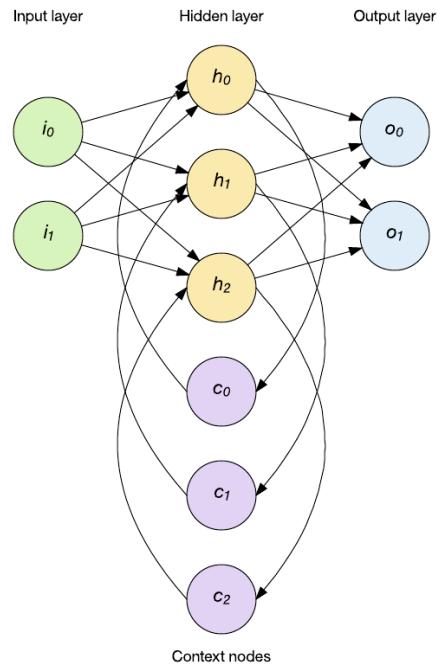


# Unidad 6.

## Redes neuronales recurrentes



“Programación de Inteligencia Artificial”  
Curso de Especialización en Inteligencia Artificial  
y Big Data

## Contenido

1. Introducción a las Redes Neuronales Recurrentes.....	3
1.1. ¿Por qué necesitamos RNN? .....	3
1.2. Aplicaciones de las RNN .....	3
2. Arquitectura y Funcionamiento de las RNR .....	4
2.1. Estructura de una RNN básica .....	5
2.2. Propagación hacia adelante en una RNN .....	6
2.3. Propagación hacia atrás a través del tiempo.....	6
2.4. Problemas del entrenamiento de RNN .....	6
2.5. Tipos de redes según entradas y salidas .....	7
3. Variantes de la celda RNN .....	9
3.1. RNN básica .....	9
3.2. Long Short-Term Memory (LSTM) .....	9
3.3. Gated Recurrent Unit (GRU).....	11
3.4. Bidirectional RNN .....	13
4. Implementación con Keras/TensorFlow .....	13
4.1. Preprocesamiento de datos para RNN .....	13
5. Casos de Uso y Aplicaciones.....	14
5.1. Generación de texto con RNN .....	14
5.2. Predicción del precio de acciones con LSTM.....	15
5.3. Análisis de sentimientos con modelos recurrentes.....	15
5.4. Aplicaciones en chatbots y asistentes virtuales .....	16
6. Análisis de series temporales .....	<b>¡Error! Marcador no definido.</b>
6. Optimización y Mejora del Rendimiento.....	<b>¡Error! Marcador no definido.</b>
7. Alternativas a las RNN y Tendencias Actuales.....	<b>¡Error! Marcador no definido.</b>
8. Proyecto final.....	<b>¡Error! Marcador no definido.</b>

# 1. Introducción a las Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN, por sus siglas en inglés) son un tipo de red neuronal diseñada para trabajar con datos secuenciales. A diferencia de las redes neuronales **feedforward**, las RNN pueden mantener una memoria interna que les permite procesar secuencias de datos de manera contextual, lo que las hace especialmente útiles para tareas como el modelado de lenguaje, la predicción de series temporales y el reconocimiento de voz.

Mientras que en las redes neuronales **feedforward** la información fluye en una única dirección, desde la capa de entrada hasta la capa de salida, sin retroalimentación. En las redes neuronales recurrentes se permiten conexiones recurrentes que crean ciclos en la red, lo que les permite almacenar información de estados anteriores. Esta memoria interna las hace ideales para tareas secuenciales, ya que cada salida depende no solo de la entrada actual, sino también del contexto previo.

## 1.1. ¿Por qué necesitamos RNN?

Existen múltiples problemas en los que la secuencia de datos es crucial para obtener resultados precisos. Algunos ejemplos incluyen:

- **Análisis de texto y modelado de lenguaje:** en una oración, el significado de una palabra depende del contexto en el que se encuentra.
- **Reconocimiento de voz:** el significado de una palabra hablada depende del contexto fonético y sintáctico.
- **Predicción de series temporales:** para prever el valor de una acción bursátil o la demanda de un producto, es necesario analizar datos pasados.
- **Traducción automática:** un modelo necesita comprender la estructura de la oración en un idioma antes de traducirlo correctamente a otro.

Las redes neuronales tradicionales no pueden manejar bien este tipo de problemas porque no tienen memoria. En contraste, las RNN pueden recordar información previa y utilizarla en su proceso de toma de decisiones, lo que las convierte en una solución ideal para este tipo de aplicaciones.

## 1.2. Aplicaciones de las RNN

Las redes neuronales recurrentes se han aplicado con éxito en una variedad de campos. Algunas de sus aplicaciones más destacadas incluyen:

- A. Procesamiento del lenguaje natural
  - Generación de texto basado en un estilo particular (por ejemplo, escribir un poema o continuar una historia).

- Autocompletado y predicción de palabras en teclados inteligentes.
- B. Análisis y predicción de series temporales
- Pronóstico de ventas y demanda en empresas.
  - Predicción del clima y patrones meteorológicos.
  - Predicción de precios de activos financieros.
  - Aspectos médicos (ver apartado G)
- C. Reconocimiento de voz
- Conversión a texto en asistentes como Siri o Google Assistant.
  - Transcripción automática de audios en reuniones o conferencias.
- D. Generación de texto
- Creación de subtítulos automáticos en videos.
  - Generación automática de noticias o resúmenes de texto.
- E. Traducción automática: sistemas como Google Translate utilizan redes recurrentes para mejorar la precisión en traducciones automáticas.
- F. Análisis de sentimientos: clasificación de opiniones en redes sociales o reseñas de productos para determinar si son positivas o negativas.
- G. Aplicaciones específicas en medicina
- Análisis de electrocardiogramas (ECG) para detectar patrones anormales en los latidos del corazón.
  - Predicción de enfermedades basadas en datos históricos del paciente.

## 2. Arquitectura y Funcionamiento de las RNR

Las **Redes Neuronales Recurrentes (RNN, Recurrent Neural Networks)** son un tipo de red neuronal especializada en trabajar con **datos secuenciales**. A diferencia de las redes neuronales tradicionales (feedforward), las RNN tienen una estructura que permite que la información fluya de forma recurrente dentro de la red, lo que les permite recordar estados anteriores y usarlos para predecir la siguiente salida.

Este tipo de red es especialmente útil en tareas donde el orden de los datos es importante, como el **procesamiento de lenguaje natural (NLP)**, el **reconocimiento de voz** y la **predicción de series temporales**.

## 2.1. Estructura de una RNN básica

La estructura básica de una **Red Neuronal Recurrente (RNN)** es similar a la de una red neuronal tradicional, con la diferencia de que cada neurona no solo recibe la entrada actual, sino también un **estado previo**, que actúa como una memoria de lo que ha ocurrido antes.

**Componentes principales de una RNN:**

1. **Capa de entrada:** recibe los datos secuenciales (pueden ser palabras, valores numéricos, etc.).
2. **Capa oculta recurrente:** Contiene una o más neuronas con conexiones recurrentes que permiten almacenar información de pasos anteriores.
3. **Capa de salida:** Genera la predicción basada en el estado actual de la red.

**Representación matemática de una RNN:**

En una RNN simple, la activación de la neurona en el tiempo  $t$  se define como:

$$h_t = f(W_x x_t + W_h h_{t-1} + b)$$

Donde:

- $x_t$  es la entrada en el tiempo  $t$ .
- $h_t$  es el estado oculto en el tiempo  $t$ .
- $W_x$  es la matriz de pesos que conecta la entrada con la capa oculta.
- $W_h$  es la matriz de pesos recurrente que conecta el estado oculto anterior  $h_{t-1}$  con el actual.
- $b$  es el sesgo (bias).
- $f$  es la función de activación, generalmente **tanh** o **ReLU**.

La salida  $y_t$  en cada instante de tiempo se obtiene aplicando una transformación al estado oculto actual:

$$y_t = g(W_y h_t + b_y)$$

Donde  $W_y$  es la matriz de pesos que conecta la capa oculta con la salida y  $g$  suele ser una función de activación como softmax (para clasificación) o una función lineal (para regresión).

## 2.2. Propagación hacia adelante en una RNN

El proceso de **propagación hacia adelante (forward pass)** en una RNN ocurre de la siguiente manera:

1. Se recibe la primera entrada  $x_1$  y se calcula el primer estado oculto  $h_1$ .
2. Se utiliza  $h_1$  junto con la siguiente entrada  $x_2$  para calcular  $h_2$ .
3. Este proceso se repite hasta llegar al último estado oculto  $h_T$ .
4. Cada estado oculto puede generar una salida intermedia  $y_t$ , o bien solo la última salida  $y_T$  se utiliza como resultado final.

### Ejemplo práctico:

En un sistema de predicción de texto, si la secuencia de entrada es:

"Hoy hace buen..."

La RNN procesará cada palabra en orden, recordando las anteriores, y generará una predicción para la siguiente palabra:

"Hoy hace buen **tiempo**."

## 2.3. Propagación hacia atrás a través del tiempo

El **entrenamiento** de una RNN sigue el mismo principio de una red neuronal estándar: se minimiza una función de pérdida ajustando los pesos mediante **descenso de gradiente**. Sin embargo, debido a la naturaleza secuencial de las RNN, se usa una versión especial del algoritmo de **backpropagation** llamada **Backpropagation Through Time (BPTT)**.

### Funcionamiento del BPTT:

1. Se calcula la **función de pérdida** después de recorrer toda la secuencia.
2. Se retropropagan los errores **hacia atrás en el tiempo**, desde el último estado oculto  $h_T$  hasta el primero  $h_1$ .
3. Se ajustan los pesos  $W_h W_x$  y  $W_y$  usando el descenso de gradiente.

## 2.4. Problemas del entrenamiento de RNN

Los problemas habituales en el entrenamiento de redes recurrentes simples pueden ser 2:

### 1. Desvanecimiento del gradiente (Vanishing Gradient)

Cuando se usan funciones de activación como **tanh** o **sigmoid**, los gradientes pueden volverse muy pequeños con cada iteración. Esto hace que

los pesos apenas se actualicen, causando que la red **olvide información a largo plazo**. Para resolverlo se emplean 2 enfoques:

- Usar arquitecturas como **LSTM** o **GRU**, que mitigan este problema gracias a sus puertas de memoria.
- Cambiar la función de activación a **ReLU**, que es menos propensa a este problema.

## 2. Explosión del Gradiente (Exploding Gradient)

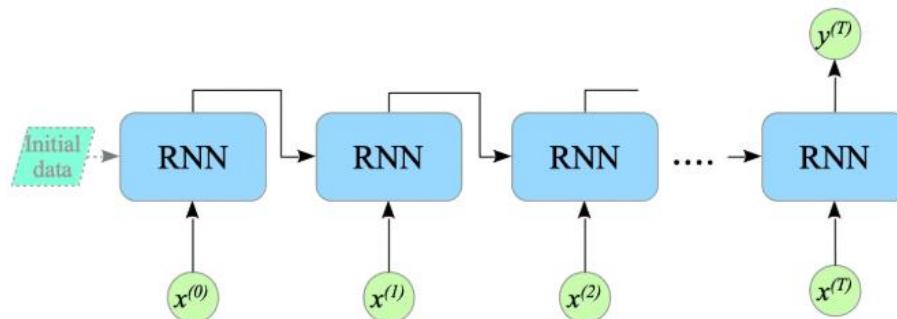
Ocurre cuando los gradientes crecen exponencialmente durante la retropropagación, causando inestabilidad en el entrenamiento. Las posibles soluciones son:

- Aplicar **clipping** al gradiente, limitando su magnitud.
- Usar **regularización** para evitar pesos demasiado grandes.

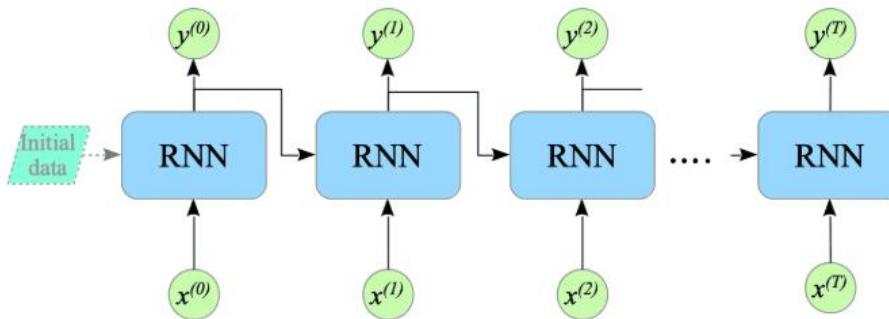
### 2.5. Tipos de redes según entradas y salidas

Las **Redes Neuronales Recurrentes (RNN)** pueden adoptar diferentes arquitecturas dependiendo del formato de sus entradas y salidas. En general, podemos clasificar estas arquitecturas en 3 categorías principales:

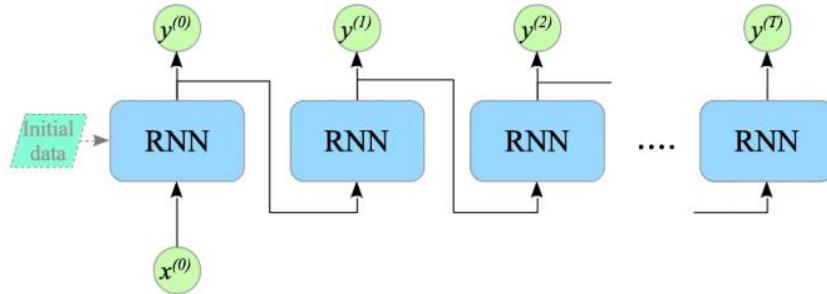
(1) Many-to-One



(2) Many-to-Many



### (3) One-to-Many



### (1) Many-to-One

Esta arquitectura toma una secuencia como entrada y genera otra secuencia como salida. Se usa cuando el número de elementos de entrada y salida pueden ser distintos. Ejemplos: traducción automática (ejemplo: inglés → español) y generación de subtítulos en tiempo real.

### (2) Many-to-Many

Este modelo toma una **secuencia completa** como entrada y genera una única salida. Es útil cuando la clasificación depende del contexto total de la secuencia. Ejemplos: análisis de sentimientos en textos y clasificación de spam en correos electrónicos.

### (3) One-to-Many

Este tipo de arquitectura toma una única entrada y genera múltiples salidas en una secuencia. Se utiliza en tareas donde una sola señal inicial puede dar lugar a una serie de eventos. Ejemplo: generación de texto o música a partir de una nota o palabra inicial.

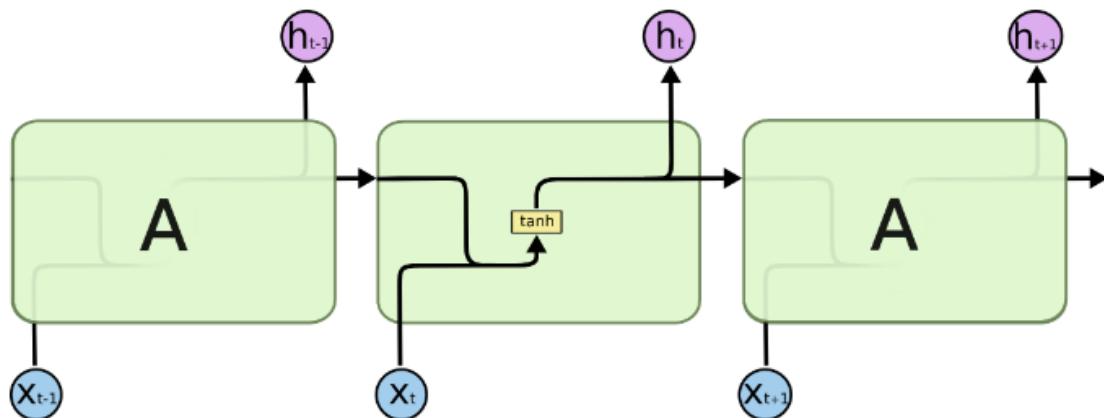
### 3. Variantes de la celda RNN

A pesar de que las Redes Neuronales Recurrentes básicas (RNN) son poderosas para modelar datos secuenciales, presentan limitaciones que han llevado al desarrollo de variantes más avanzadas. En este apartado analizaremos las principales modificaciones y arquitecturas derivadas de las RNN básicas.

#### 3.1. RNN básica

Todas las redes neuronales recurrentes tienen la forma de una cadena de módulos repetitivos. En las redes neuronales recurrentes estándar, este módulo repetitivo tendrá una estructura muy simple, con una sola capa tanh.

- En cada paso de tiempo, la celda recibe una entrada  $x$  (el dato actual) y el estado oculto previo  $h_{t-1}$  (la "memoria" del paso anterior).
- Estos se combinan mediante una transformación lineal con pesos  $W$  (para la entrada) y  $U$  (para el estado oculto), más un sesgo  $b$ .
- La salida se pasa por una función de activación no lineal, típicamente tanh, para producir el nuevo estado oculto  $h_t$ .
- Este  $h_t$  se usa como salida de la celda y se pasa al siguiente paso de tiempo.



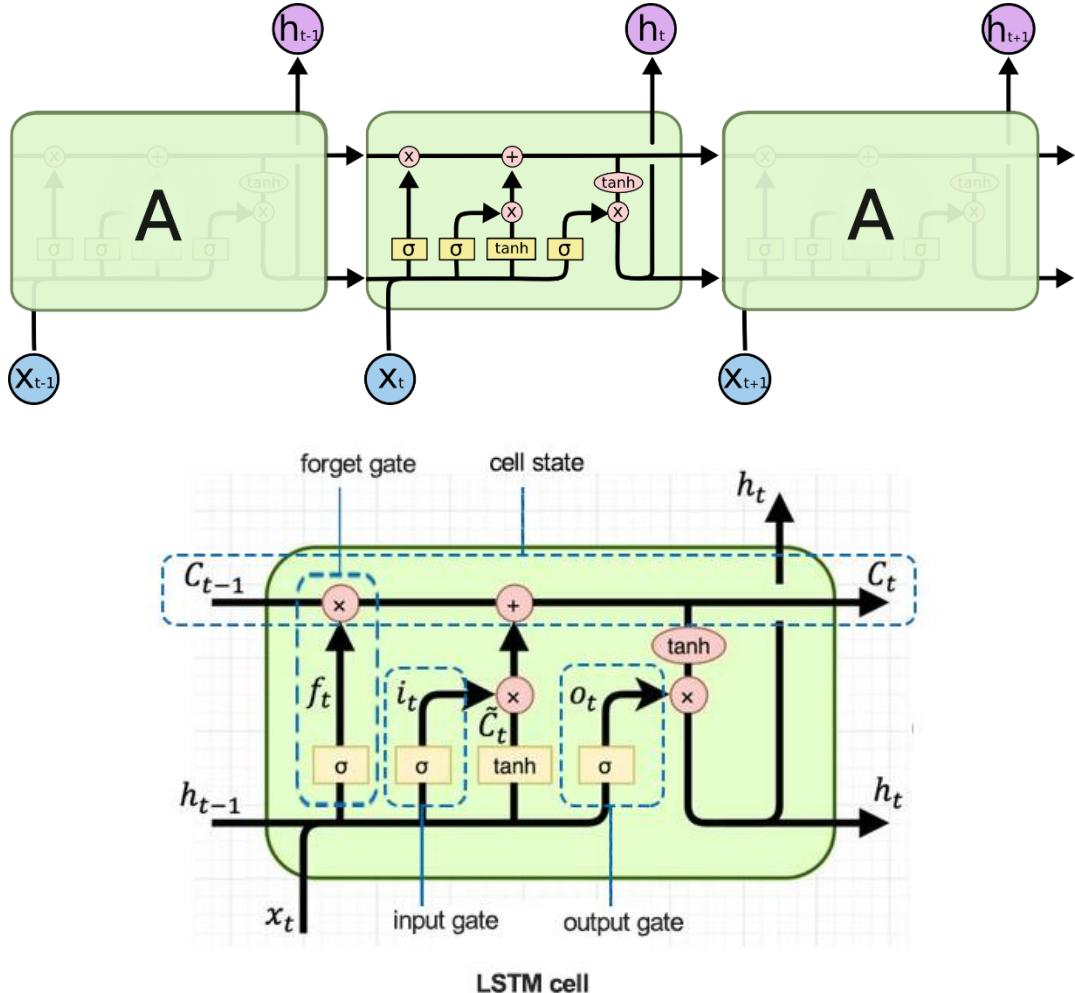
Aunque simple, la RNN estándar sufre del problema de desvanecimiento del gradiente, lo que dificulta aprender dependencias a largo plazo, ya que los gradientes se vuelven muy pequeños durante la retropropagación.

#### 3.2. Long Short-Term Memory (LSTM)

Las redes neuronales Long Short-Term Memory (LSTM) constituyen un tipo especializado de RNNs diseñadas para superar las limitaciones asociadas con la captura de dependencias temporales a largo plazo. A diferencia de las RNN tradicionales, las celdas LSTMs incorporan una arquitectura más compleja,

introduciendo unidades de memoria y mecanismos de puertas para mejorar la gestión de la información a lo largo del tiempo.

Los LSTM también tienen la estructura de cadena que hemos visto en la versión simple de RNN, pero el módulo repetitivo tiene una estructura diferente. En lugar de tener una sola capa de red neuronal, hay cuatro que interactúan de una manera muy especial.



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = \tanh(C_t) * o_t$$

## Estructura de las LSTMs

Las LSTMs presentan una estructura modular que consta de tres puertas (*gates*) fundamentales: la puerta de olvido (*forget gate*), la puerta de entrada (*input gate*), y la puerta de salida (*output gate*). Estas puertas trabajan en conjunto para regular el flujo de información a través de la unidad de memoria, permitiendo un control más preciso sobre qué información retener y cuál olvidar. Veamos sus componentes más relevantes:

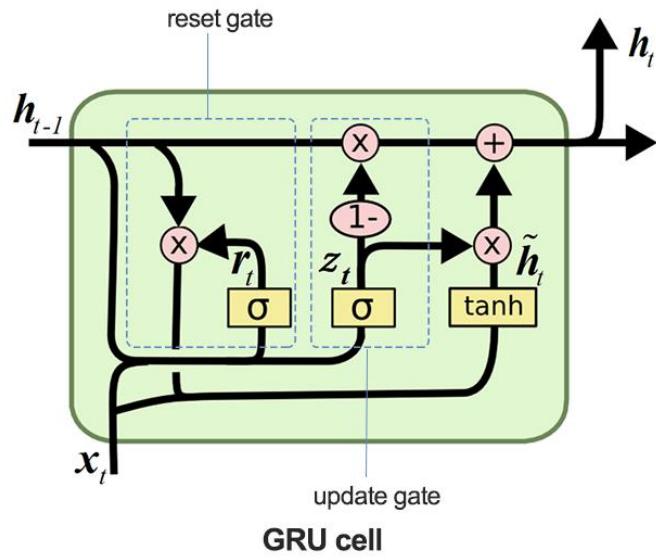
- **Puerta de olvido (*Forget Gate, ft*):** la puerta de olvido determina qué información del estado anterior de la celda debe transferirse. Emite un número entre 0 y 1 para cada número del estado de la celda  $C_{t-1}$ , donde 0 significa olvido completo y 1 significa retención completa.
- **Puerta de entrada (*Input Gate, it*):** esta puerta decide qué valores de la nueva entrada se utilizan para actualizar el estado de la celda. Regula el flujo de nueva información hacia la celda.
- **Puerta de salida (*Output Gate*):** esta puerta determina la salida final para el estado actual de la celda. Decide qué partes del estado de la celda deben salir en función de la entrada  $x_t$  y el estado oculto anterior  $h_{t-1}$ .
- **Memoria candidata ( $\tilde{C}_t$ ):** este componente genera los nuevos valores candidatos que se pueden agregar al estado de la celda. Utiliza la función de activación tanh por lo que los valores están entre -1 y 1.
- **Actualización del estado de la celda ( $C_t$ ):** el estado de la celda se actualiza combinando el estado de la celda anterior y los valores candidatos. La salida de la puerta de olvido controla la contribución del estado de la celda anterior, y la salida de la puerta de entrada controla la contribución de los nuevos valores candidatos.
- **Actualización del estado oculto ( $h_t$ ):** el estado oculto se actualiza según el estado de la celda y la decisión de la puerta de salida. Se utiliza como salida para el paso de tiempo actual y como entrada para el siguiente paso de tiempo.

### 3.3. Gated Recurrent Unit (GRU)

La GRU (Unidad Recurrente con Puertas) es una variante simplificada de la LSTM, con menos parámetros, pero aún efectiva para capturar dependencias temporales. GRU utiliza menos memoria y es más rápido que LSTM, sin embargo, LSTM es más preciso cuando se utilizan conjuntos de datos con secuencias más largas.

Veamos los componentes más destacados:

- **Puerta de actualización:** controla qué parte del estado oculto anterior  $h_{t-1}$  debe trasladarse al siguiente paso temporal. Decide efectivamente el equilibrio entre mantener la información antigua e incorporar información nueva.
- **Puerta de reinicio:** determina cuánto del estado oculto anterior debe olvidarse antes de calcular la nueva activación candidata. Permite que el modelo elimine información irrelevante del pasado.
- **Activación candidata:** genera nuevos valores potenciales para el estado oculto que pueden incorporarse en función de la decisión de la puerta de actualización.
- **Actualización del estado oculto ( $h_t$ ):** se actualiza combinando el estado oculto anterior y el estado oculto candidato. La puerta de actualización controla esta combinación, asegurando que se retiene la información relevante del pasado a la vez que se incorpora nueva información.



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

### 3.4. Bidirectional RNN

Las Redes Neuronales Recurrentes Bidireccionales (BiRNN, Bidirectional RNN) mejoran la capacidad de las RNN tradicionales al procesar la información en dos direcciones:

1. Hacia adelante (del pasado al futuro).
2. Hacia atrás (del futuro al pasado).

Esto permite que la red tenga contexto completo antes de generar una predicción, lo cual es especialmente útil en tareas como el procesamiento de lenguaje natural (NLP), donde una palabra puede depender de las anteriores y de las siguientes.

Una BiRNN está compuesta por dos capas recurrentes:

- Una capa recurrente **hacia adelante** con estados ocultos  $\vec{h}_t$
- Una capa recurrente **hacia atrás** con estados ocultos  $\overleftarrow{h}_t$

La salida final en cada paso de tiempo se obtiene combinando ambos estados ocultos:

$$h_t = [\vec{h}_t, \overleftarrow{h}_t]$$

#### Ventajas

- Captura dependencias a largo plazo en ambas direcciones.
- Mejora la precisión en tareas de procesamiento de lenguaje y reconocimiento de voz.

#### Desventajas

- Mayor costo computacional debido a la duplicación de parámetros.
- No es útil en problemas donde la predicción depende exclusivamente de datos pasados (ejemplo: predicción de series temporales).

## 4. Implementación con Keras/TensorFlow

### 4.1. Preprocesamiento de datos para RNN

Antes de entrenar una RNN, debemos preparar los datos de forma adecuada. Existen dos tipos de datos recurrentes más comunes:

- **Texto:** se requiere tokenización y padding.
- **Series temporales:** necesitan normalización y reformato en secuencias.

## Tokenización y padding para texto

Cuando trabajamos con texto, no podemos alimentar palabras directamente a una red neuronal, por lo que debemos convertirlas en números. Para ello, utilizamos Tokenización, que asigna un índice numérico a cada palabra.

Una vez tokenizado el texto, las frases pueden tener distintas longitudes, lo que puede causar problemas en una RNN. El padding nos ayuda a estandarizar las secuencias agregando ceros al inicio o final.

## Normalización de series temporales

Cuando usamos RNN para procesar series temporales, como precios de acciones o datos meteorológicos, es esencial normalizar los valores para mejorar la estabilidad del entrenamiento.

4.2. Construcción de una RNN simple en Keras

4.3. Implementación de una LSTM para predicción de series temporales

4.4. Uso de una GRU para modelado de lenguaje

4.5. Visualización de la evolución de la pérdida y la precisión

## 5. Casos de Uso y Aplicaciones

Las Redes Neuronales Recurrentes (RNN), incluyendo LSTM y GRU, tienen múltiples aplicaciones en el mundo real. Su capacidad para modelar secuencias las hace ideales para tareas en las que el contexto previo es relevante. En este apartado, exploraremos algunos casos de uso destacados.

### 5.1. Generación de texto con RNN

La generación de texto con RNN se basa en entrenar un modelo para que aprenda patrones en secuencias de texto y sea capaz de predecir la siguiente palabra o carácter en función del contexto previo.

Este enfoque se usa en:

- **Generación de escritura creativa** (poesía, cuentos).
- **Asistentes de autocompletado** en editores de texto.
- **Traducción automática** y generación de resúmenes.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding

# Definir el modelo de generación de texto
model = Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    LSTM(128, return_sequences=True),
    Dense(128, activation='relu'),
    Dense(vocab_size)
])
```

```

        Embedding(input_dim=5000, output_dim=128, input_length=50),
        LSTM(128, return_sequences=True),
        LSTM(128),
        Dense(5000, activation="softmax")
    ])

model.compile(loss="categorical_crossentropy", optimizer="adam")
model.summary()

```

## 5.2. Predicción del precio de acciones con LSTM

LSTM es ampliamente utilizada en **predicción de series temporales**, como precios de acciones o ventas. Su capacidad para capturar dependencias a largo plazo la hace ideal para analizar tendencias en datos financieros.

### Aplicaciones:

- Predicción del **precio de acciones** en la bolsa.
- Estimación de **tendencias de mercado**.
- Predicción de **demandas de productos**.

```

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Crear el modelo LSTM para series temporales
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(10, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(optimizer="adam", loss="mse")
model.summary()

```

## 5.3. Análisis de sentimientos con modelos recurrentes

El análisis de sentimientos consiste en entrenar un modelo para clasificar textos en categorías como **positivo, negativo o neutral**. Las RNN (especialmente LSTM y GRU) son útiles porque pueden capturar la estructura y el contexto de las frases.

### Aplicaciones:

- Análisis de **opiniones de clientes** en redes sociales.
- Clasificación de **reseñas** en plataformas como Amazon o TripAdvisor.
- Monitorización de **sentimientos en mercados financieros**.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense

# Modelo de análisis de sentimientos con LSTM
model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=100),
    LSTM(64, return_sequences=True),
    LSTM(64),
    Dense(3, activation="softmax") # 3 clases: positivo, negativo, neutral
])

model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=[ "accuracy"])
model.summary()
```

#### 5.4. Aplicaciones en chatbots y asistentes virtuales

Los chatbots y asistentes virtuales usan modelos recurrentes para **entender el contexto de una conversación** y responder de manera coherente.

##### Aplicaciones:

- **Atención al cliente** (Amazon Alexa, Google Assistant).
- **Automatización en empresas** (chatbots en páginas web).
- **Soporte técnico automatizado**.