. Para un caso más general, se tiene el siguiente teorema.

**Teorema 3.5.** *La solución y del IVP (3.12- 3.13) es uniformemente estable si se cumplen las siguientes dos condiciones*

1. La matriz  $A$  es **estrictamente diagonalmente dominante**, es decir

$$\sum_{j \neq i} |a_{i,j}| \leq (1 - \delta) |a_{ii}|, \quad i = 1, \dots, n \quad (3.14)$$

2. Los elementos de la diagonal son menores que 0

$$\operatorname{Re}(a_{ii}) < 0, \quad i = 1, \dots, n \quad (3.15)$$

Además, como consecuencia del Teorema de Gershgorin se tiene que si se satisfacen las hipótesis del Teorema (3.5) los eigenvalores cumplen que  $\operatorname{Re}(\lambda_i) < 0$ . De modo que los eigenvalores de nuestra matriz  $A$  son elementos suficientes para determinar la estabilidad de una ecuación diferencial.

### 3.2.2. IVP no lineales (El caso general)

Para determinar la estabilidad de un IVP no lineal, se puede hacer uso de los resultados obtenidos en la Subsección anterior. El tema de interés en esta ocasión es la ecuación

$$\dot{x} = f(x, t) \quad (3.16)$$

$$x(t_0) = \alpha \quad (3.17)$$

Sea  $\hat{x}$  una solución de (3.16), con  $\hat{x}(t_0)$  no muy lejano de  $x(t_0)$ . Expandiendo  $f(x, t)$  utilizando su serie de Taylor, se obtiene que

$$f(\hat{x}, t) = f(x, t) + J(x, t)(\hat{x} - x) + r(x, \hat{x}, t), \quad (3.18)$$

donde  $J$  es el jacobiano definido por

$$J(x, t) := \frac{\partial f}{\partial x}. \quad (3.19)$$

y el término  $r$  es el residuo. Si se ignora el residuo  $r$  en (3.18), y tomando  $z = x - \hat{x}$  se obtiene

$$\dot{z} = J(t)z \quad (3.20)$$

Por lo que para determinar la estabilidad, es suficiente conocer los eigenvalores del Jacobiano  $J$ .

### 3.3. Métodos de Runge-Kutta

Es posible generalizar el método de Euler, evaluando la derivada multiples veces en un paso. Esta idea se le atribuye a Runge [40]. y en 1901 Kutta hizo contribuciones importantes al método, dando paso a métodos de orden 4 y el primer método de orden 5, denominados métodos de Runge-Kutta. A continuación se describen alguno de los métodos de Runge-Kutta más relevantes.

#### 3.3.1. Método de Euler

La manera más elemental para resolver el IVP (3.3 - 3.4) es con el método de Euler.

Nótese que es posible particionar el intervalo  $[t_0, t_f]$  en  $N$  partes iguales, obteniendo así la sucesión

$$t_0, t_0 + h, t_0 + 2h, \dots, t_0 + Nh,$$

donde

$$h = \frac{t_f - t_0}{N}.$$

Se denotará  $t_n = t_0 + nh$  y  $x_n = x(t_n)$ . De modo que se tiene la siguiente sucesión:

$$t_0 < t_1 < t_2 < \dots < t_N = t_f. \quad (3.21)$$

El siguiente paso, es aproximar los valores de  $x_n$ , y a estas aproximaciones les llamaremos  $w_n$ , teniendo en cuenta que  $w_0 = x_0$ . Para ello, se hará uso de la expansión de Taylor de  $x(t + h)$ .

$$x(t + h) = x(t) + h\dot{x}(t) + R_1(t), \quad (3.22)$$

donde  $R_1(t)$  es el residuo de la expansión de Taylor. Sustituyendo  $\dot{x}(t) = f(t, x(t))$ , obtenemos

$$x(t+h) = x(t) + hf(t, x(t)) + R_1(t). \quad (3.23)$$

Sustituyendo  $t = t_n$  con  $i < N$  obtenemos que

$$x(t_n+h) = x(t_n) + hf(t_n, x(t_n)) + R_1(t_n). \quad (3.24)$$

Por lo tanto, substrayendo  $R_1(t)$ , es posible obtener una sucesión que aproxime a la solución  $x(t)$ :

$$w_{n+1} = w_n + hf(t_n, w_n), \quad w_0 = x_0. \quad (3.25)$$

### Error de truncamiento local

En general, el error de truncamiento local de un método numérico, es el error generado a partir de una iteración. Es decir:

$$e_n = |w_n - z(t_n)| \quad (3.26)$$

donde  $z$  es la solución del problema

$$\begin{cases} \dot{x} = f(x, t) \\ x(t_n) = w_n \\ t \in [t_n, t_{n+1}] \end{cases} \quad (3.27)$$

En el caso del método de Euler, es posible encontrar el error de truncamiento local. Asumiendo que en la identidad (3.22),  $x(t)$  es doblemente diferenciable en el intervalo  $(t_0, t_f)$ , ocurre que

$$R_1(t) = \frac{1}{2!} h^2 \dot{x}(\xi), \quad \xi \in (t, t+h). \quad (3.28)$$

### Error de truncamiento global

El error de truncamiento global de un método numérico, es el error generado por multiples iteraciones.

$$g_n = |w_n - x_n| \quad (3.29)$$

Para investigar el error global del método de Euler, asúmase una cota superior  $mh^2$  para la norma del error de truncamiento local, es decir  $e_n \leq mh^2$ . Además se asume Lipschitz continuidad para  $f$ , cuya constante de Lipschitz es  $L$ .

Es posible acotar el error global, de la siguiente manera:

$$g_n \leq \frac{Ch^k}{L}(e^{L(t_n-a)} - 1) \quad (3.30)$$

La demostración de (3.30) puede encontrarse en [31]. En general, se comparan los distintos métodos numéricos basándose en el error, y los mejores métodos son aquellos cuyo **orden** es mayor:

**Definición 3.6.** *Se dice que un método es de orden  $\rho$  cuando  $|w_n - x_n| = \mathcal{O}(h^\rho)$ .*

### 3.3.2. Método de Euler modificado

El siguiente método, es un ejemplo de un método de Runge-Kuta, el cuál se usará para desarrollar las ideas tras los métodos generales.

$$\begin{aligned} k_1 &= f(t_n, w_n), \\ k_2 &= f(t_n + \tfrac{1}{2}h, w_n + \tfrac{1}{2}hk_1), \\ w_{n+1} &= w_n + hk_2, \\ t_{n+1} &= t_n + h. \end{aligned}$$

Este es un método de segundo orden.

### 3.3.3. Método general

El método de Runge-Kutta general con  $s$  etapas se puede definir usando  $s^2 + 2s$  números

$$a_{i,j}, \quad i, j = 1, 2, \dots, s. \quad b_i, c_i, \quad i = 1, 2, \dots, s, \quad (3.31)$$

usando el siguiente esquema

$$w_{n+1} = w_n + h \sum_{i=1}^s b_i k_i. \quad (3.32)$$

En donde la sucesión  $\{k_i\}$  es calculada usando la función  $f$ :

$$k_i = f\left(t_n + c_i h, w_n + h \sum_{i=1}^s a_{i,j} k_i\right), \quad i = 1, 2, \dots, s. \quad (3.33)$$

Es posible determinar si el método es explícito o implícito, basándose en la matriz  $A = \{a_{i,j}\}$ . En caso de que sea una matriz triangular inferior, con todos los elementos de la diagonal iguales a cero ( $a_{i,j} = 0$  para  $j = i, i+1, \dots, s$ ) el esquema es explícito.

Las siguientes relaciones, son conocidas como condiciones de orden

$$\sum_{i=1}^s b_i = 1, \quad \sum_{i,j=1}^s b_i a_{i,j} = \frac{1}{2}, \quad \sum_{i,j,k=1}^s b_i a_{i,j} a_{j,k} = \frac{1}{6}, \quad \sum_{i,j,k=1}^s b_i a_{i,j} a_{j,k} = \frac{1}{3} \quad (3.34)$$

y aseguran esquemas de al menos orden 3.

### 3.3.4. Runge Kutta de orden 4 (RK4)

Uno de los métodos más conocidos es el de orden 4

$$w_{n+1} = w_n + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4), \quad (3.35)$$

donde

$$\begin{aligned} s_1 &= f(t_n, w_n) \\ s_2 &= f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}s_2\right) \\ s_4 &= f(t_n + h, w_n + hs_3). \end{aligned}$$

La simplicidad de este método, hace que sea muy fácil de programar, y al ser de orden 4, se prefiere por sobre otros métodos como Euler, o el Trapezoide. Como se puede apreciar, RK4 es un método de orden 4 y además tiene 4 etapas (i.e.  $s = 4$ ). Sin embargo, no se conoce cuántas etapas son necesarias para esquemas de órdenes mayores a 8.

### 3.3.5. Métodos de Runge Kutta simplécticos

Dentro de la categoría de métodos de Runge Kutta, existen los métodos simplécticos [41]. Se caracterizan por preservar *invariantes cuadráticas* [42, 43].

**Definición 3.7.** *Un esquema de Runge Kutta se dice simpléctico si satisface la relación*

$$b_i a_{i,j} + b_j a_{j,i} - b_i b_j = 0, \quad i, j = 1, \dots, s. \quad (3.36)$$

### 3.3.6. Métodos implícitos

Los métodos numéricos analizados hasta este momento se conocen como métodos explícitos, debido a que es posible calcular  $w_{n+1}$  conociendo el valor  $w_n$ .

#### Euler hacia atrás

Una versión implícita de Euler se conoce como Euler hacia atrás.

$$w_0 = x_0 \quad (3.37)$$

$$w_{n+1} = w_n + hf(t_{n+1}, w_{n+1}) \quad (3.38)$$

Difiere del método de Euler tradicional en que la pendiente que se usa involucra  $w_{n+1}$ . De modo que en cada nueva iteración, es necesario resolver un sistema de ecuaciones, y por consiguiente el costo computacional se incrementa.





# Capítulo 4

## Teoría de control óptimo

Las ResNets guardan cierta relación con las Ecuaciones Diferenciales [30]. En el Capítulo 5 se estudia la conexión entre las DNNs y las ODEs. Debido a esto y a que el problema de aprendizaje implica optimizar una función, el problema es equivalente a encontrar un *control óptimo*

A continuación, se desarrollarán algunos resultados de Teoría de control óptimo con base en [44].

Cuando se habla de sistemas, se hace referencia a un proceso que cambia de estado conforme pasa el tiempo. En este caso considérese un sistema dinámico definido en el intervalo  $[0, T]$  como

$$\dot{x} = f(x(t), t), \quad x(0) = x_0, \quad (4.1)$$

donde  $x(t)$ , conocida como la *variable de estado* puede representar la producción de alimento en un tiempo  $t$ , el dinero recaudado en un tiempo  $t$ , o en general el estado de un proceso en un tiempo  $t$ . En logística y física, existen variables que se pueden controlar, como la cantidad de publicidad, o la reinversión de capital cada cierto momento, con lo que es posible extender el sistema (4.1) al siguiente:

$$\dot{x} = f(x(t), u(t), t), \quad x(0) = x_0, \quad (4.2)$$

donde la variable  $u(t)$  es denominada *variable de control*. Conociendo la variable de control, es posible determinar la solución para  $x$  en el sistema (4.2). El problema de control óptimo

reside en maximizar la *función objetivo*, definida como

$$J = \int_0^T F(x(t), u(t), t) dt + S[x(T), T]. \quad (4.3)$$

La función  $S$  determina el *valor de rescate* en el estado final  $x(T)$  en el tiempo  $T$ , y obtiene pues en el área de economía es *el valor de un activo, una vez que ha concluido su vida útil* [45]. Se denotará al conjunto de valores posibles de  $u(t)$  como  $\Omega(t)$ .

**Definición 4.1.** *El problema de control óptimo es encontrar un valor admisible  $u^*$  tal que*

$$u^* = \max_{u(t) \in \Omega(t)} \left\{ \int_0^T F(x, u, t) dt + S(x(T), T) \right\} \quad (4.4)$$

*con la condición*

$$\dot{x} = f(x, u, t), \quad x(0) = 0. \quad (4.5)$$

*La trayectoria óptima denotada como  $x^*$  es la trayectoria que se obtiene cuando  $u = u^*$ .*

## 4.1. La ecuación de Hamilton-Jacobi-Bellman

Para estudiar la Ecuación de Hamilton-Jacobi-Bellman, es necesario el principio de optimalidad, el cuál establece que para obtener un camino óptimo de 0 a  $T$ , es necesario que para toda  $t$ , el camino de  $t$  a  $T$  sea óptimo también. Es importante remarcar que la siguiente prueba se encuentra en [44] y se especifica que no tiene el propósito de ser rigurosa.

Sea  $V : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$  una función definida por

$$V(x, t) = \max_{u(s) \in \Omega(s)} \left\{ \int_t^T F(x(s), u(s), s) ds + S(x(T), T) \right\} \quad (4.6)$$

donde  $s \geq t$ . Sea  $\delta t$  un diminuto incremento en el tiempo, nótese que

$$\begin{aligned} V(x, t) - V(x(t + \delta t), t + \delta t) &= \max_{u(s) \in \Omega(s)} \left\{ \int_t^T F(x(s), u(s), s) ds + S(x(T), T) \right\} \\ &\quad - \max_{u(s) \in \Omega(s)} \left\{ \int_{t+\delta}^T F(x(s), u(s), s) ds + S(x(T), T) \right\} \end{aligned}$$

Lo cuál, gracias al principio de optimalidad [46] se tiene que

$$V(x(t), t) - V(x(t + \delta t), t + \delta t) = \max_{u(\tau) \in \Omega(\tau)} \left\{ \int_t^{t+\delta t} F(x(\tau), u(\tau), \tau) \right\}, \quad (4.7)$$

y por consiguiente

$$V(x(t), t) = \max_{u(\tau) \in \Omega(\tau)} \left\{ \int_t^{t+\delta t} F(x(\tau), u(\tau), \tau) + V(x(t + \delta t), t + \delta t) \right\}. \quad (4.8)$$

Debido a que  $F$  es una función continua, la integral en (4.8) es igual a  $F(x, u, t)\delta t$ . Por consiguiente:

$$V(x, t) = \max_{u \in \Omega(t)} F(x, u, t)\delta t + V[x(t + \delta t), t + \delta t] + \mathcal{O}(\delta t) \quad (4.9)$$

Asumiendo que la función  $V$  es continuamente diferenciable en ambas entradas es posible hacer uso de la expansión de Taylor de  $V$  en  $\delta t$ :

$$V[x(t + \delta t), t + \delta t] = V(x, t) + [V_x(x, t)\dot{x} + V_t(x, t)]\delta t + \mathcal{O}(\delta t) \quad (4.10)$$

Sustituyendo (4.2) en (4.9) se obtiene que

$$V(x, t) = \max_{u \in \Omega(t)} \{F(x, u, t)\delta t + V(x, t) + V_x(x, t)f(x, u, t)\delta t + V_t(x, t)\delta t\} + \mathcal{O}(\delta t) \quad (4.11)$$

Cancelando  $V(x, t)$  de ambos lados y dividiendo entre  $\delta t$  se deduce que

$$0 = \max_{u \in \Omega(t)} \{F(x, u, t) + V_x(x, t)f(x, u, t) + V_t(x, t)\} + \frac{\mathcal{O}(\delta t)}{\delta t}, \quad (4.12)$$

con lo que si  $\delta t \rightarrow 0$  se tiene que

$$0 = \max_{u \in \Omega(t)} \{F(x, u, t) + V_x(x, t)f(x, u, t) + V_t(x, t)\} \quad (4.13)$$

dónde si se evalúa  $t = T$  en  $V$  se obtiene que

$$V(x, T) = S(x, T). \quad (4.14)$$

Cuando la función  $V$  es aplicada en el estado óptimo  $x^*$  se obtiene un nuevo vector

**Definición 4.2.** (*Vector de retorno marginal*) Sea  $V$  la función definida en (4.6), el vector de retorno marginal  $\lambda(t)$  se define como

$$\lambda(t) = V_x(x^*(t), t) := V_x(x, t)|_{x=x^*} \quad (4.15)$$

Lo cuál será útil para definir el concepto de Hamiltoniano.

**Definición 4.3.** *Considérese el sistema (4.4)-(4.5). Se denota el hamiltoniano  $H : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$  del sistema como*

$$H(x, u, \lambda, t) = F(x, u, t) + \lambda f(x, u, t). \quad (4.16)$$

La ecuación (4.13) puede reescribirse como

$$\max_{u \in \Omega(t)} [H(x, u, V_x, t) + V_t] = 0 \quad (4.17)$$

la cuál se conoce como *Ecuación de Hamilton-Jacobi-Bellman*(HJB). Más aún, ya que  $V_t$  no depende de  $u$  es posible retirar  $V_t$  del operador máx.

$$\max_{u \in \Omega(t)} [H(x, u, V_x, t)] + V_t = 0. \quad (4.18)$$

Gracias a la definición del control óptimo y de  $\lambda(t)$ , el operador  $u^*$  maximiza la ecuación (4.18), por tanto sea  $u \in \Omega(t)$ , se tiene que

$$H[x^*(t), u^*(t), \lambda(t), t] + V_t(x^*(t), t) \geq H[x^*(t), u(t), \lambda(t), t] + V_t(x^*(t), t) \quad (4.19)$$

$$\Rightarrow H[x^*(t), u^*(t), \lambda(t), t] \geq H[x^*(t), u(t), \lambda(t), t]. \quad (4.20)$$

## 4.2. Ecuación adjunta

Nótese que para encontrar el control óptimo, la ecuación (4.18) se maximiza con  $x = x^*$  y  $u = u^*$ . Consideremos entonces pequeñas perturbaciones en el camino óptimo. Sea  $x(t) = x^*(t) + \delta x(t)$  y sea  $t \in [0, T]$ , podemos escribir (4.18) como:

$$0 = H[x^*(t), u^*(t), V_x(x^*(t), t), t] + V_x(x^*(t), t) \quad (4.21)$$

$$\geq H[x(t), u^*(t), V_x(x(t), t), t] + V_x(x(t), t). \quad (4.22)$$

Lo cuál significa que  $x^*(t)$  es un máximo local, de modo que la derivada con respecto a  $x$ , debe ser 0.

$$H_x[x^*(t), u^*(t), V_x(x^*(t), t), t] + V_{tx}(x^*(t), t) = 0, \quad (4.23)$$

Asúmase además que  $V$  es doblemente diferenciable en todos sus argumentos. Por otro lado, con  $H = F + V_x f$ , podemos derivar con respecto de  $x$

$$H = F + V_x f \quad (4.24)$$

$$\Rightarrow H_x = \frac{\partial(F + V_x f)}{\partial x} \quad (4.25)$$

$$\Rightarrow H_x = F_x + \frac{\partial(V_x f)}{\partial x} \quad (4.26)$$

$$\Rightarrow H_x = F_x + V_x F_x + f^T V_{xx} \quad (4.27)$$

$$\Rightarrow H_x = F_x + V_x F_x + (V_{xx} f)^T \quad (4.28)$$

Nótese que de (4.26) a (4.27) se hizo uso de la derivada de un producto y para (4.28) que  $V_{xx}$  es una matriz simétrica. Sustituyendo esto en (4.23) obtenemos:

$$H_x = F_x + V_x F_x + (V_{xx} f)^T + V_{tx} = 0. \quad (4.29)$$

Tómese ahora la derivada de  $V_x$  con respecto al tiempo

$$\frac{dV_x}{dt} = \left( \frac{dV_{x_1}}{dt}, \frac{dV_{x_2}}{dt}, \dots, \frac{dV_{x_n}}{dt} \right) \quad (4.30)$$

$$= (V_{x_1 x} \dot{x} + V_{x_1 t}, V_{x_2 x} \dot{x} + V_{x_2 t}, \dots, V_{x_n x} \dot{x} + V_{x_n t}) \quad (4.31)$$

$$= \left( \sum_{i=1}^n V_{x_1 x_i} \dot{x}_i, \dots, \sum_{i=1}^n V_{x_n x_i} \dot{x}_i \right) + (V_x)_t \quad (4.32)$$

$$= (V_{xx} \dot{x})^T + V_{xt} \quad (4.33)$$

$$= (V_{xx} f)^T + V_{tx} \quad (4.34)$$

Combinando (4.29) con (4.34) se obtiene que

$$F_x + V_x F_x + \frac{dV_x}{dt} = 0 \quad (4.35)$$

y ya que  $\lambda$  fue definida como  $V_x$ , se tiene lo siguiente:

$$\dot{\lambda} = -F_x - \lambda f_x. \quad (4.36)$$

Sin embargo, el lado derecho de esta ecuación, es igual a  $H_x$  debido a la definición del hamiltoniano  $H$ . Con lo que se puede reescribir como

$$\dot{\lambda} = -H_x. \quad (4.37)$$



# Capítulo 5

## Relación entre las ecuaciones diferenciales y las ResNets

Después de la aparición de la ResNet, los esfuerzos por mejorar el desempeño de las redes dieron lugar al uso de Ecuaciones Diferenciales Ordinarias (ODEs por sus siglas en inglés). Las Ecuaciones Diferenciales Ordinarias (ODEs) se encuentran en más de una forma en la literatura de Redes Neuronales [47, 48]. En 2019 Xinshi Chen realizó una revisión de los artículos relacionados con esto [49].

El uso de las ODEs en el aprendizaje profundo tiene dos líneas de investigación principales:

1. El proceso de entrenamiento de una red como un problema de control óptimo.
2. Arquitecturas diseñadas con base en métodos numéricos de Ecuaciones Diferenciales.

Se describen brevemente, ambas líneas dando mayor importancia al segundo punto.

### 5.1. Control Óptimo para DL

El problema de clasificación, así como otros problemas de ML implican aprender una función  $g : \mathbb{R}^d \rightarrow \mathbb{R}$ . En este paradigma, es posible considerar el mapeo de  $x$  a  $g(x)$



como una evolución desde el estado inicial  $X(0) = x$  hasta el estado final  $X(T)$ , dónde la dinámica puede ser modelada por una ecuación diferencial  $\dot{X}(t) = f(X(t), t)$ .

Por lo que es posible definir el problema de aprendizaje supervisado cómo un problema de control óptimo [50, 51].

$$\min_{\theta} C(X(T)) + \int_0^T R(\theta(t)) dt \quad (5.1)$$

$$\text{Sujeto a } \dot{X}(t) = f(X(t), \theta(t), t), \quad X(0) = x_0, t \in [0, T]. \quad (5.2)$$

donde  $R$  es el regularizador, y el control  $\theta(t)$  los parámetros de la red. Resolver este problema, implica hallar un control óptimo  $\theta^*$ . Para ello, es posible usar la teoría desarrollada alrededor de la ecuación de HJB, y el Principio del Máximo de Pontryagin (PMP). Para resolver el PMP existen muchos métodos numéricos, sin embargo, la mayoría no son escalables, esto se explica en [52], con lo que se propone el Método de Aproximaciones Sucesivas (MSA por su siglas en inglés). En [53] se pretende investigar la versión discreta de este problema.

## 5.2. DNNs como discretización de ODEs

La propagación hacia adelante de una ResNet fue definida en la Sección (2.3) por la siguiente ecuación

$$X_{n+1} = X_n + F(X_n). \quad (5.3)$$

En algunas investigaciones [47, 48, 54] se establece una conexión entre la ecuación (5.3) y la forma discreta de una ecuación diferencial. La versión discreta es

$$X_{n+1} - X_n = F(X_n), \quad n = 1, 2, \dots, N, \quad (5.4)$$

siendo la versión continua

$$\dot{X}(t) = F(X(t), t), \quad t \in [0, T], \quad (5.5)$$

dónde  $F$  puede ser la composición de activación, con convoluciones y normalizaciones. Un ejemplo es:

$$F(X(t), t) = \sigma(X(t)A(t) + b(t)). \quad (5.6)$$

La discretización se vuelve más evidente cuando se agrega el factor  $h$  de estabilidad, en la ecuación (5.4)

$$\frac{X_{n+1} - X_n}{h} = F(X_n), \quad n = 1, 2, \dots, N. \quad (5.7)$$

Nótese que reescribiendo (5.7) se tiene el método de Euler descrito en la Sección 3.3.1 para resolver el problema de valor inicial (5.5) con  $X(0) = X_0$ .

$$X_{n+1} = X_n + hF(X_n, t_i). \quad (5.8)$$

Analizando la ResNet como una versión discreta de una Ecuación Diferencial, es posible hacer uso de la teoría de ecuaciones diferenciales desarrollada en el Capítulo 3.

### 5.2.1. Estabilidad de Redes por medio de ODEs

En la sección 2.3.2 hubo una pequeña introducción al concepto de estabilidad para las redes Neuronales. Sin embargo, ahora que se ha construido una conexión entre las redes y las ODEs, se puede hacer uso de los teoremas de la sección 3.2 para estabilidad de ODEs.

En [30] se hace un estudio sobre la estabilidad de las ResNets utilizando ecuaciones diferenciales. Gracias a la Sección 3.2.2 se conoce la condición suficiente para la estabilidad del IVP (5.5) y es que  $Re(\lambda) < 0$  para cada eigenvalor del jacobiano  $J$  de  $F$ . De modo que puede reescribirse como

$$\max_{n=1,2,\dots,L} Re[\lambda_i(J(t))] \leq 0, \quad \forall t \in [0, T], \quad (5.9)$$

donde  $\lambda_i(J(t))$  representa el  $i$ -ésimo eigenvalor del jacobiano de  $F$ . Tomando como  $F = (F_1, F_2, \dots, F_d)$  de la ecuación (5.6) se tiene que

$$F(y) = \sigma(Ay + b) = \begin{pmatrix} \sigma(a_{11}y_1 + \dots + a_{1d}y_d + b) \\ \vdots \\ \sigma(a_{d1}y_1 + \dots + a_{dd}y_d + b) \end{pmatrix} \quad (5.10)$$

De modo que

$$F_j(y) = \sigma(a_{j1}y_1 + a_{j2}y_2 + \dots + a_{jd}y_d + b) \quad (5.11)$$

y por consiguiente, el jacobiano de  $F$  queda determinado por las derivadas parciales:

$$\frac{\partial F_j}{\partial y_k} = \frac{\sigma(a_{j1}y_1 + a_{j2}y_2 + \dots + a_{jd}y_d + b)}{\partial y_k} = \sigma'(A_{j*} \cdot y + b)a_{jk}, \quad (5.12)$$

donde  $A_{j*}$  representa el vector  $[A_{j1}, A_{j2}, \dots, A_{jd}]^T$ . Por lo tanto, es posible escribir el Jacobiano de  $F$  de la siguiente manera:

$$J(t) = \begin{pmatrix} \sigma'(A_{1*} \cdot y + b)a_{11} & \sigma'(A_{1*} \cdot y + b)a_{12} & \dots & \sigma'(A_{1*} \cdot y + b)a_{1d} \\ \sigma'(A_{2*} \cdot y + b)a_{21} & \sigma'(A_{2*} \cdot y + b)a_{22} & \dots & \sigma'(A_{2*} \cdot y + b)a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma'(A_{d*} \cdot y + b)a_{d1} & \sigma'(A_{d*} \cdot y + b)a_{d2} & \dots & \sigma'(A_{d*} \cdot y + b)a_{dd} \end{pmatrix} \quad (5.13)$$

Debido al modo en que las matrices diagonales se multiplican con otras matrices, la ecuación anterior se puede reescribir como

$$J = \text{diag}(\sigma'(A_{1*} \cdot y), \dots, \sigma'(A_{N*} \cdot y))A \quad (5.14)$$

$$= \text{diag}(\sigma'(Ay + b))A. \quad (5.15)$$

Debido a que las funciones de activación son casi siempre no decrecientes, se asume que  $\sigma'(\cdot)$  es una función positiva, es decir  $\sigma(x) \geq 0$  para todo  $x$ . De modo que en [30] se propone que la ecuación (5.9) se satisface cuando

$$\max_{n=1,2,\dots,N} \text{Re}[\lambda_i(A(t))] \leq 0, \quad \forall t \in [0, T]. \quad (5.16)$$

Así como la ResNet se equipara al método de Euler, existen otras redes que son equivalentes a distintos métodos numéricos para la resolución de ODEs, algunas que incluso no fueron diseñadas con el propósito de replicar estos esquemas. A continuación se discuten algunas de estas arquitecturas.

### 5.2.2. PolyNet

En 2017 se propuso la PolyNet [55] por Zhang et al. Se implementó un módulo de *polincepción* que contiene polinomios con la operación de composición

$$X_{n+1} = X_n + F(X_n) + F(F(X_n)) = (I + F + F^2)(X_n) \quad (5.17)$$

En el artículo [47] se muestra la relación de algunas redes con las ecuaciones diferenciales. Particularmente, la PolyNet que inicialmente no está inspirada en una ODE, resulta ser de forma aproximada una discretización del método de Euler hacia atrás (Backward Euler). Veremos que la siguiente igualdad se cumple:

$$(I - hF)^{-1} = I + hF + (hF)^2 + \dots + (hF)^n + \dots \quad (5.18)$$

Para ello, es necesario ver que la composición de  $(I - hF)$  con la suma infinita de (5.18) es la identidad.

$$\begin{aligned} (I - hF)(I + hF + (hF)^2 + \dots) &= (I + hF + (hF)^2 + \dots + (hF)^n + \dots) \\ &\quad - hF(I + hF + (hF)^2 + \dots + (hF)^n + \dots) \\ &= I + [hF - hF] + [(hF)^2 - (hF)^2] + \dots \\ &= I. \end{aligned}$$

Consecuentemente

$$\begin{aligned} u_{n+1} &= (I + hF + (hF)^2 + \dots + (hF)^n + \dots)u_n \\ &= (I - hF)^{-1}u_n, \end{aligned}$$

Aplicando la función  $I - hF$  en ambos lados de la ecuación se tiene que

$$u_{n+1} - hF(u_{n+1}) = u_n, \quad (5.19)$$

y por consiguiente

$$u_{n+1} = u_n + hF(u_{n+1}). \quad (5.20)$$

Nótese que esta ecuación es análoga a la ecuación (3.38) del método de Euler hacia atrás.

### 5.2.3. FractalNet

Así como la PolyNet, la FractalNet [56] propuesta en 2016, no fue inspirada inicialmente por un método numérico para resolver ODEs, pero de igual forma es posible encontrar una conexión.

Sea  $c$  el índice del fractal truncado  $f_c(\cdot)$ . Considérese el caso base:

$$f_1(z) = \text{conv}(z) \quad (5.21)$$

En [56] define el fractal de manera recursiva

$$f_{c+1}(z) = [(f_c \circ f_c)(z)] \oplus [\text{conv}(z)], \quad (5.22)$$

donde  $\oplus$  es el operador de unión, y puede representar concatenación, adición, entre otras.

Traduciendo la ecuación (5.22) a la notación de convoluciones y tomando  $a \oplus b = \frac{a+b}{2}$  queda como

$$f_{c+1} = \frac{1}{2}k_c * z + \frac{1}{2}f_c(f_c(z)). \quad (5.23)$$

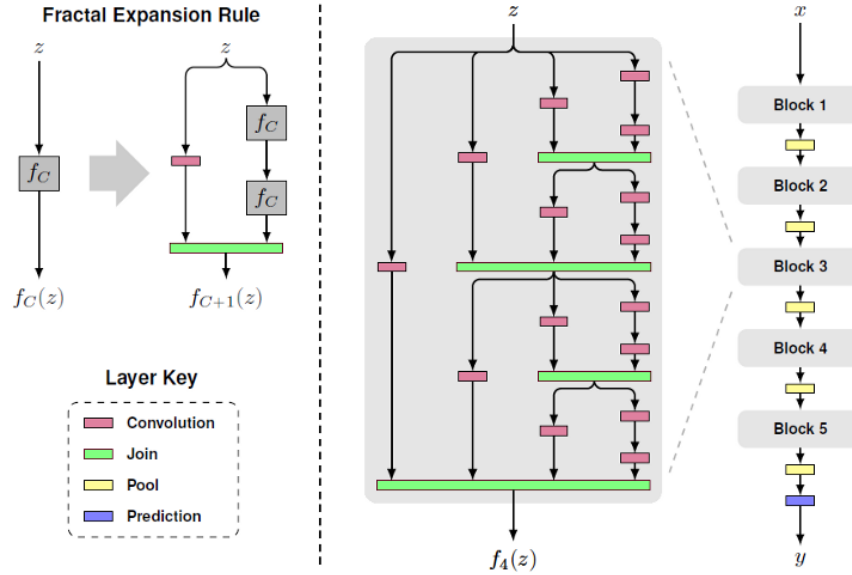


Figura 5.1: Representación gráfica de un bloque de la FractalNet extraída del artículo original [56].

La segunda iteración se obtiene de la siguiente manera:

$$f_2(z) = \frac{1}{2}k_1 * z + \frac{1}{2}f_1(f_1(z)), \quad (5.24)$$

de modo que en [47] establecen una relación con el método de Runge Kutta de segundo orden.

### 5.2.4. Midpoint Network

La Midpoint Network fue propuesta por Chang et al en 2018 junto con otras redes reversibles [57]. El concepto de reversibilidad en las redes puede verse en [58], y su propósito es no tener que almacenar las activaciones en la memoria, debido a que estas pueden ser reconstruidas con la salida.

$$\frac{X_{n+1} - X_{n-1}}{2h} = F(X_n) \quad (5.25)$$

Lo cuál da lugar a la siguiente propagación hacia adelante

$$X_{n+1} = X_{n-1} + 2hF(X_n) \quad (5.26)$$

Se puede observar la reversibilidad algebraicamente, asumiendo que se tienen los dos últimos estados  $X_L$  y  $X_{L-1}$ ,

$$X_{L-2} = X_L - 2hF(X_{L-1}). \quad (5.27)$$

Por otro lado, en [57] se modifica la función (5.6) para satisfacer el criterio (5.16), de modo que se propone la función

$$F(X) = \sigma((A - A^T)X + b) \quad (5.28)$$

debido a que todos los eigenvalores de  $A - A^T$  son imaginarios.

$$X_{n+1} = \begin{cases} 2h\sigma((A_n - A_n^T)X_n + b_n), & j = 0 \\ X_{i-1} + 2h\sigma((A_n - A_n^T)X_n + b_n), & j > 0. \end{cases} \quad (5.29)$$

### 5.2.5. IMEXnet

Así como el problema de Valor Inicial puede ser estable, es importante que el método numérico seleccionado también goce de estabilidad. Los métodos explícitos destacan por su simplicidad, y sobre todo por su costo computacional reducido. Sin embargo, cuando se habla de estabilidad, los métodos implícitos suelen mostrar superioridad [59]. Para aprovechar los beneficios de ambos tipos de métodos, se creó el método *Implícito-Explícito* (IMEX) en [60]. En su versión más simple, consiste en que si se tiene una ecuación de la siguiente forma:

$$\dot{X} = f(X, t) + g(X, t), \quad X(0) = X_0, \quad (5.30)$$

se aproxima la solución con el uso del Euler hacia adelante para  $f$  y Euler hacia atrás para  $g$

$$w_{n+1} = w_n + hf(w_n, t_n) + hf(w_{n+1}, t_{n+1}), \quad w_0 = y_0. \quad (5.31)$$

es posible variar los métodos implícitos e implícitos utilizados y convendrá usarlos dependiendo de la forma específica de  $g$ .

Motivados por este método, surge una red llamada *IMEXnet* [61] propuesta por Haber y Ruthotto con el propósito de crear una red estable. Es posible reescribir la ecuación (5.5)

$$\dot{X}(t) = F(X(t), t) + MX(t) - MX(t) \quad (5.32)$$

donde se puede seleccionar la matriz  $M$  cómo una matriz simétrica y definida positiva. El sumando  $F(X(t), t) + MX(t)$  es el término explícito y  $-MX(t)$  el término implícito. De modo que la discretización con el método IMEX es como sigue:

$$X_{n+1} = X_i + h[F(X_n) + MX_n] - hMX_{n+1} \quad (5.33)$$

$$\Rightarrow X_{n+1} + hMX_{n+1} = X_n + hF(X_n) + hMX_n \quad (5.34)$$

$$\Rightarrow (I + hM)X_{n+1} = X_n + hF(X_n) + hMX_n \quad (5.35)$$

$$(5.36)$$

Con lo que es la propagación hacia adelante de la *IMEXnet* queda determinada por:

$$X_{n+1} = (I + hM)^{-1}(X_n + hF(X_n) + hMX_n). \quad (5.37)$$

En el artículo original [61] se discuten sus ventajas y desventajas.

### 5.2.6. Otros Avances

En este capítulo se vio que se puede hacer un análisis de estabilidad de las redes [30,62], gracias sus versiones continuas como IVPs. Además analizó una variedad de redes que guardan relación con métodos numéricos para la solución de ecuaciones diferenciales basadas en el sistema dinámico (5.5).

Entre las redes reversibles propuestas en [57] también se propuso una red basada en una ecuación de segundo orden

$$\ddot{X}(t) = -A(t)^T \sigma(A(t)X(t) + b(t)). \quad (5.38)$$

En [63] se proponen redes basadas en Ecuaciones Diferenciales Parciales, basadas en el IVP

$$\partial_t X(\theta, t) = F(\theta(t), X(t)), \quad t \in (0, T] \quad (5.39)$$

$$X(\theta, 0) = X_0 \quad (5.40)$$

Se consideran redes basadas en métodos numéricos de múltiples pasos [47], y también basadas en sistemas hamiltonianos [30, 57].





# Capítulo 6

## Experimentos y Resultados

### 6.1. Descripción del problema

Para la fase experimental se utilizaron dos bases de datos. Como punto comparativo al actual estado del arte, se utilizó una base de datos provista por el *Pollen Challenge* [64]. El objetivo del concurso era clasificar los granos de polen utilizando un conjunto de imágenes bajo microscopio.

Las imágenes de microscopio fueron digitalizadas y clasificadas por expertos aerobiólogos en 4 clases, las cuales incluyen 3 especies de polen y una clase extra que podría ser confundida con polen (burbujas, aire, etc).



Figure 1 – Example of class 1 (Normal Pollen)



Figure 2 – Example of class 2 (Anomalous Pollen)



Figure 3 – Example of class 3 (Alnus)



Figure 4 – Example of class 4 (Debris)

Figura 6.1: Imagen extraída de la página oficial del Pollen Challenge [64]

## 6.2. Métricas para clasificación Binaria

Analizaremos primero el caso de las métricas en un problema de clasificación binario. Es decir, dada una clase  $\mathcal{C}$ , es posible que un dato  $y$  pertenezca o no pertenezca a  $\mathcal{C}$ .

Es importante describir primero las métricas en este tipo de problemas, debido a que cuando nos adentremos a la clasificación multiclase, estas métricas nos serán de utilidad.

### 6.2.1. Matriz de confusión

Considérese un problema de clasificación binario. En el mejor de los casos, el modelo podría predecir perfectamente qué datos pertenecen a la clase y cuáles no lo hacen. Sin embargo, siendo que es un modelo estadístico, el modelo no siempre acertará, y usando los datos etiquetados se puede detectar en dónde ha habido una *confusión*. El modelo puede confundirse de dos maneras: La primera es afirmando que  $y \in \mathcal{C}$  cuando no es el caso, y la segunda es clasificar  $y \notin \mathcal{C}$ , cuando en realidad  $y$  sí pertenecía a la clase. Por tanto, existen 4 posibilidades:

1. **Verdaderos positivos:** Aquellos datos que sí pertenecen a la clase, y fueron clasificados dentro de la clase. A la cantidad de verdaderos positivos se le denota  $T_P$  por sus siglas en inglés.
2. **Falsos positivos:** Aquellos datos que sí pertenecen a la clase, y fueron clasificados fuera de la clase. A la cantidad de falsos positivos se le denota  $F_P$  por sus siglas en inglés.
3. **Verdaderos negativos:** Aquellos datos que no pertenecen a la clase, y fueron clasificados fuera de la clase. A la cantidad de verdaderos negativos se le denota  $T_N$  por sus siglas en inglés.
4. **Falsos negativos:** Aquellos datos que no pertenecen a la clase, y fueron clasificados dentro de la clase. A la cantidad de falsos negativos se le denota  $F_N$  por sus siglas en inglés.

Es posible agrupar toda esta información en una matriz conocida como *matriz de confusión*.

**Definición 6.1.** *Considérese el problema de clasificación (2.9) con  $m = 2$ . Sea  $f : \mathbb{R}^n \rightarrow \mathcal{C}$*

el modelo. Definimos los siguientes valores:

$$T_P = |\{y_i : c_i = (1, 0)^T \text{ y } f(y_i) = (1, 0)^T\}|$$

$$F_P = |\{y_i : c_i = (0, 1)^T \text{ y } f(y_i) = (1, 0)^T\}|$$

$$T_N = |\{y_i : c_i = (0, 1)^T \text{ y } f(y_i) = (0, 1)^T\}|$$

$$F_N = |\{y_i : c_i = (1, 0)^T \text{ y } f(y_i) = (0, 1)^T\}|.$$

La matriz de confusión se define como

$$D = \begin{bmatrix} T_P & F_P \\ F_N & T_N \end{bmatrix}. \quad (6.1)$$

		<b>Ground Truth</b>	
		Positive	Negative
<b>Prediction</b>	Positive	True positives	False positives
	Negative	False negatives	True negatives

Figura 6.2: Conseguir imagen propia

### 6.2.2. Exactitud

La exactitud se define como la razón de las predicciones acertadas y las predicciones totales. En el aprendizaje automático suele ser la métrica principal, ya que determina el porcentaje de aciertos. Sin embargo, no es el único factor a tomar en cuenta.

**Definición 6.2** (exactitud). *Considérese los valores  $T_P, F_P, T_N, F_N$  de la definición (6.1). La exactitud  $A$  se define como*

$$A = \frac{T_P + T_N}{T_P + F_P + T_N + F_N} \quad (6.2)$$

### 6.2.3. Sensibilidad y Especificidad

Existen ocasiones en las que no es conveniente utilizar la Exactitud como único criterio del desempeño, especialmente cuando se tiene un conjunto de datos desbalanceado (Sección 1.3.5). Por ejemplo, al detectar llamadas fraudulentas, no servirá usar la Exactitud, porque casi ninguna llamada es fraudulenta. Lo conveniente sería conocer cuántas veces se acertó al tratar de predecir resultados negativos (Especificidad) o viceversa, cuántas veces se acertó al tratar de predecir resultados positivos (Sensibilidad).

**Definición 6.3.** *Considérese los valores  $T_P, F_P, T_N, F_N$  de la definición (6.1). La sensibilidad  $R$  y la especificidad  $E$  se definen respectivamente como*

$$R = \frac{T_P}{T_P + F_P} \quad (6.3)$$

$$E = \frac{T_N}{T_N + F_N} \quad (6.4)$$

### 6.2.4. Precisión

**Definición 6.4** (Precisión). *Considérese los valores  $T_P, F_P, T_N, F_N$  de la definición (6.1). La precisión  $P$  se define como*

$$P = \frac{T_P}{T_P + F_P} \quad (6.5)$$

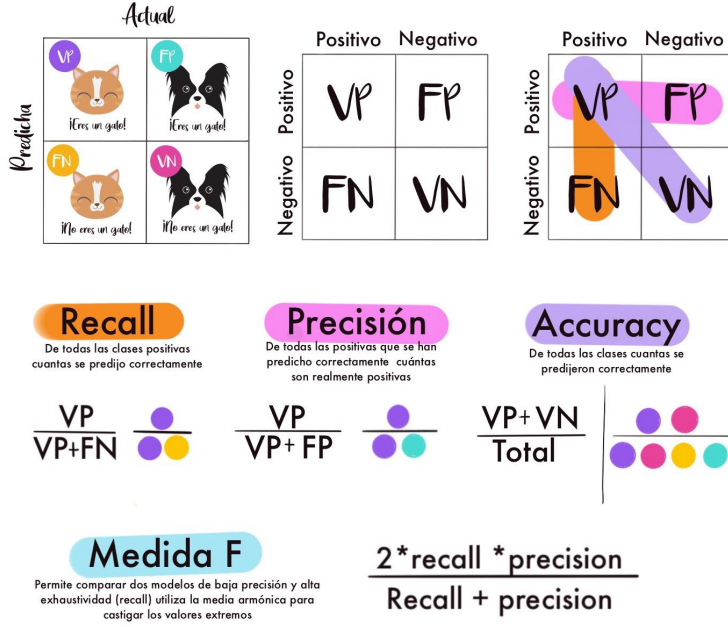
### 6.2.5. **F1-score**

En algunos problemas, es necesario tomar en cuenta la precisión y la sensibilidad por igual. La F1-score, propuesta en 2006 en [65], se define como la media armónica de estas dos métricas.

**Definición 6.5 (F1-score).** *Considérese los valores  $T_P, F_P, T_N, F_N$  de la definición (6.1).*

*La **F1-score**  $F_1$  se define como*

$$F_1 = \frac{2}{\frac{1}{R} + \frac{1}{P}} = \frac{2RP}{R + P} = \frac{T_P}{T_P + \frac{1}{2}(F_P + F_N)} \quad (6.6)$$



### 6.3. Métricas para clasificación multiclase

En cuanto a la problema de clasificación con  $m$  clases, no podemos hacer uso directo de las todas las fórmulas anteriores. Sin embargo, es posible conseguir una matriz de confusión. En este caso, el eje  $x$  determina la verdadera etiqueta, y el eje  $y$  determina la etiqueta predicha por el modelo.

De modo que la matriz de confusión para el problema de clasificación multiclase es de la siguiente forma:

$$D = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,m} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m,1} & d_{m,2} & \cdots & d_{m,m} \end{pmatrix}, \quad (6.7)$$

dónde  $d_{i,j}$  es la cantidad de elementos pertenecientes a la clase  $j$  que fueron clasificados en la clase  $i$ . Por tanto, un modelo que clasifique todas las instancias correctamente, tendrá una matriz de confusión diagonal.

**Definición 6.6.** Sea  $D$  una matriz de confusión, su versión normalizada se determina por la regla

$$\hat{d}_{i,j} = \frac{d_{i,j}}{|C_i|} \quad (6.8)$$

Es sencillo definir la forma generalizada de la Exactitud, basándonos en que es la razón entre la cantidad total de aciertos (La suma de los elementos en la diagonal de la matriz de confusión) y la cantidad total de instancias, es decir la suma de cada entrada de la matriz.

**Definición 6.7.** Sea  $D$  una matriz de confusión. Se define la exactitud  $A$  de la siguiente manera:

1. Exactitud:

$$A = \frac{\sum_{i=1}^n d_{i,i}}{\sum_{i,j} d_{i,j}} \quad (6.9)$$

All Confusion Matrix				
Output Class	1	2	3	
	1974 32.2%	0 0.0%	0 0.0%	100% 0.0%
	1 0.0%	2092 34.2%	2 0.0%	99.9% 0.1%
	2 0.0%	0 0.0%	2053 33.5%	99.9% 0.1%
				99.8% 0.2%
				100% 0.0%
				99.9% 0.1%
				99.9% 0.1%
				Target Class
				1 2 3

Figura 6.3: [Conseguir imagen propia](#)

Por otro lado, para las métricas de Precisión y Sensibilidad se tienen dos posibles definiciones: La *Macro* y la *Ponderada*.



**6.3.1. Métricas Macro****6.3.2. Métricas Ponderadas**

# Bibliografía

- [1] Reinhard Klette. *Concise Computer Vision - An Introduction into Theory and Algorithms*. 01 2014.
- [2] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1 edition, 1997.
- [3] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019.
- [4] positividad covid-19 en méxico.
- [5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.
- [6] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.
- [7] Shengguo Hu, Yanfeng Liang, Lintao Ma, and Ying He. Msmote: Improving classification performance when training data is imbalanced. *Computer Science and Engineering, International Workshop on*, 2:13–17, 01 2009.
- [8] Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

- [9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [11] Linan Zhang and Hayden Schaeffer. Forward stability of resnet and its variants. *CoRR*, abs/1811.09885, 2018.
- [12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [13] Mawardi Bahri, Ryuichi Ashino, and Rémi Vaillancourt. Convolution theorems for quaternion fourier transform: Properties and applications. *Abstract and Applied Analysis*, 2013:162769, Nov 2013.
- [14] Allen C. Pipkin. *Volterra Equations*. Springer New York, New York, NY, 1991.
- [15] Md. Amirul Islam, Matthew Kowal, Sen Jia, Konstantinos G. Derpanis, and Neil D. B. Bruce. Position, padding and predictions: A deeper look at position information in cnns. *CoRR*, abs/2101.12322, 2021.
- [16] Guilin Liu, Kevin J. Shih, Ting-Chun Wang, Fitsum A. Reda, Karan Sapra, Zhiding Yu, Andrew Tao, and Bryan Catanzaro. Partial convolution based padding. *CoRR*, abs/1811.11718, 2018.
- [17] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML’10, page 111–118, Madison, WI, USA, 2010. Omnipress.

- [18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for on-line learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.
- [21] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
- [22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [23] What is a convolutional neural network?
- [24] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.
- [25] Michael Hauser, Sean Gunn, Samer Saab Jr., and Asok Ray. State space representations of deep neural networks. *CoRR*, abs/1806.03751, 2018.
- [26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

- [27] Wadii Boulila, Maha Driss, Mohammed Al-Sarem, Faisal Saeed, and Moez Krichen. Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives. *CoRR*, abs/2102.07004, 2021.
- [28] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.
- [29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [30] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *CoRR*, abs/1705.03341, 2017.
- [31] Timothy Sauer. *Numerical Analysis*. Addison-Wesley Publishing Company, USA, 2nd edition, 2011.
- [32] Uri M. Ascher. Numerical methods for evolutionary differential equations. 2008.
- [33] T. Archibald, Craig Fraser, and Ivor Grattan-Guinness. The history of differential equations, 1670–1950. *Oberwolfach Reports*, pages 2729–2794, 01 2004.
- [34] Jan Dereziński and Adam Latosiński. From heun class equations to painlevé equations, 07 2020.
- [35] Gerd Baumann. *Mathematica for Theoretical Physics: Classical Mechanics and Non-linear Dynamics*. Springer, 2nd edition, 2005.
- [36] Richard Fitzpatrick. Maxwell’s equations and the principles of electromagnetism. 2008.
- [37] F. Haas. Review: Richard bellman, stability theory of differential equations. *Bulletin of the American Mathematical Society*, 60(4):400 – 401, 1954.

- [38] S.-M. Jung. Hyers-ulam stability of linear differential equations of first order. *Applied Mathematics Letters*, 17(10):1135–1140, 2004.
- [39] Robert M. M. Mattheij Uri M. Ascher and Robert D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations (Classics in Applied Mathematics)*. SIAM, 1987.
- [40] John C. Butcher. *Numerical methods for ordinary differential equations*. Wiley, 2nd ed edition, 2008.
- [41] J. M. Sanz-Serna. Symplectic runge-kutta schemes for adjoint equations, automatic differentiation, optimal control and more, 2015.
- [42] J M Sanz-Serna. Runge-kutta schemes for hamiltonian systems. *BIT Numerical Mathematics*, 28(4):877–883, December 1988.
- [43] F M Lasagni. Canonical Runge-Kutta methods. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 39(6):952–953, November 1988.
- [44] Suresh P. Sethi. *Optimal Control Theory*. Number 978-3-319-98237-3 in Springer Books. Springer, December 2019.
- [45] C. Patrick Koelling William G. Sullivan, Elin M. Wicks. *Engineering Economy*. Pearson Education, 16th edition, c2015.
- [46] Richard Bellman. *Dynamic Programming*. Princeton Landmarks in Mathematics and Physics. Dover Publications, 2003.
- [47] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *CoRR*, abs/1710.10121, 2017.
- [48] Qianli Liao and Tomaso A. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *CoRR*, abs/1604.03640, 2016.

- [49] Xinshi Chen. Review: Ordinary differential equations for deep learning. *CoRR*, abs/1911.00502, 2019.
- [50] Qianxiao Li and Shuji Hao. An optimal control approach to deep learning and applications to discrete-weight neural networks. *CoRR*, abs/1803.01299, 2018.
- [51] Weinan E, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1), Dec 2018.
- [52] Qianxiao Li, Long Chen, Cheng Tai, and Weinan E. Maximum principle based algorithms for deep learning. *CoRR*, abs/1710.09513, 2017.
- [53] Martin Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: models and numerical methods, 2019.
- [54] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *CoRR*, abs/1804.04272, 2018.
- [55] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. *CoRR*, abs/1611.05725, 2016.
- [56] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.
- [57] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtam. Reversible architectures for arbitrarily deep residual neural networks. *CoRR*, abs/1709.03698, 2017.
- [58] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. *CoRR*, abs/1707.04585, 2017.
- [59] Linda Ruth Petzold Uri M. Ascher. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. 1998.

- [60] Uri M. Ascher, Steven J. Ruuth, and Brian T. R. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM Journal on Numerical Analysis*, 32(3):797–823, 1995.
- [61] Eldad Haber, Keegan Lensink, Eran Treister, and Lars Ruthotto. Imexnet: A forward stable deep neural network. *CoRR*, abs/1903.02639, 2019.
- [62] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks, 2019.
- [63] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018.
- [64] Pollen challenge, 2020.
- [65] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. volume Vol. 4304, pages 1015–1021, 01 2006.