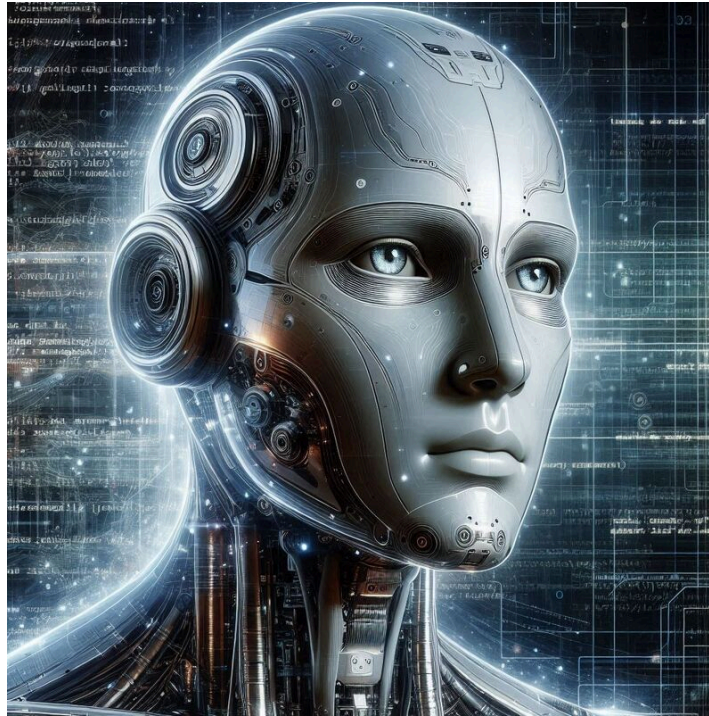


# DESARROLLO DE SISTEMAS DE IA

Profesor Carlos Charletti



## GRUPO 4

### Integrantes

**Cristian Falco**

**Eduardo Figueroa**

**Jorge Flores**

**Carlos Giménez**

**Cinthyia Gomez**

**Montserrat Gutierrez**

**Jorgelina Tissera**



# Informe Técnico

Exploración de la convergencia en Redes Neuronales

# Indice

<b>Arquitecturas de redes neuronales.....</b>	<b>6</b>
Redes neuronales perceptrón:.....	6
Redes neuronales convolucionales:.....	7
Redes neuronales recurrentes:.....	7
<b>Aprendizaje en redes neuronales.....</b>	<b>8</b>
Descenso del gradiente.....	8
Funcionamiento del descenso del gradiente.....	9
Ventajas del descenso del gradiente.....	9
Desventajas del descenso del gradiente.....	9
Hiperparámetros del descenso del gradiente.....	9
Tasa de aprendizaje:.....	9
Momentum:.....	10
Forward Propagation.....	10
Paso a paso de la propagación hacia adelante:.....	10
Entrada de datos:.....	10
Propagación a través de capas ocultas:.....	10
Salida final:.....	10
Backpropagation:.....	11
El motor de aprendizaje de las redes neuronales.....	11
Proceso paso a paso:.....	12
La magia de la cadena de derivadas:.....	12
Beneficios de la backpropagation:.....	12
La importancia de la backpropagation:.....	13
Componentes de una red neuronal.....	13
Funciones de Activación.....	13
Tipos de funciones de activación.....	13
Elección de la función de activación adecuada.....	15
Consideraciones adicionales.....	16
Funciones de Costo.....	16
Características de las Funciones de Costo.....	16
Tipos Comunes de Funciones de Costo.....	16
Selección de la Función de Costo Adecuada.....	18
Consideraciones Adicionales.....	18
Tasa de Aprendizaje.....	18
Función de la tasa de aprendizaje.....	18
Tipos de estrategias de tasa de aprendizaje.....	19
Selección de la tasa de aprendizaje adecuada.....	19
Herramientas para optimizar la tasa de aprendizaje.....	19
Consideraciones adicionales.....	20
Regularizando la red neuronal.....	20
Regularización L1 (Lasso).....	20
Regularización L2 (Ridge).....	21

Elegir entre L1 y L2.....	22
Consideraciones adicionales.....	22
Dropout.....	22
¿Cómo funciona el Dropout?.....	23
Beneficios del Dropout.....	23
Implementación del Dropout.....	23
Consideraciones adicionales.....	23
Conclusión.....	24
colab.....	25
Código desarrollado.....	25
Metodología:.....	29
Implementación de Forward Propagation, Backpropagation y GD:.....	29
Modificación de hiperparámetros:.....	29
Habilitación de una arquitectura CNN:.....	30
Resultados:.....	30
Conclusiones:.....	30
Un equipo diverso, un objetivo común.....	31
Explorando el laberinto de optimizadores.....	31
Activando el poder de las funciones de activación.....	32
Ajustando los parámetros con precisión.....	33
La búsqueda por cuadrilla: Un esfuerzo colaborativo.....	35
El veredicto final: ¡RMSprop emerge victorioso!.....	37

# Introducción

La inteligencia artificial (IA) ha revolucionado numerosos campos al ofrecer soluciones innovadoras para problemas complejos. En particular, las redes neuronales surgen como una herramienta esencial en el procesamiento de datos y el aprendizaje automático.

Este proyecto se centra en la convergencia de datos en redes neuronales, un aspecto crucial para optimizar el rendimiento y la precisión de modelos de IA.

## Objetivo del Proyecto

El principal objetivo de este trabajo es explorar en las arquitecturas y los métodos de aprendizaje de las redes neuronales, y cómo estas técnicas contribuyen a la convergencia de datos.

Se busca alcanzar una comprensión de los distintos tipos de redes neuronales, los algoritmos de optimización como el descenso del gradiente, y los mecanismos de propagación hacia adelante y hacia atrás que son fundamentales para el entrenamiento efectivo de estos modelos.

## Utilidad del Proyecto

Este conocimiento es invaluable para investigadores y profesionales del campo de la inteligencia artificial. Al desglosar y analizar los componentes de las redes neuronales, desde las arquitecturas hasta los hiperparámetros y las funciones de activación, se alcanza una mayor comprensión para facilitar el diseño y la implementación de modelos eficientes.

Además, la implementación práctica y el análisis de resultados permiten una metodología aplicativa que puede ser trasladada a diversos problemas reales, desde la clasificación de imágenes hasta la predicción de series temporales.

## Convergencia en Redes Neuronales

La convergencia en redes neuronales se refiere al proceso mediante el cual un modelo de IA ajusta sus parámetros hasta alcanzar un estado óptimo en el que el error de predicción se minimiza.

Este proceso es crucial para asegurar que la red neuronal aprenda de manera efectiva y generalice bien a datos nuevos.

La convergencia es un indicador de que el modelo ha aprendido las relaciones subyacentes en los datos de entrenamiento y está listo para aplicarse a tareas prácticas.

En este proyecto, nuestro equipo se embarca en una exploración detallada de los factores que influyen en la convergencia, utilizando métodos tanto teóricos como experimentales para entender mejor este fenómeno.

## Estructura del Proyecto

El contenido del trabajo está estructurado de manera lógica y progresiva, cubriendo los siguientes puntos clave:

### Arquitecturas de Redes Neuronales:

- Redes neuronales perceptrón
- Redes neuronales convolucionales (CNN)
- Redes neuronales recurrentes (RNN)

### Aprendizaje en Redes Neuronales:

- Descenso del gradiente y sus variantes
- Forward Propagation
- Backpropagation
- Componentes de una Red Neuronal:

### Funciones de activación y su selección adecuada

- Funciones de costo y su optimización
- Regularización y técnicas como Dropout
- Metodología y Resultados:

### Implementación práctica de los algoritmos

- Ajuste de hiperparámetros y optimización
- Resultados obtenidos y análisis de rendimiento

## Resumen del proyecto

Este trabajo se concibe como una herramienta exploratoria, dentro de este campo particular de la IA, destacando la importancia de la convergencia de datos y la optimización continua en los proyectos.

Es decir, tratar de ofrecer una visión general y práctica de cómo las redes neuronales procesan y convergen datos para resolver problemas complejos.

Al combinar teoría y práctica, se pretende alcanzar la técnica y el conocimiento necesario para implementar y optimizar redes neuronales en diversas aplicaciones de inteligencia artificial.

# I - Las Redes neuronales

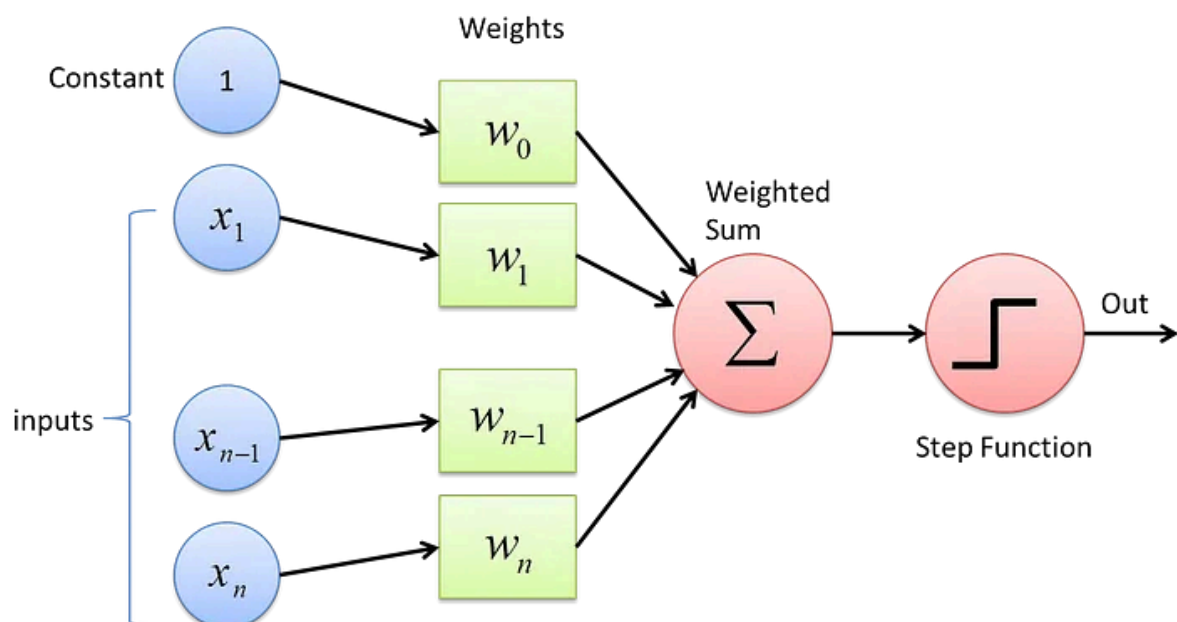
Las redes neuronales artificiales son un campo de la inteligencia artificial inspirado en el funcionamiento del cerebro humano. Están compuestas por unidades interconectadas llamadas neuronas artificiales, que procesan información y aprenden de los datos. Las redes neuronales han demostrado ser una herramienta poderosa para resolver una amplia gama de problemas, incluyendo el reconocimiento de imágenes, el procesamiento del lenguaje natural y la generación de contenido.

## Arquitecturas de redes neuronales

Existen diferentes arquitecturas de redes neuronales, cada una con sus propias características y aplicaciones. Algunas de las arquitecturas más comunes incluyen:

### Redes neuronales perceptrón:

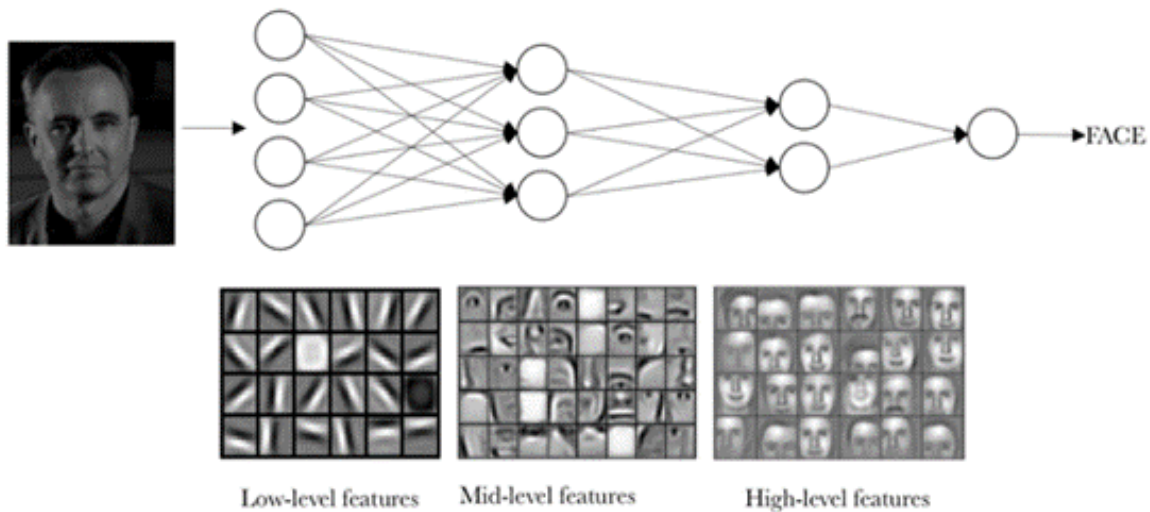
Son la arquitectura más simple, y consisten en una capa de entrada, una capa de salida y una o más capas ocultas. Las neuronas en cada capa están conectadas entre sí por pesos.





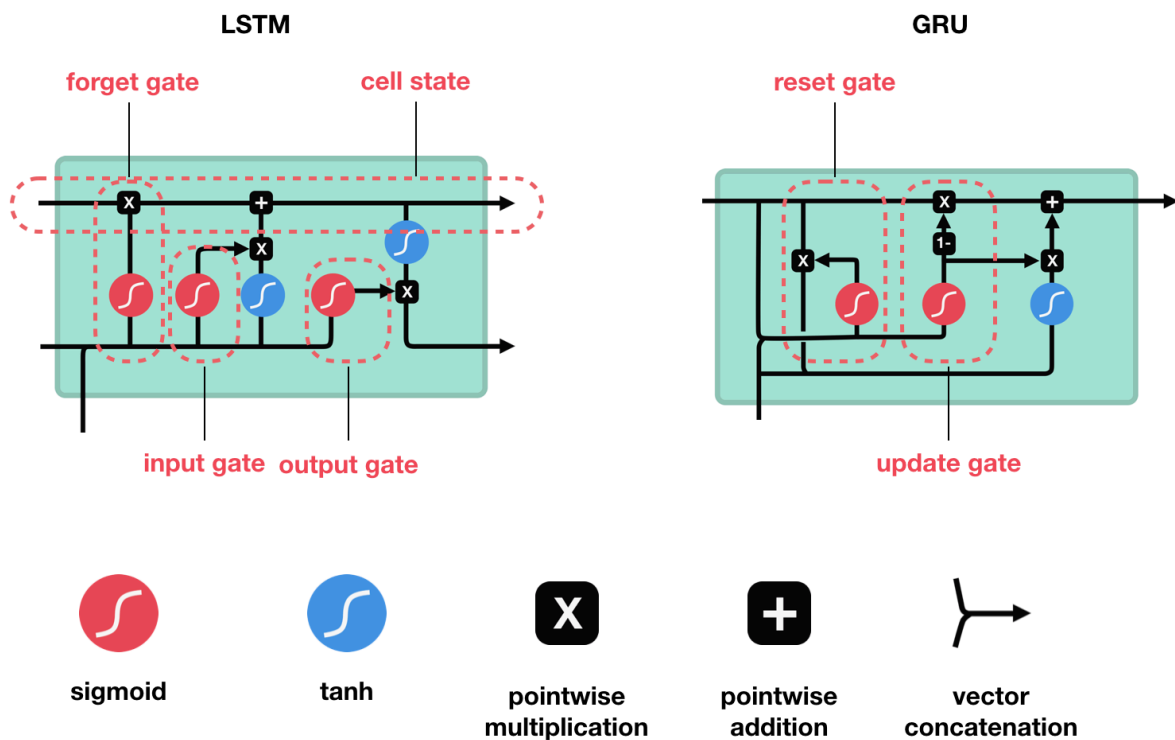
## Redes neuronales convolucionales:

Son utilizadas para el reconocimiento de imágenes y el procesamiento de señales. Estas redes emplean filtros convolucionales para extraer características de los datos de entrada.



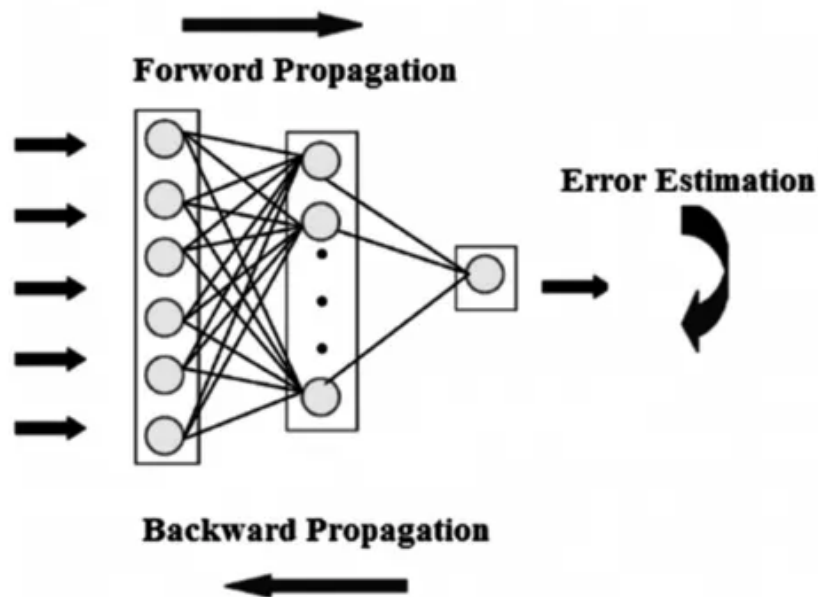
## Redes neuronales recurrentes:

Son utilizadas para procesar secuencias de datos, como el texto o el habla. Estas redes tienen conexiones recurrentes que les permiten almacenar información del pasado y utilizarla para procesar el presente.



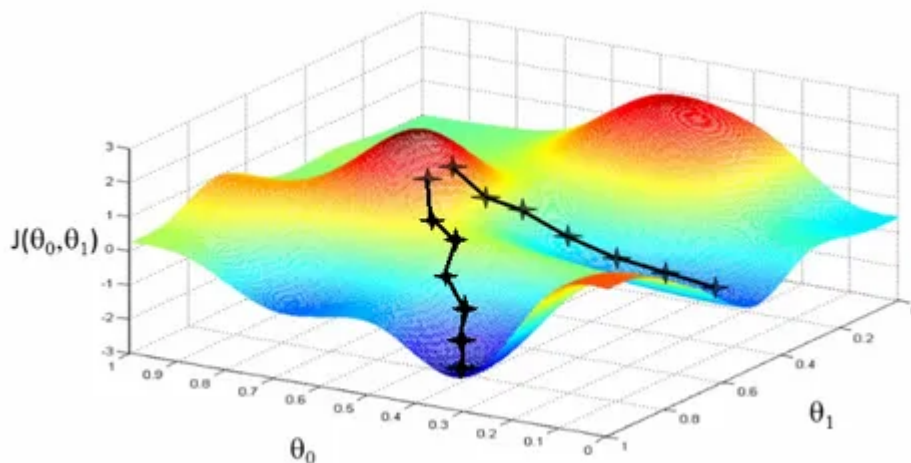
# Aprendizaje en redes neuronales

Las redes neuronales aprenden de los datos mediante un proceso llamado aprendizaje automático. El objetivo del aprendizaje automático es encontrar un conjunto de pesos que permita a la red neuronal generar predicciones precisas para nuevos datos. Existen diferentes algoritmos de aprendizaje automático, como el descenso del gradiente y la propagación hacia atrás.



## Descenso del gradiente

El descenso del gradiente es un algoritmo de optimización utilizado para encontrar el mínimo de una función. Es uno de los algoritmos más utilizados en el aprendizaje automático, ya que es simple, eficiente y efectivo para una amplia gama de problemas.



## Funcionamiento del descenso del gradiente

El descenso del gradiente funciona iterativamente, moviéndose en la dirección de la pendiente negativa de la función. La pendiente negativa indica la dirección en la que la función disminuye más rápido, por lo que seguirla nos lleva al mínimo.

El algoritmo se puede resumir en los siguientes pasos:

- **Inicializar los parámetros:** Se inicializan los parámetros del modelo con valores aleatorios o predeterminados.
- **Calcular el gradiente:** Se calcula el gradiente de la función de pérdida con respecto a los parámetros del modelo.
- **Actualizar los parámetros:** Se actualizan los parámetros del modelo restando un pequeño paso de la pendiente.
- **Repetir** los pasos 2 y 3: Se repiten los pasos 2 y 3 hasta que la función de pérdida converja a un mínimo o se alcance un número máximo de iteraciones.

## Ventajas del descenso del gradiente

- Es un algoritmo simple y eficiente.
- Es fácil de implementar.
- Funciona con una amplia gama de funciones.
- Puede encontrar mínimos locales y globales.

## Desventajas del descenso del gradiente

- Puede quedar atrapado en mínimos locales.
- Puede ser sensible al factor de aprendizaje.
- Puede ser lento para converger en algunas funciones.

## Hiperparámetros del descenso del gradiente

El descenso del gradiente tiene varios hiperparámetros que pueden afectar su rendimiento, como:

### Tasa de aprendizaje:

La tasa de aprendizaje controla la magnitud de las actualizaciones de los parámetros. Una tasa de aprendizaje demasiado grande puede hacer que el algoritmo se salga del mínimo, mientras que una tasa de aprendizaje demasiado pequeña puede hacer que el algoritmo converja lentamente.

### Momentum:

El momentum es una técnica que ayuda a acelerar la convergencia del algoritmo. El momentum agrega un término de velocidad a las actualizaciones de los parámetros, lo que ayuda a que el algoritmo se mueva en una dirección más consistente.

## Forward Propagation

Es el proceso de alimentar información a través de la red, desde la capa de entrada hasta la capa de salida, para generar una predicción.

La información de entrada, ya sean números, imágenes o texto, se introduce en la capa de entrada. Cada neurona en esta capa recibe la información y la combina con pesos adjuntos a las conexiones.

A continuación, se aplica una función de activación, como la sigmoide o la ReLU, para transformar la suma ponderada. Esta función de activación introduce una no linealidad importante en la red, permitiendo que la red aprenda patrones complejos en los datos.

El resultado de la función de activación se convierte en la salida de la neurona y se propaga como entrada a las neuronas de la siguiente capa. Este proceso se repite capa por capa hasta llegar a la capa de salida, donde se genera la predicción final.

### Paso a paso de la propagación hacia adelante:

#### Entrada de datos:

Se alimenta la información de entrada a la capa de entrada de la red neuronal.

#### Propagación a través de capas ocultas:

Cada neurona en una capa oculta recibe la salida de las neuronas de la capa anterior.

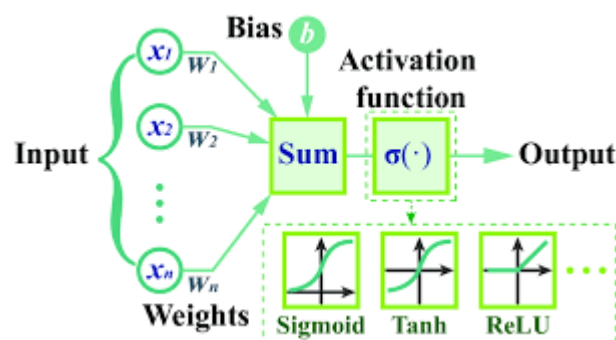
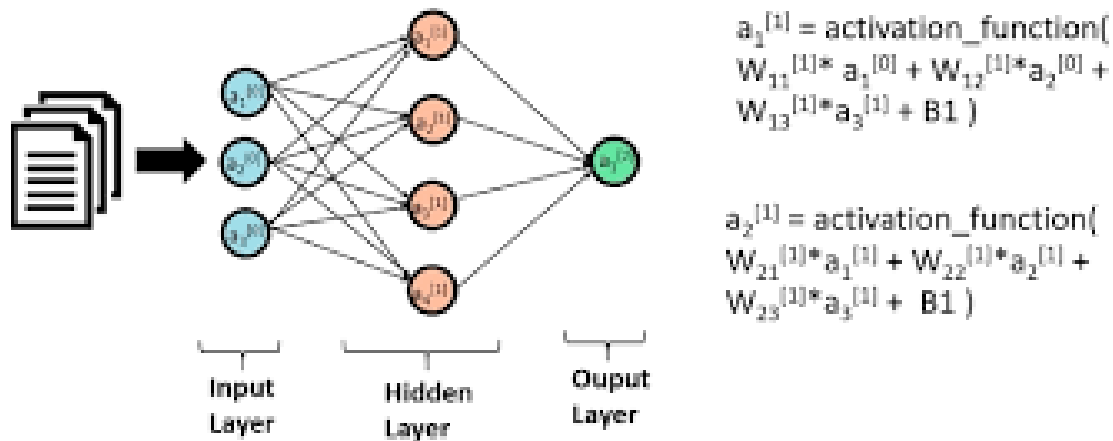
Las entradas se multiplican por los pesos correspondientes y se suman.

Se aplica una función de activación a la suma ponderada.

El resultado de la activación se convierte en la salida de la neurona y se propaga a la siguiente capa.

#### Salida final:

La capa de salida procesa las entradas de la última capa oculta y genera la predicción final. La propagación hacia adelante es la base para entrenar redes neuronales. Al comparar la predicción con el valor real deseado (salida objetivo), se calcula el error. Este error se propaga hacia atrás a través de la red mediante un proceso llamado backpropagation, que ajusta los pesos de las conexiones para minimizar el error en la siguiente iteración de forward propagation.



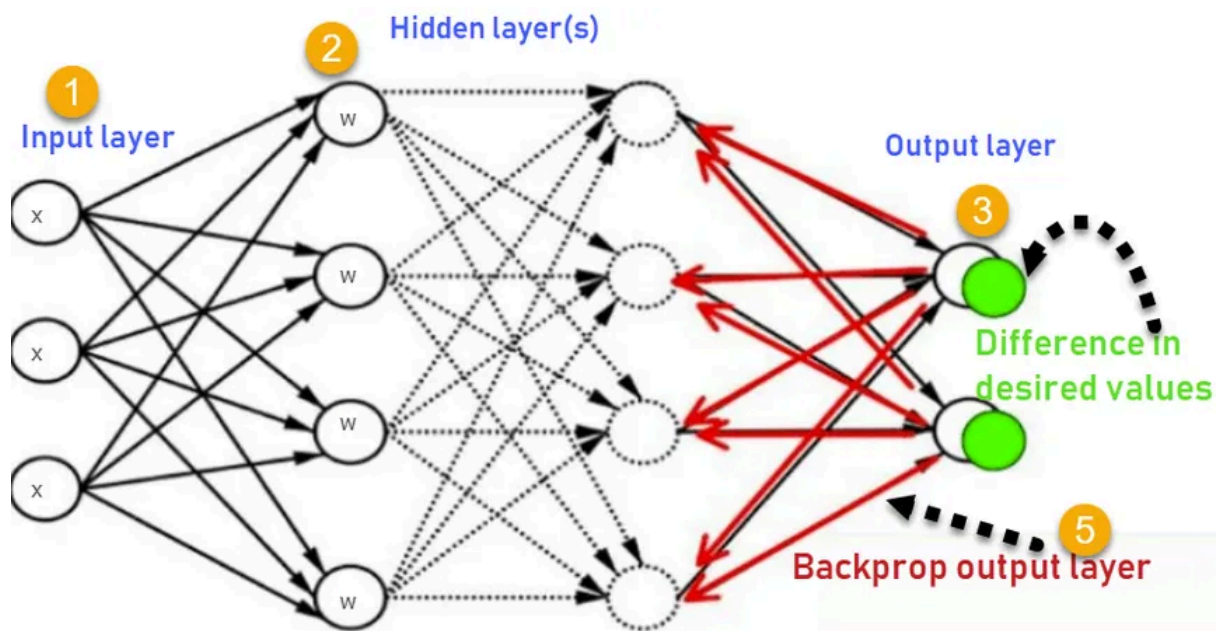
## Backpropagation:

El motor de aprendizaje de las redes neuronales

La red neuronal artificial (RNA) ha revolucionado la inteligencia artificial gracias a su capacidad para aprender de los datos. Pero, ¿cómo aprende una red neuronal? Ahí es donde entra en juego la backpropagation o propagación hacia atrás.

La información entra por la capa de entrada, fluye a través de capas ocultas y llega a una salida. La forward propagation (propagación hacia adelante) calcula esta salida inicial. Pero, ¿cómo sabe la red si esa salida es correcta? Aquí es donde entra la backpropagation.

La backpropagation es un algoritmo de aprendizaje iterativo que permite a la red neuronal ajustar sus pesos internos para minimizar el error entre la salida predicha y la salida real deseada.



Proceso paso a paso:

1. **Calcular el error:** Se compara la salida de la red neuronal con la salida real deseada. La diferencia entre ambas se denomina error.
2. **Propagación hacia atrás del error:** El error se propaga hacia atrás, capa por capa, comenzando por la capa de salida.
3. **Ajuste de pesos:** En cada capa, se calcula cuánto contribuye cada peso a la salida final y al error general. Estos valores de contribución se denominan gradientes.
4. **Actualización de pesos:** Utilizando los gradientes, se ajustan los pesos de las conexiones entre neuronas. Los pesos se actualizan en la dirección opuesta al gradiente del error, lo que significa que la red se aleja de las decisiones que provocaron un error alto.
5. **Repetición:** Los pasos 1 a 4 se repiten para cada ejemplo de entrenamiento en el conjunto de datos. Con cada iteración, la red neuronal aprende de sus errores y ajusta sus pesos para generar mejores predicciones.

La magia de la cadena de derivadas:

El cálculo de los gradientes en la backpropagation se basa en la cadena de derivadas, un concepto matemático que permite calcular cómo un cambio en la entrada a una función compuesta afecta a la salida final. Al aplicar la cadena de derivadas a la red neuronal capa por capa, se puede determinar cuánto contribuye cada peso al error final.

Beneficios de la backpropagation:

- **Aprendizaje automático:** Permite a las redes neuronales aprender de los datos sin necesidad de programarlas explícitamente para cada tarea.

- **Optimización de la red:** Ajusta los pesos para minimizar el error y mejorar la precisión de las predicciones.
- **Flexibilidad:** Se aplica a diversas arquitecturas de redes neuronales, desde redes simples hasta redes profundas complejas.

La importancia de la backpropagation:

La backpropagation es un algoritmo crítico para el entrenamiento de redes neuronales profundas con múltiples capas ocultas. Sin él, estas redes no serían capaces de aprender patrones complejos en los datos. Esencialmente, permite a la red **"aprender de sus errores"** y mejorar gradualmente su rendimiento a medida que se entrena con más datos.

## Componentes de una red neuronal

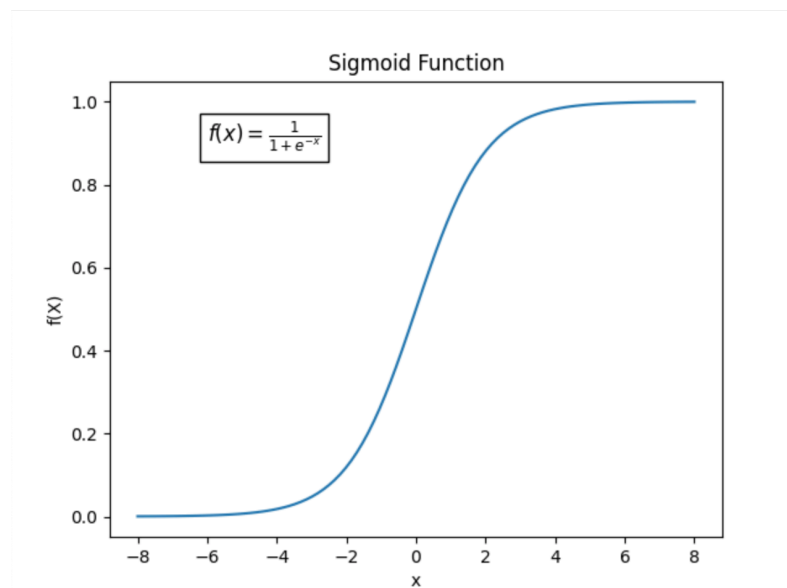
### Funciones de Activación

En una red neuronal, cada neurona recibe entradas, las procesa y genera una salida. Las funciones de activación se aplican a la salida de la neurona para introducir una no linealidad crucial en el sistema. Sin ellas, las redes neuronales solo podrían realizar transformaciones lineales, lo que limitaría severamente su capacidad de aprendizaje.

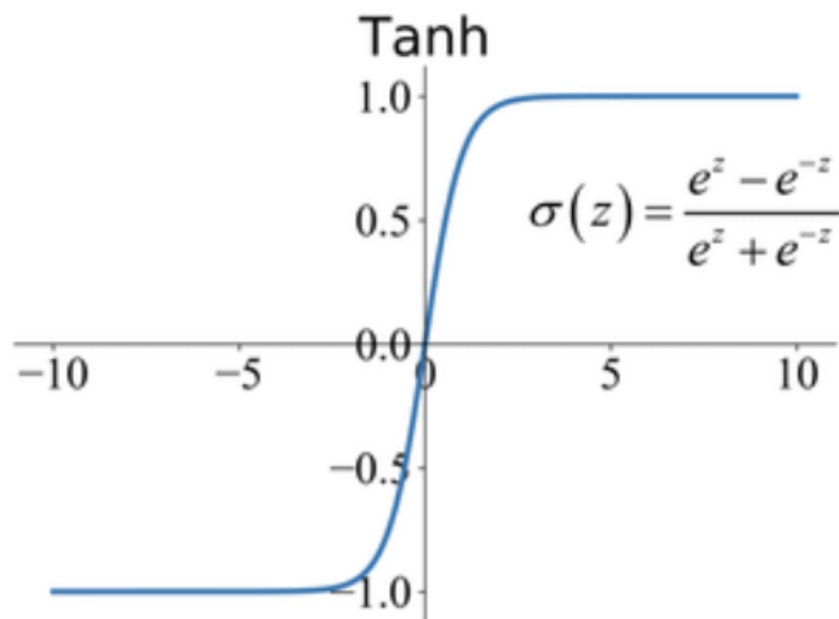
Tipos de funciones de activación

Existen diversas funciones de activación, cada una con sus propias características y propiedades. Algunas de las más comunes son:

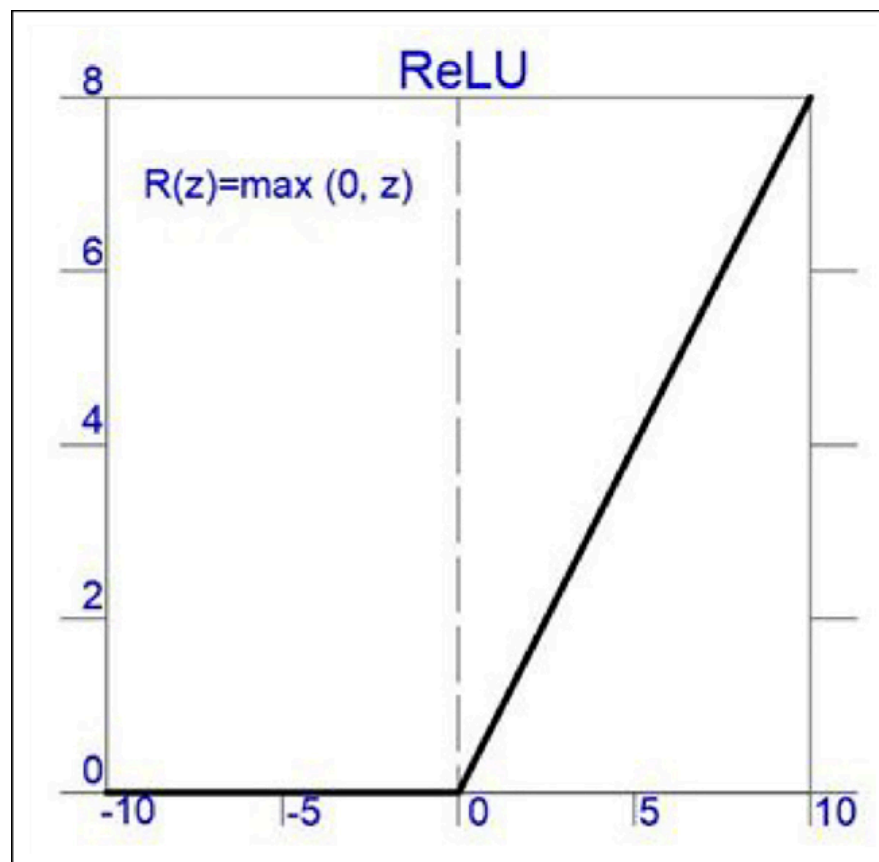
**Sigmoide:** Esta función transforma la entrada en un valor entre 0 y 1, asemejándose a una curva en forma de "S". Es útil para modelar probabilidades.



**TanH (Tangente hiperbólica):** Similar a la sigmoide, la TanH produce una salida entre -1 y 1. Es útil para tareas que requieren un rango de salida más amplio.

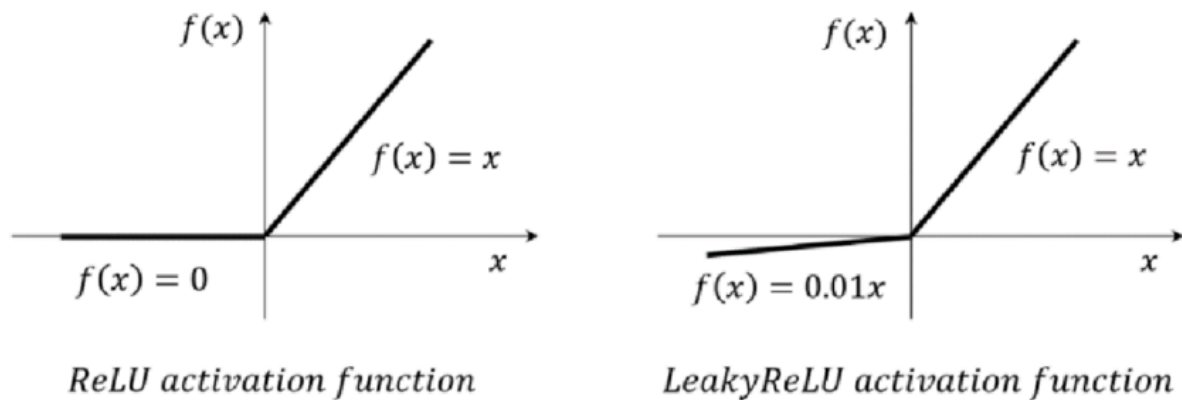


**ReLU (Rectified Linear Unit):** Esta función devuelve la entrada si es positiva, o 0 si es negativa. Es popular por su simplicidad y eficiencia computacional.



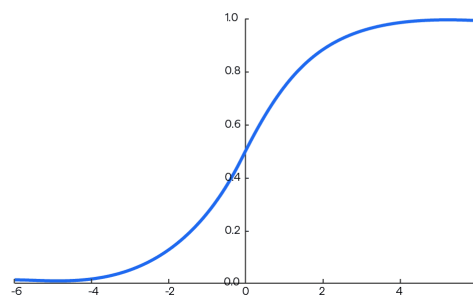


**Función escalonada suave (Leaky ReLU):** Una variante del ReLU, que devuelve un pequeño valor negativo si la entrada es negativa. Ayuda a evitar el problema de la "muerte de las neuronas" durante el entrenamiento.



**Función softmax:** Esta función normaliza un vector de entradas en distribuciones de probabilidad, donde cada elemento representa la probabilidad de pertenecer a una clase específica. Es útil para tareas de clasificación multiclase.

## Softmax Function



### Elección de la función de activación adecuada

La elección de la función de activación adecuada para una tarea específica depende de diversos factores, como el tipo de tarea (clasificación, regresión, etc.), la arquitectura de la red y los datos de entrenamiento.

En general, las funciones sigmoide y TanH son adecuadas para tareas de clasificación binaria, mientras que Softmax se utiliza para clasificación multiclase. ReLU y Leaky ReLU son populares para una amplia gama de tareas, incluyendo clasificación y regresión.

## Consideraciones adicionales

**Derivadas:** Las funciones de activación deben tener derivadas bien definidas para permitir el entrenamiento de redes neuronales mediante algoritmos de retropropagación.

**Saturación:** Algunas funciones, como la sigmoide y TanH, pueden saturarse en valores extremos de entrada, lo que puede afectar el aprendizaje.

**Escalamiento:** Es importante escalar los datos de entrada a un rango apropiado para evitar problemas de saturación y mejorar el rendimiento de la red.

Las funciones de activación son elementos primordiales en las redes neuronales, proporcionando la no linealidad necesaria para modelar relaciones complejas en los datos. La elección de la función de activación adecuada puede tener un impacto significativo en el rendimiento de la red. La comprensión de las características y propiedades de las diferentes funciones de activación es esencial para el desarrollo de redes neuronales efectivas.

## Funciones de Costo

Su función principal es evaluar la discrepancia entre la predicción de la red y el valor real deseado (salida objetivo). Esta evaluación guía el proceso de ajuste de parámetros de la red para minimizar el error y mejorar su rendimiento.

### Características de las Funciones de Costo

- **Diferenciabilidad:** Las funciones de costo deben ser diferenciables para permitir el cálculo del gradiente, que es esencial para el algoritmo de backpropagation.
- **Minimización:** El objetivo es encontrar los parámetros de la red que minimicen la función de costo, lo que significa que la red genera predicciones lo más cercanas posibles a los valores reales.
- **Interpretabilidad:** Las funciones de costo deben ser interpretables para comprender mejor el comportamiento de la red y la naturaleza del error.
- **Robustez:** Las funciones de costo deben ser robustas a valores atípicos o ruido en los datos.

### Tipos Comunes de Funciones de Costo

**Error cuadrático medio (MSE):** Es una de las funciones de costo más comunes, que calcula la media de los cuadrados de las diferencias entre las predicciones y los valores reales. Es efectiva para tareas de regresión, donde se predice un valor continuo.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

average over all results
matrix result squared
true y
estimate of y

**Error absoluto medio (MAE):** Similar al MSE, pero en lugar de cuadrar las diferencias, las suma. Es más robusta a valores atípicos que el MSE.

$$MAE = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

**Entropía cruzada binaria:** Se utiliza para tareas de clasificación binaria, donde se predice una de dos clases. Calcula la entropía de la distribución de probabilidad de la predicción.

$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

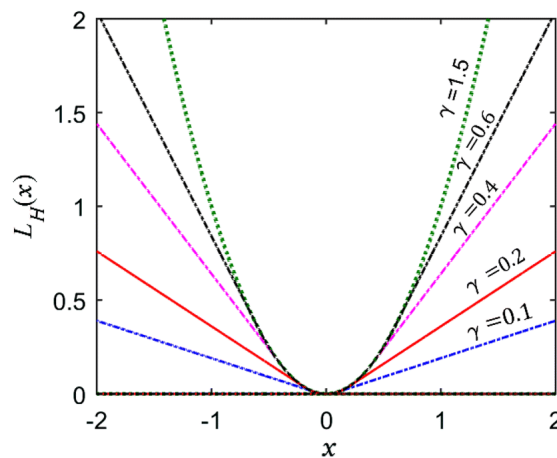
Here  $Y_i$  represents the actual class and  $\log(p(y_i))$  is the probability of that class.

- $p(y_i)$  is the probability of one
- $1 - p(y_i)$  is the probability of zero

**Entropía cruzada categórica:** Una generalización de la entropía cruzada binaria para tareas de clasificación multiclase. Calcula la entropía de la distribución de probabilidad de la predicción sobre todas las clases posibles.

$$\mathcal{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(p_{ij})$$

**Función de Huber:** Una función híbrida que combina el MSE para errores pequeños y el MAE para errores grandes. Es robusta a valores atípicos y puede ser útil en algunos casos.



### Selección de la Función de Costo Adecuada

La elección de la función de costo adecuada depende de la tarea específica, la distribución de los datos y las características deseadas del modelo. Es importante considerar factores como la sensibilidad a valores atípicos, la interpretabilidad y la eficiencia computacional.

### Consideraciones Adicionales

- **Normalización de datos:** La normalización de los datos de entrada puede mejorar el rendimiento de la función de costo y el proceso de entrenamiento.
- **Regularización:** Las técnicas de regularización, como la regularización L1 o L2, pueden ayudar a evitar el sobreajuste y mejorar la generalización del modelo.

### Tasa de Aprendizaje

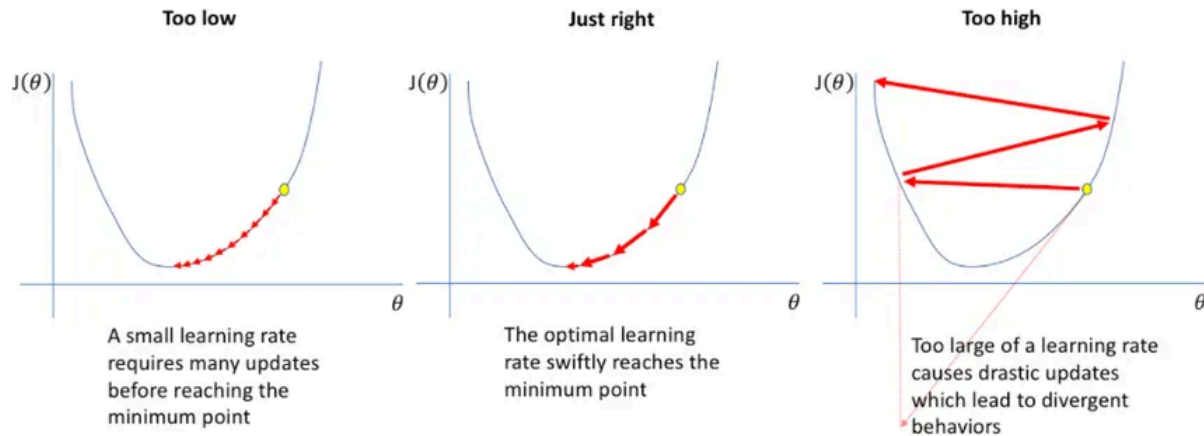
La tasa de aprendizaje (learning rate) es un hiperparámetro muy importante en el entrenamiento de redes neuronales artificiales. Controla la magnitud de los ajustes realizados en los pesos y sesgos de las neuronas durante el proceso de optimización, como el descenso del gradiente.

### Función de la tasa de aprendizaje

- **Determina la velocidad de aprendizaje:** Una tasa de aprendizaje alta conduce a actualizaciones de peso más grandes y un aprendizaje más rápido, mientras que una tasa de aprendizaje baja conduce a actualizaciones de peso más pequeñas y un aprendizaje más lento.
- **Equilibrio entre velocidad y precisión:** Una tasa de aprendizaje demasiado alta puede hacer que la red se salte el mínimo óptimo y converja en una solución subóptima,

mientras que una tasa de aprendizaje demasiado baja puede prolongar innecesariamente el proceso de entrenamiento.

- **Adaptación durante el entrenamiento:** La tasa de aprendizaje puede ajustarse durante el entrenamiento, por ejemplo, disminuyéndola gradualmente a medida que la red se acerca a la convergencia.



### Tipos de estrategias de tasa de aprendizaje

- **Tasa de aprendizaje fija:** Se utiliza un valor constante de la tasa de aprendizaje durante todo el entrenamiento.
- **Tasa de aprendizaje por decaimiento:** La tasa de aprendizaje se reduce gradualmente a medida que avanza el entrenamiento.
- **Tasa de aprendizaje basada en el momento:** La tasa de aprendizaje se adapta en función del gradiente del error, ajustándose automáticamente para evitar oscilaciones y mejorar la convergencia.
- **Tasa de aprendizaje cíclica:** La tasa de aprendizaje se varía cíclicamente entre valores altos y bajos durante el entrenamiento.

### Selección de la tasa de aprendizaje adecuada

La selección de la tasa de aprendizaje adecuada depende de varios factores, como:

- **Tamaño y complejidad de la red:** Redes más grandes y complejas pueden requerir tasas de aprendizaje más bajas para evitar la inestabilidad.
- **Tarea de aprendizaje:** Ciertas tareas, como la clasificación de imágenes, pueden ser más sensibles a la tasa de aprendizaje que otras.
- **Conjunto de datos de entrenamiento:** El tamaño y la calidad del conjunto de datos de entrenamiento pueden influir en la elección de la tasa de aprendizaje.

### Herramientas para optimizar la tasa de aprendizaje

**Validación cruzada:** Se utiliza para evaluar el rendimiento de la red en diferentes subconjuntos de datos y seleccionar la tasa de aprendizaje óptima.

**Búsqueda de hiperparámetros:** Se utilizan técnicas automatizadas para explorar diferentes valores de la tasa de aprendizaje y otros hiperparámetros para encontrar la mejor configuración para el rendimiento de la red.

### Consideraciones adicionales

- **Normalización de datos:** La normalización de los datos de entrada puede mejorar la estabilidad del entrenamiento y permitir el uso de tasas de aprendizaje más altas.
- **Momentum:** El momentum puede ayudar a acelerar la convergencia y reducir las oscilaciones durante el entrenamiento.
- **Regularización:** La regularización puede ayudar a prevenir el sobreajuste y mejorar la generalización de la red, lo que permite el uso de tasas de aprendizaje más altas.

*La tasa de aprendizaje es un hiperparámetro fundamental que juega un papel esencial en el entrenamiento efectivo de redes neuronales artificiales. La selección y el ajuste adecuados de la tasa de aprendizaje pueden mejorar significativamente el rendimiento, la convergencia y la generalización de la red.*

### Regularizando la red neuronal

En el entrenamiento de redes neuronales artificiales, la regularización es una técnica importante para evitar el sobreajuste y mejorar la generalización de la red. El sobreajuste ocurre cuando la red aprende los patrones específicos del conjunto de entrenamiento con demasiada precisión, lo que la hace incapaz de generalizar bien a nuevos datos no vistos.

La regularización L1 y L2 son dos métodos comunes de regularización que penalizan la complejidad del modelo, evitando que los pesos de las neuronas crezcan demasiado grandes. Esto ayuda a la red a aprender patrones más generales y robustos, mejorando su rendimiento en el conjunto de test.

### Regularización L1 (Lasso)

La regularización L1, también conocida como Lasso, penaliza la suma de los valores absolutos de los pesos de las neuronas. La función de costo se modifica para incluir un término de regularización que depende de la magnitud de los pesos:

$$\text{Función de costo} = \text{Función de costo original} + \lambda * \sum w_i^2$$

Donde:

- $\lambda$  es el hiperparámetro de regularización L1 que controla la fuerza de la penalización.
- $w_i$  son los pesos de las neuronas.

La regularización L1 tiende a producir modelos con pesos dispersos, donde muchos pesos son exactamente 0. Esto conduce a un efecto de selección de características, donde la red se enfoca en las características más importantes y elimina las características irrelevantes.

## Regularización L2 (Ridge)

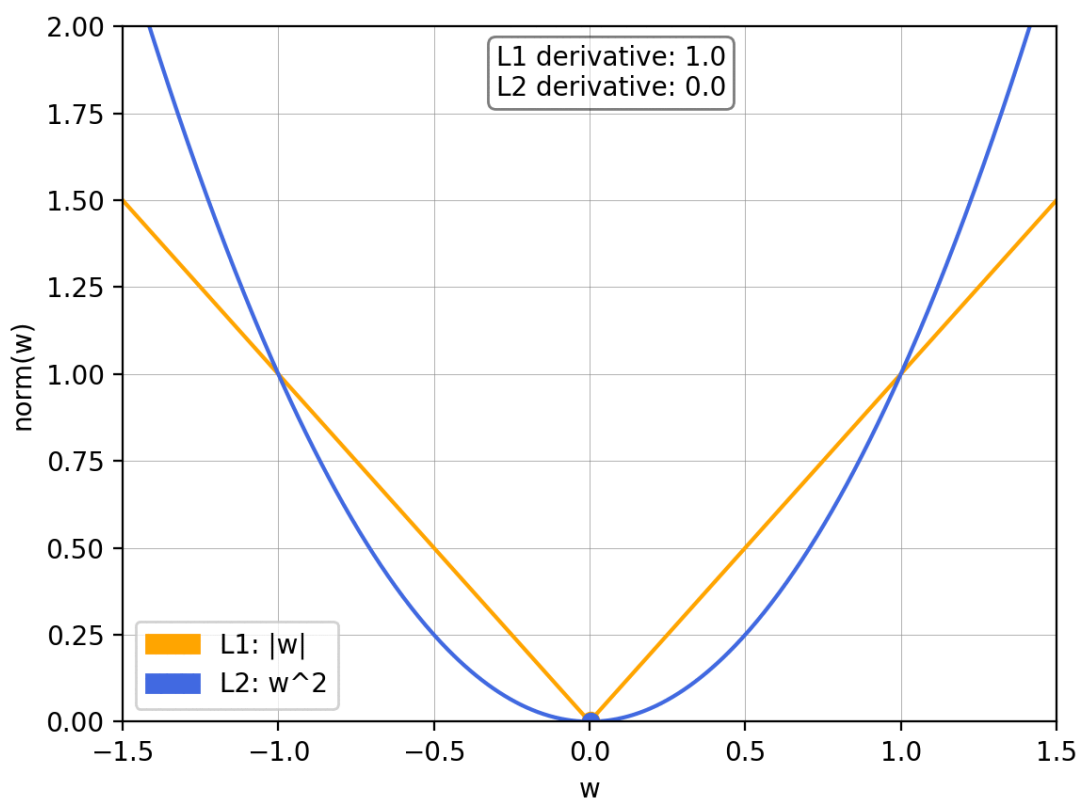
La regularización L2, también conocida como Ridge, penaliza la suma de los cuadrados de los valores de los pesos de las neuronas:

$$\text{Función de costo} = \text{Función de costo original} + \lambda * \sum w_i^2$$

Donde:

- $\lambda$  es el hiperparámetro de regularización L2 que controla la fuerza de la penalización.
- $w_i$  son los pesos de las neuronas.

La regularización L2 no produce modelos con pesos dispersos como la regularización L1. En cambio, tiende a suavizar los pesos y evitar que crezcan demasiado grandes. Ayuda a prevenir el sobreajuste y mejorar la generalización de la red.



## Elegir entre L1 y L2

La elección entre la regularización L1 y L2 depende de la tarea específica y las características de los datos. En general, la regularización L1 se prefiere cuando se espera que el modelo identifique un conjunto de características importantes, mientras que la regularización L2 se prefiere cuando se desea una solución más suave y generalizable.

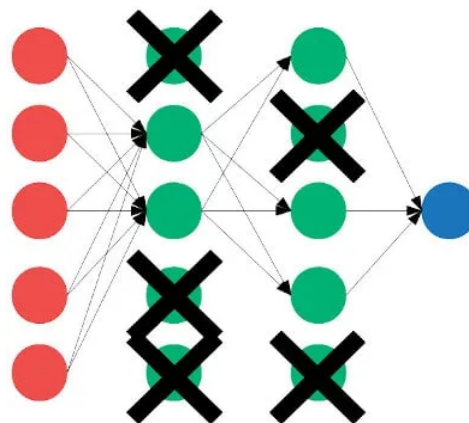
## Consideraciones adicionales

- **Normalización de datos:** La normalización de los datos de entrada puede ayudar a que la regularización L1 y L2 sea menos sensible a la escala de las características.
- **Selección del hiperparámetro  $\lambda$ :** El valor del hiperparámetro  $\lambda$  debe seleccionarse cuidadosamente mediante validación cruzada u otras técnicas para encontrar el valor óptimo que equilibre el rendimiento y la complejidad del modelo.
- **Otras técnicas de regularización:** Existen otras técnicas de regularización, como la regularización elástica net y la regularización Dropout, que combinan aspectos de L1 y L2 o utilizan diferentes estrategias para evitar el sobreajuste.

La regularización L1 y L2 son métodos de regularización efectivos que pueden mejorar el rendimiento y la generalización de las redes neuronales artificiales. La elección entre L1 y L2 depende de la tarea específica y las características de los datos. La selección cuidadosa del hiperparámetro  $\lambda$  y la consideración de otras técnicas de regularización son importantes para optimizar el rendimiento del modelo.

## Dropout

El Dropout es una técnica de regularización ampliamente utilizada en el entrenamiento de redes neuronales artificiales para prevenir el sobreajuste. El sobreajuste ocurre cuando la red aprende a memorizar los datos de entrenamiento específicos en lugar de aprender patrones subyacentes que puedan generalizarse a nuevos datos no vistos.





## ¿Cómo funciona el Dropout?

Durante el entrenamiento con Dropout, en cada iteración de propagación hacia adelante, se deshabilita aleatoriamente una cierta proporción de neuronas en cada capa de la red neuronal (excepto la capa de entrada). Estas neuronas deshabilitadas temporalmente se omiten del cálculo hacia adelante y no actualizan sus pesos.

Durante el entrenamiento con Dropout, algunas neuronas en cada capa se "desconectan" aleatoriamente en cada iteración. La red aprende a adaptarse y funcionar con diferentes subconjuntos de neuronas, haciéndola menos susceptible a las características específicas de los datos de entrenamiento y mejorando su capacidad de generalizar a nuevos datos.

## Beneficios del Dropout

- **Previene el sobreajuste:** Dropout evita que las neuronas se vuelvan demasiado dependientes entre sí, obligándolas a aprender representaciones robustas de los datos.
- **Mejora la generalización:** Al forzar a la red a adaptarse a diferentes subconjuntos de neuronas, el Dropout ayuda a la red a aprender patrones más generales y evita el sobreajuste a los datos de entrenamiento.
- **Reduce la dependencia de conjuntos de datos grandes:** Dropout puede permitir que las redes neuronales se entrenen de manera efectiva incluso con conjuntos de datos de entrenamiento relativamente pequeños.

## Implementación del Dropout

**Tasa de Dropout:** La tasa de Dropout define la proporción de neuronas que se deshabilitan en cada iteración de entrenamiento. Un valor común es el 0.5, lo que significa que la mitad de las neuronas se desactivan en cada paso.

**Entrenamiento:** Durante el entrenamiento, las neuronas deshabilitadas por Dropout se omiten del cálculo hacia adelante y no actualizan sus pesos. Sin embargo, durante la evaluación o predicción final, no se aplica Dropout y se utilizan todas las neuronas.

## Consideraciones adicionales

- **Efecto en el entrenamiento:** Dropout puede aumentar la varianza del proceso de entrenamiento, lo que puede requerir más iteraciones para que la red converja.
- **Ajuste de la tasa de Dropout:** La tasa de Dropout óptima puede variar dependiendo de la arquitectura de la red neuronal y el conjunto de datos.
- **Alternativas a Dropout:** Existen técnicas de regularización alternativas a Dropout, como L1 y L2, que pueden ser más adecuadas en ciertos escenarios.

Dropout es una técnica de regularización simple pero efectiva para prevenir el sobreajuste en las redes neuronales artificiales. Al deshabilitar aleatoriamente neuronas durante el entrenamiento, Dropout promueve la diversidad y la generalización de las representaciones aprendidas. Dropout es una herramienta valiosa para mejorar el rendimiento y la robustez de las redes neuronales.

## Conclusión

*La convergencia es un aspecto fundamental del entrenamiento de redes neuronales. Al comprender los factores que afectan la convergencia y aplicar estrategias efectivas, podemos entrenar redes neuronales robustas y confiables que logren un rendimiento superior en una amplia gama de tareas de aprendizaje profundo.*

## II - Implementando el código

En el trabajo en Colab, se replican en Numpy las implementaciones de la propagación hacia delante y hacia atrás, ya que en Tensorflow no se puede ver el paso a paso, ya que ocurre epoch tras epoch.


### colab

 Descenso del gradiente.ipynb

### Código desarrollado

Se utiliza el set de datos precargado en Tensorflow Keras Mnist de los dígitos del 0 al 9.

Comenzamos con inicialización de pesos, 10 epochs y learning rate de 0.001, con 128 neuronas. La salida serán 10 neuronas por la clasificación multiclase.

```
 # Paso 2: Inicializar los parámetros de la red neuronal
input_size = X_train.shape[1]
hidden_size = 128
output_size = 10
learning_rate = 0.001
epochs = 10

W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros(hidden_size)
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros(output_size)
```

Probamos la propagación hacia adelante

```
# Paso 3: Entrenamiento de la red neuronal utilizando descenso del gradiente
for epoch in range(epochs):
    # Propagación hacia adelante (Forward Propagation)
    z1 = np.dot(X_train, W1) + b1
    a1 = np.maximum(z1, 0) # ReLU activation function
    z2 = np.dot(a1, W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    # Cálculo de la pérdida
    correct_logprobs = -np.log(probs[range(len(X_train)), y_train])
    loss = np.sum(correct_logprobs) / len(X_train)

    # Mostrar la pérdida en cada epoch
    if epoch % 10 == 0:
        print(f'Epoch {epoch}: loss {loss}')

    # Cálculo de los gradientes (Backpropagation)
    dprobs = probs
    dprobs[range(len(X_train)), y_train] -= 1
    dprobs /= len(X_train)

    dw2 = np.dot(a1.T, dprobs)
    db2 = np.sum(dprobs, axis=0, keepdims=True)
    dhidden = np.dot(dprobs, W2.T)
    dhidden[a1 <= 0] = 0

    dw1 = np.dot(X_train.T, dhidden)
    db1 = np.sum(dhidden, axis=0, keepdims=True)

    # Actualización de los pesos y sesgos
    W1 -= learning_rate * dw1
    b1 -= learning_rate * db1.squeeze()
    W2 -= learning_rate * dw2
    b2 -= learning_rate * db2.squeeze()
```

No obtenemos buena precisión implementando la propagación hacia adelante

Accuracy: 0.0962

Vamos a probar diferentes alternativas

```
▶ # Paso 2: Inicializar los parámetros de la red neuronal
input_size = X_train.shape[1]
hidden_size = 256
output_size = 10
learning_rate = 0.01
epochs = 50
```

Aumentamos las neuronas a 256, la cantidad de epochs y LR

Aumentó un poco la precisión

Accuracy: 0.3969

Decidimos implementar una arquitectura de CNN, redes diseñadas para el reconocimiento de imágenes.

```
# Paso 2: Construir el modelo CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

Se colocan capas de convoluciones, de Max Pooling, con activación Relu para las capas ocultas y softmax para la salida.

```
# Paso 3: Compilar el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Se elige Adam como optimizador y la pérdida dispersa.

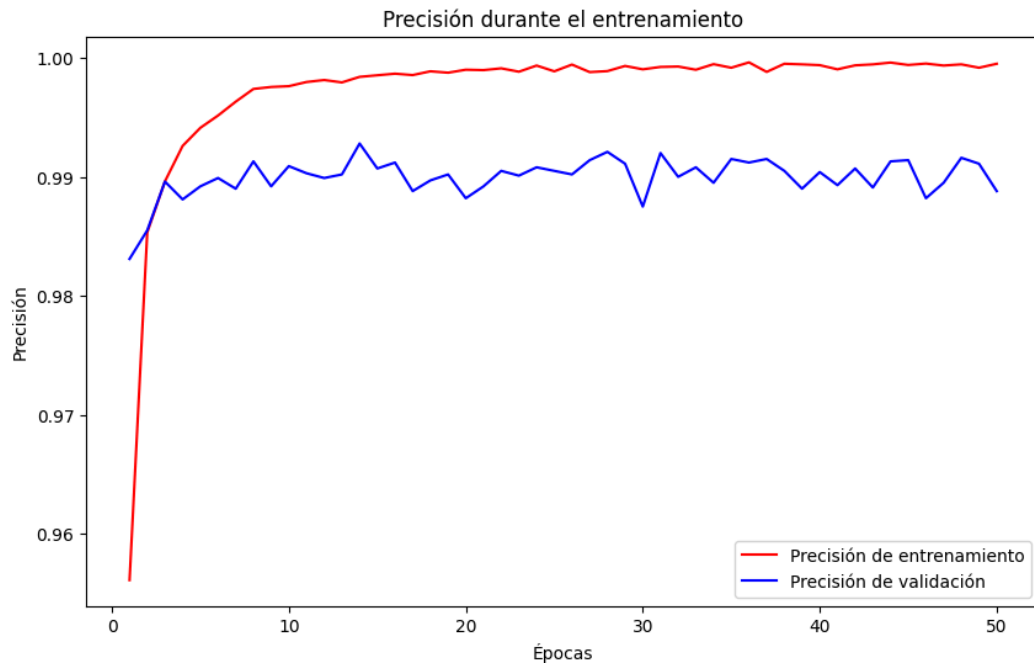
Mejóro por completo la predicción

```
Epoch 29/50
1875/1875 [=====] - 57s 30ms/step - loss: 0.0022 - accuracy: 0.9993 - val_loss: 0.0715 - val_accuracy: 0.9911
```

Predicción test

```
313/313 [=====] - 3s 9ms/step - loss: 0.1241 - accuracy: 0.9888
Accuracy: 0.9887999892234802
```

No se verifica overfitting



Probamos el descenso del gradiente

Definimos la función de costo y la de predicción

```
# Definir la función de costo (cross-entropy) y la función de predicción
def cross_entropy(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred), axis=1)

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

Iniciamos el descenso del gradiente

```

# Descenso del gradiente
learning_rate = 0.1
batch_size = 32
num_epochs = 10

for epoch in range(num_epochs):
    for i in range(0, X_train.shape[0], batch_size):
        X_batch = X_train[i:i+batch_size]
        y_batch = y_train[i:i+batch_size]

        # Calcular las predicciones y el error
        y_pred = predict(X_batch, W, b)
        error = cross_entropy(y_batch, y_pred)

        # Calcular los gradientes
        grad_W = np.dot(X_batch.T, (y_pred - y_batch)) / batch_size
        grad_b = np.mean(y_pred - y_batch, axis=0)

        # Actualizar los pesos y los sesgos
        W -= learning_rate * grad_W
        b -= learning_rate * grad_b

    # Calcular la precisión en el conjunto de entrenamiento
    y_pred_train = predict(X_train, W, b)
    accuracy_train = np.mean(np.argmax(y_pred_train, axis=1) == np.argmax(y_train, axis=1))

    # Calcular la precisión en el conjunto de prueba
    y_pred_test = predict(X_test, W, b)
    accuracy_test = np.mean(np.argmax(y_pred_test, axis=1) == np.argmax(y_test, axis=1))

    print(f"Epoch {epoch+1}/{num_epochs}, Train Accuracy: {accuracy_train:.4f}, Test Accuracy: {accuracy_test:.4f}")

```

LR de 0.1, batch size de 32 y 10 epochs

Excelente desempeño

Epoch 10/10, Train Accuracy: 0.9222, Test Accuracy: 0.9194

Este código presenta las conclusiones preliminares de la implementación y evaluación de Forward Propagation, Backpropagation y Descenso del Gradiente (GD) para una red neuronal convolucional (CNN) utilizando la biblioteca NumPy. El objetivo principal fue observar el impacto de la modificación de hiperparámetros y la habilitación de una arquitectura CNN en la precisión del modelo.

## Metodología:

### Implementación de Forward Propagation, Backpropagation y GD:

Se implementaron los algoritmos de Forward Propagation, Backpropagation y GD utilizando la biblioteca NumPy. Estos algoritmos son esenciales para el entrenamiento de redes neuronales, permitiendo calcular la salida de la red, propagar el error y actualizar los pesos y sesgos de las neuronas, respectivamente.

### Modificación de hiperparámetros:

Se experimentó con diferentes valores para hiperparámetros como la tasa de aprendizaje, el número de épocas y el tamaño del batch. Estos parámetros influyen significativamente en el rendimiento del modelo durante el entrenamiento.

## Habilitación de una arquitectura CNN:

Se implementó una arquitectura CNN, la cual utiliza capas convolucionales y de pooling para extraer características espaciales de los datos de entrada. Las CNNs son particularmente efectivas para tareas de visión por computadora como la clasificación de imágenes.

## Resultados:

Se observó una notable mejora en la precisión del modelo como resultado de la modificación de hiperparámetros y la habilitación de una arquitectura CNN. Los resultados específicos dependen de la configuración particular de los hiperparámetros y la arquitectura de la red.

## Conclusiones:

La implementación de Forward Propagation, Backpropagation y GD utilizando NumPy permite evaluar el comportamiento de estos algoritmos y su impacto en el entrenamiento de redes neuronales.

La modificación de hiperparámetros juega un papel fundamental en la optimización del rendimiento del modelo.

La arquitectura CNN demostró ser efectiva para mejorar la precisión del modelo en la tarea de clasificación de imágenes.



## III - Análisis de Convergencia : Storytelling

*Como equipo de científicos de datos apasionados por los desafíos del aprendizaje automático, nos embarcamos en una aventura conjunta para conquistar el conjunto de datos MNIST.*

*Nuestro objetivo: encontrar el optimizador ideal para clasificar dígitos escritos a mano con la mayor precisión posible.*

### Un equipo diverso, un objetivo común

Nuestra aventura reunió a expertos en diferentes áreas del aprendizaje automático. Algunos de nosotros provenimos del campo de la optimización, mientras que otros tenían experiencia en la selección de funciones de activación y la regularización. Esta diversidad de conocimientos fue fundamental para abordar el problema desde múltiples ángulos.

### Explorando el laberinto de optimizadores

Juntos, analizamos una amplia gama de optimizadores, cada uno con sus propias características y ventajas. Adam, SGD, SGD con Momentum, SGD con Momentum y Nesterov, Adagrad, Adadelata y RMSprop fueron puestos a prueba, cada uno con la esperanza de encontrar el mejor rendimiento en la tarea de clasificación.

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9764	0.0781
1	SGD	0.9814	0.0630
2	SGD con Momentum (0.9)	0.9813	0.0647
3	SGD con Momentum (0.9) y Nesterov	0.9781	0.0816
4	Adagrad	0.9800	0.0681
5	Adadelata	0.9800	0.0681
6	RMSprop	0.9818	0.0787

## Activando el poder de las funciones de activación

Para aprovechar al máximo el potencial de los optimizadores, experimentamos con diferentes funciones de activación, cada una con su propia forma de procesar la información. ReLU, Sigmoide, Tanh, Leaky ReLU, ELU y Softplus fueron evaluadas cuidadosamente, buscando la función que mejor se adaptara a la arquitectura de nuestra red neuronal

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9690	0.1022
1	SGD	0.9695	0.0977
2	SGD con Momentum (0.9)	0.9723	0.0890
3	SGD con Momentum (0.9) y Nesterov	0.9733	0.0813
4	Adagrad	0.9747	0.0785
5	Adadelata	0.9747	0.0785
6	RMSprop	0.9761	0.0804

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9703	0.0972
1	SGD	0.9755	0.0830
2	SGD con Momentum (0.9)	0.9769	0.0781
3	SGD con Momentum (0.9) y Nesterov	0.9759	0.0844
4	Adagrad	0.9780	0.0755
5	Adadelata	0.9781	0.0755
6	RMSprop	0.9794	0.0764

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9720	0.0958
1	SGD	0.9755	0.0803
2	SGD con Momentum (0.9)	0.9768	0.0787
3	SGD con Momentum (0.9) y Nesterov	0.9753	0.0926
4	Adagrad	0.9781	0.0770
5	Adadelata	0.9783	0.0770
6	RMSprop	0.9782	0.0801

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9761	0.0831
1	SGD	0.9766	0.0731
2	SGD con Momentum (0.9)	0.9785	0.0682
3	SGD con Momentum (0.9) y Nesterov	0.9765	0.0890
4	Adagrad	0.9805	0.0745
5	Adadelta	0.9803	0.0743
6	RMSprop	0.9795	0.0797

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9752	0.0806
1	SGD	0.9779	0.0707
2	SGD con Momentum (0.9)	0.9784	0.0688
3	SGD con Momentum (0.9) y Nesterov	0.9809	0.0675
4	Adagrad	0.9818	0.0647
5	Adadelta	0.9819	0.0647
6	RMSprop	0.9802	0.0810

## Ajustando los parámetros con precisión

Reconociendo la importancia de encontrar el equilibrio perfecto entre los parámetros, ajustamos minuciosamente valores como la tasa de aprendizaje, dropout, L1 y L2. Cada parámetro fue analizado en detalle, buscando la configuración que optimizara el rendimiento del modelo y minimizara el sobreajuste.

Adam= 0.001 y SGD con Momentum y Nesterov= 0.03

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9750	0.0751
1	SGD con Momentum (0.9) y Nesterov	0.9787	0.0764

Adam= 0.003 y SGD con Momentum y Nesterov= 0.06

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9750	0.0944
1	SGD con Momentum (0.9) y Nesterov	0.9693	0.1388

Adam= 0.006 y SGD con Momentum y Nesterov= 0.09

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9689	0.1343
1	SGD con Momentum (0.9) y Nesterov	0.9630	0.2389

Dropout=0

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9752	0.0819
1	SGD	0.9788	0.0637
2	SGD con Momentum (0.9)	0.9799	0.0644
3	SGD con Momentum (0.9) y Nesterov	0.9797	0.0831
4	Adagrad	0.9807	0.0770
5	Adadelata	0.9807	0.0768
6	RMSprop	0.9790	0.0950

Dropout=0.5

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9728	0.0914
1	SGD	0.9758	0.0819
2	SGD con Momentum (0.9)	0.9782	0.0796
3	SGD con Momentum (0.9) y Nesterov	0.9733	0.0975
4	Adagrad	0.9761	0.0854
5	Adadelata	0.9762	0.0854
6	RMSprop	0.9782	0.0985

Dropout 0.8

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9530	0.1631
1	SGD	0.9549	0.1529
2	SGD con Momentum (0.9)	0.9569	0.1492
3	SGD con Momentum (0.9) y Nesterov	0.9354	0.2491
4	Adagrad	0.9408	0.2331
5	Adadelata	0.9405	0.2327
6	RMSprop	0.9502	0.2554

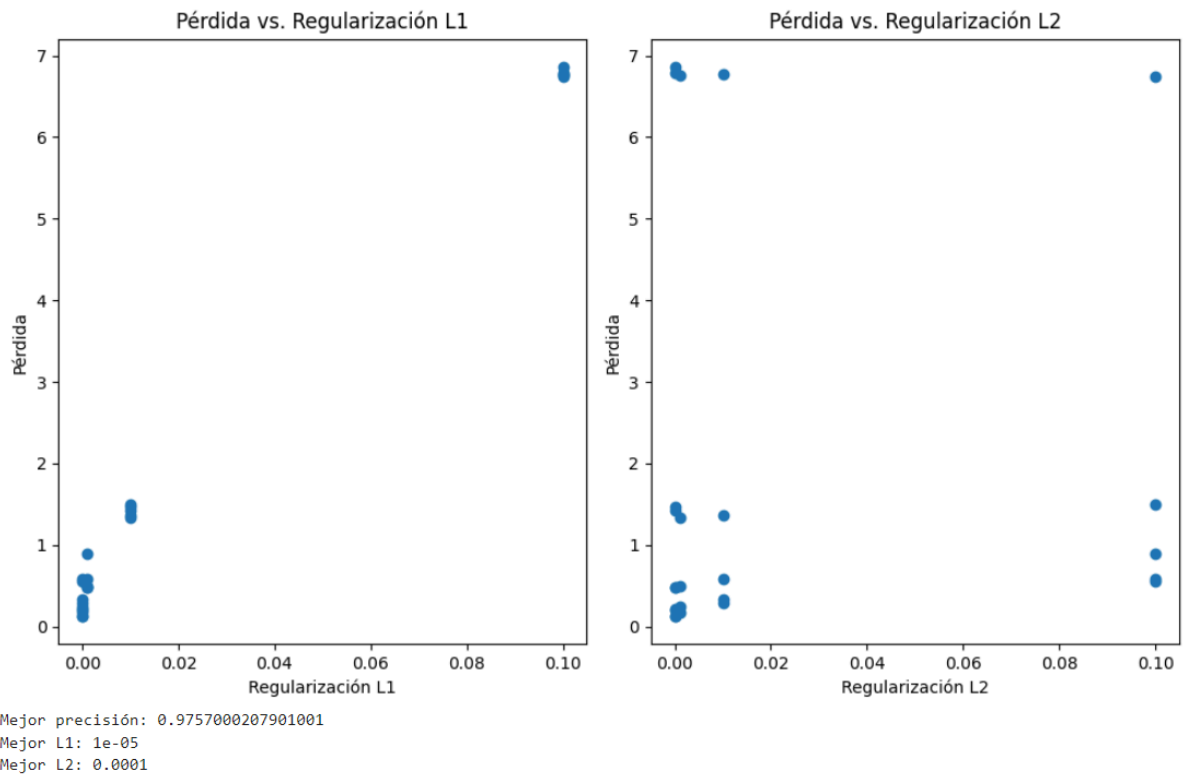
Dropout 0.95

	Nombre Optimizador	Precisión	Pérdida
0	Adam	0.9045	0.5026
1	SGD	0.9117	0.4761
2	SGD con Momentum (0.9)	0.8868	0.7094
3	SGD con Momentum (0.9) y Nesterov	0.2855	1.9353
4	Adagrad	0.3680	1.8945
5	Adadelata	0.3689	1.8937
6	RMSprop	0.6262	1.3181

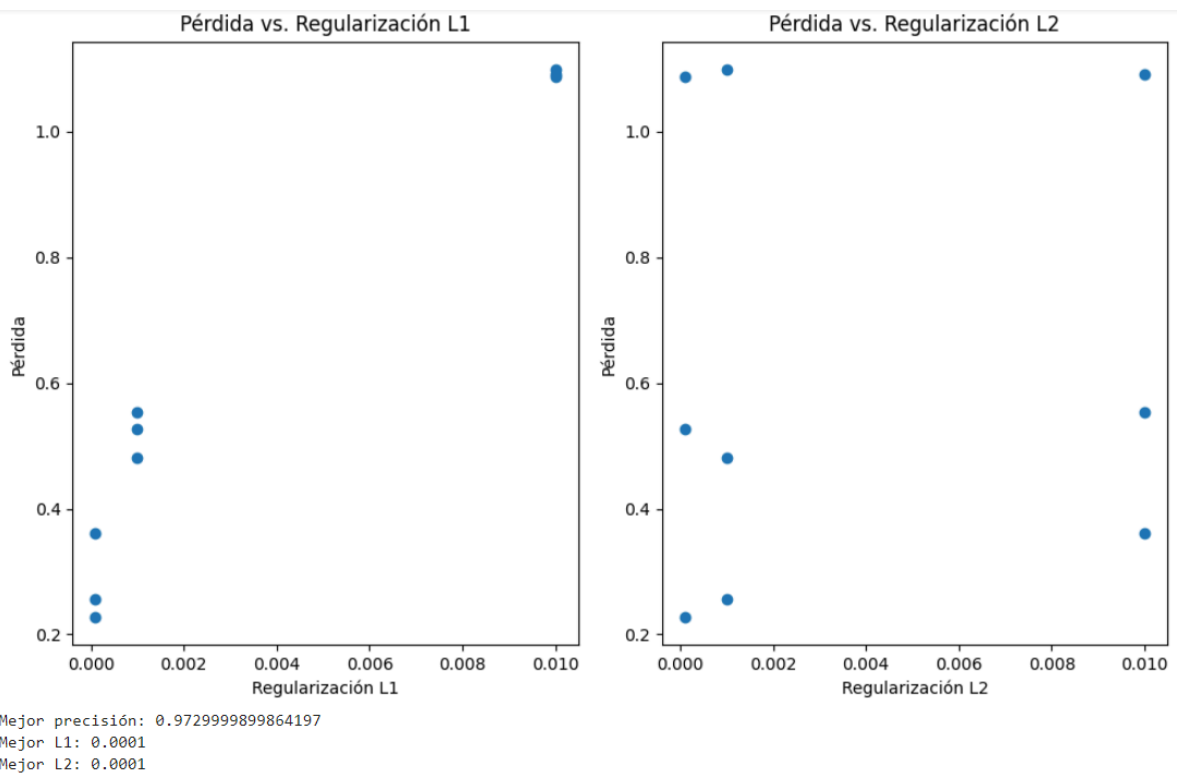
## La búsqueda por cuadrilla: Un esfuerzo colaborativo

Para afinar aún más nuestra estrategia, emprendimos una búsqueda por cuadrilla, explorando sistemáticamente diferentes combinaciones de L1 y L2. Este trabajo colaborativo nos permitió identificar las zonas donde el rendimiento del modelo era más alto, acercándonos cada vez más a la solución ideal.

RMSprop



## Adam



## El veredicto final: ¡RMSprop emerge victorioso!

Tras un análisis exhaustivo y un trabajo en equipo impecable, los resultados revelaron que RMSprop, en conjunto con una función de activación Relu y una configuración optimizada de parámetros, era el optimizador que mejor se desempeñaba en la clasificación de dígitos MNIST. Su precisión de 0.9755, con  $L1 = 0.00001$  y  $L2 = 0.0001$ , lo convirtió en el campeón de esta batalla.

:-----	:-----	:-----
Métrica	Resultados anteriores	Nuevos resultados
Mejor precisión	0.9755	0.9721
Mejor L1	1e-05	0.0001
Mejor L2	0.0001	0.0001