

Inteligencia Artificial Aplicada para la Economía

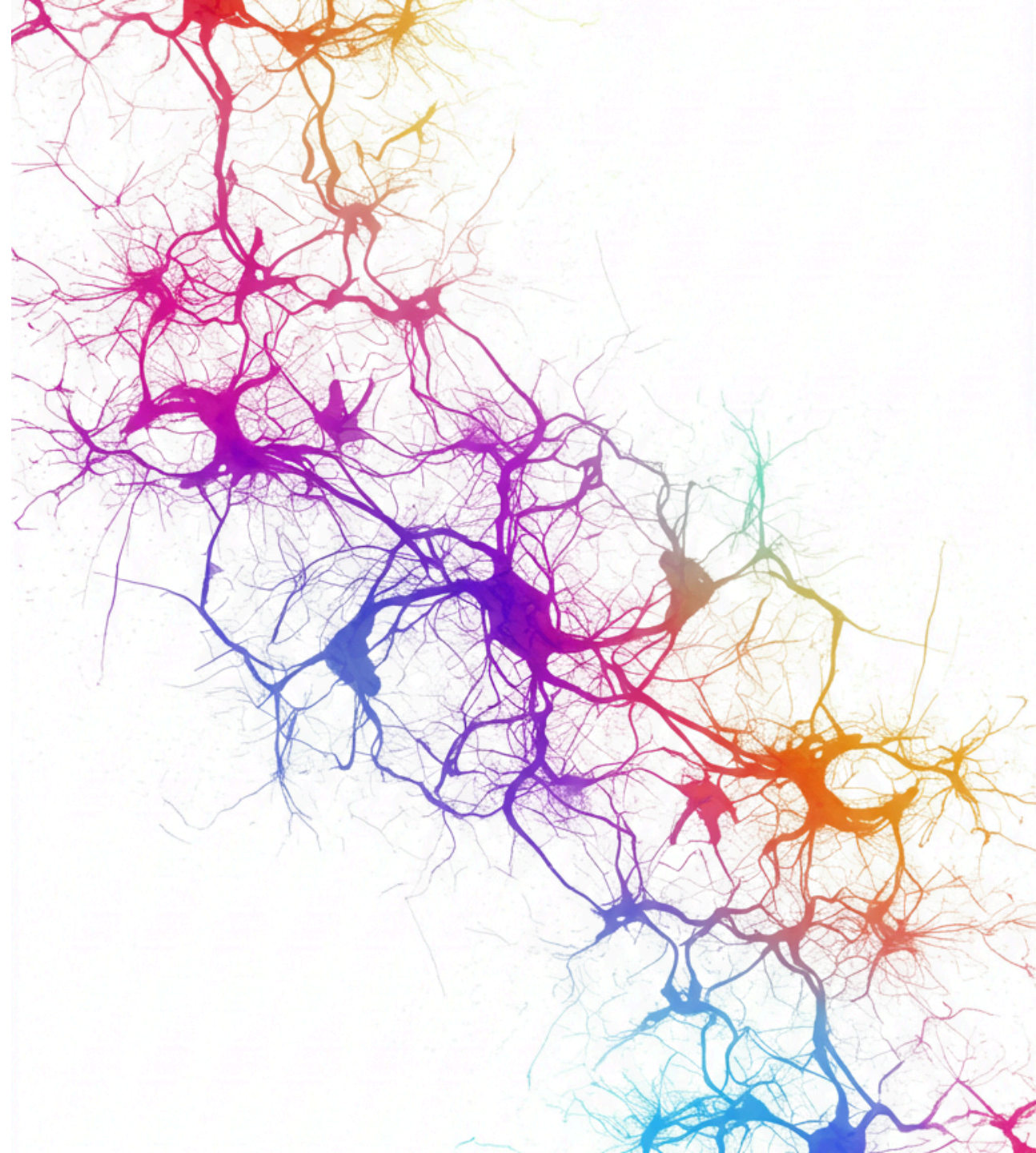
Profesores

Profesor Magistral

Camilo Vega Barbosa

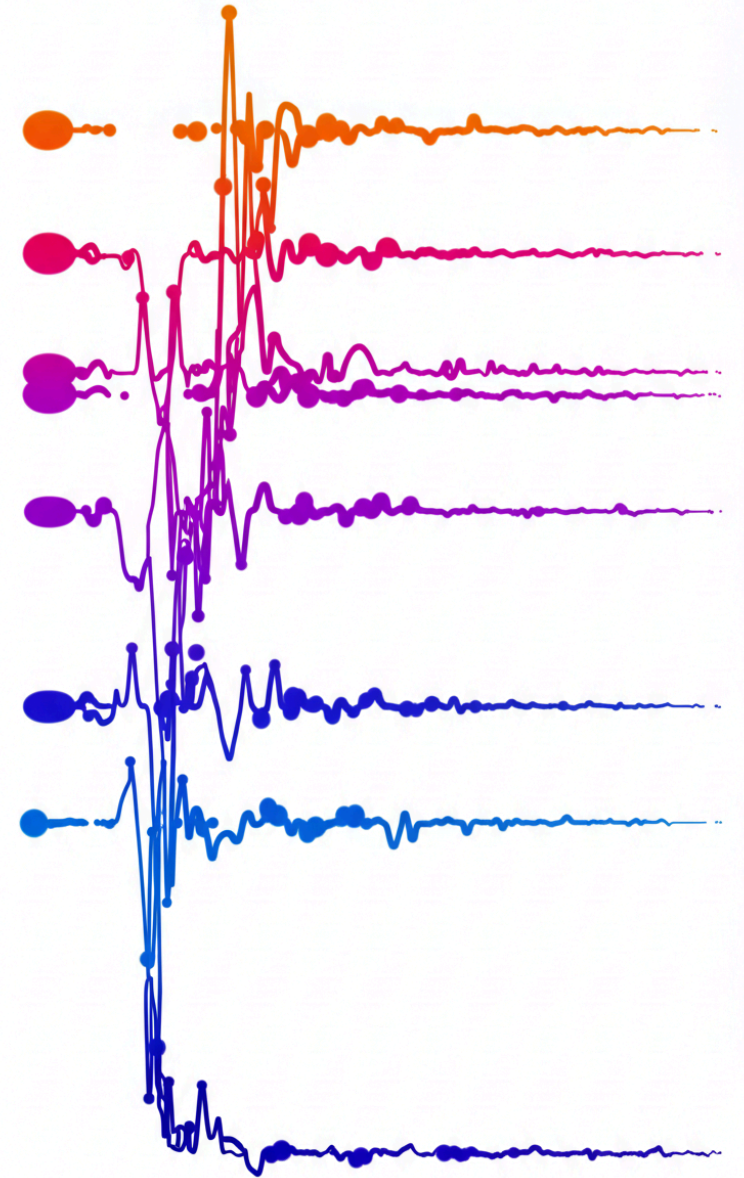
Asistente de Docencia

Sergio Julian Zona Moreno



Redes Neuronales Recurrentes (RNN)





Modelado de Series de Tiempo




Introducción a las Series Temporales

¿Por qué importa la estructura de los datos?

Antes de presentar los modelos, necesitamos entender **cómo varían los datos en el tiempo y entre entidades**:

-  **Corte transversal**: muchas entidades, un solo momento.
-  **Serie de tiempo**: una entidad, muchos momentos.
-  **Datos combinados**: múltiples entidades en distintos momentos, sin orden claro.
-  **Panel**: múltiples entidades a lo largo del tiempo, con estructura ordenada.

 Series de tiempo, combinados y panel son ejemplos de **datos longitudinales**: siguen a las entidades en el tiempo.

Repasemos: Corte Transversal

- Observaciones de muchas entidades en **un solo punto en el tiempo**.


País	Ingreso (USD)
Perú	8,900
México	10,200
Colombia	9,500
Chile	10,800

Repasemos: Serie de Tiempo

- Observaciones de una sola entidad a lo largo del tiempo.

Año	Ingreso (Perú)
2020	8,000
2021	8,500
2022	8,900
2023	9,200




Repasemos: Datos Combinados / Corte Transversal Cruzada

- Múltiples entidades medidas en distintos momentos, pero sin estructura fija.
-  No todos los años tienen la misma cantidad de observaciones por entidad.

País	Año	Ingreso (USD)
México	2020	9,800
Perú	2020	8,000
Colombia	2021	9,500
Perú	2021	8,900
México	2021	10,200
Chile	2022	10,500



Repasemos: Datos Panel

- Los **datos panel** siguen a múltiples entidades en distintos momentos del tiempo.
- Pueden representarse en dos formatos:
 -  **Formato ancho:** cada año en una columna.
 -  **Formato largo:** una fila por entidad y año.
-  Ambos formatos contienen la misma información. El largo es más flexible para análisis y **modelos**.



Formato ancho

País	2020	2021	2022	2023
Perú	8,000	8,500	8,900	9,200
México	9,500	9,800	10,200	10,600
Colombia	9,000	9,200	9,500	9,700
Chile	10,000	10,300	10,800	11,000

=



Formato largo

País	Año	Ingreso (USD)
Perú	2020	8,000
Perú	2021	8,500
Perú	2022	8,900
México	2020	9,500
México	2021	9,800
México	2022	10,200



Mismo contenido, distinto formato. El largo facilita el procesamiento.

Introducción a las Series Temporales en IA

Las series temporales son secuencias de datos organizadas cronológicamente, que plantean desafíos particulares para los modelos tradicionales de redes neuronales.

¿Por qué se requieren modelos especiales?

Las redes neuronales tradicionales (*feedforward*) no están diseñadas para capturar relaciones que dependen del tiempo.

⚙️ ¿Qué es un proceso (en series de tiempo)?

Un **proceso** es el mecanismo que genera los valores de una serie a lo largo del tiempo.

Puede ser de dos tipos:

- **Determinístico**: siempre genera los mismos valores.
- **Estocástico**: incluye componentes aleatorios. Cada valor X_t es una variable aleatoria.

Cuando observamos una secuencia de valores generados por ese proceso, obtenemos una **serie temporal**:

$$\{X_1, X_2, \dots, X_t\}$$

El proceso estocástico puede tener **estructura** (como tendencia, estacionalidad o autocorrelación) o ser completamente aleatorio (ruido blanco).



¿Cómo modelamos una serie de tiempo?

Una serie de tiempo es una **realización de un proceso estocástico**.

Un modelo simple es:

$$X_t = \mu + \varepsilon_t$$

- μ es la media o el **valor esperado** de la serie.
- ε_t representa el **error**, que puede ser completamente aleatorio o seguir algún patrón.

Este modelo asume que los datos fluctúan en torno a un nivel central determinado por el proceso generador.



Modelar una serie implica entender su estructura y separar lo predecible del error.

Componentes de una serie temporal

- **Tendencia (T_t):** cambio sostenido en el tiempo.
→ Ej: crecimiento poblacional, inflación acumulada.
- **Estacionalidad (S_t):** patrón que se repite en intervalos regulares.
→ Ej: aumento de ventas en diciembre, más búsquedas de vuelos en julio.
- **Error (ε_t):** variación no explicada por los anteriores.
→ Ej: factores externos impredecibles como una huelga o una ola de calor.

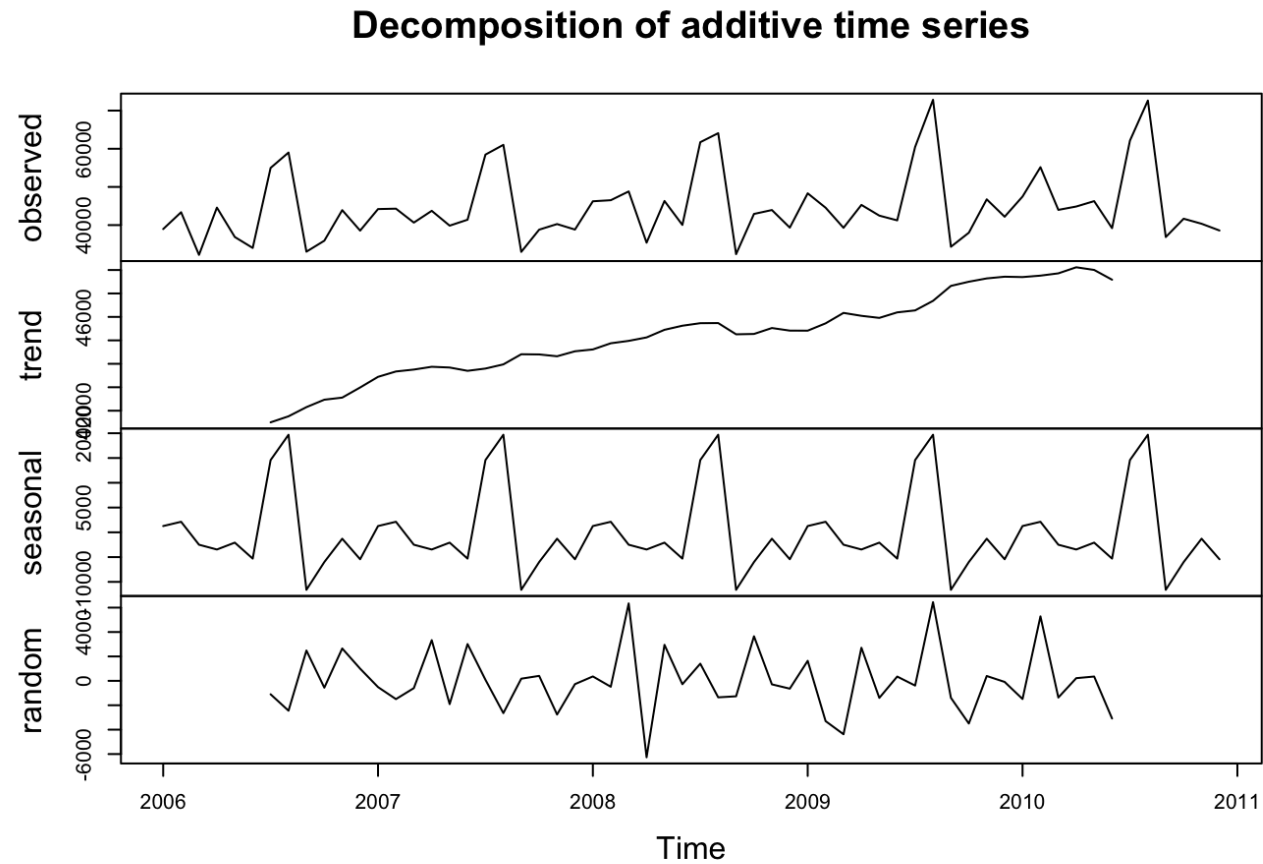
Modelo aditivo común:

$$X_t = T_t + S_t + \varepsilon_t$$



Separar en componentes ayuda a entender y predecir el comportamiento de la serie.

🧱 Componentes de una serie temporal



📌 Imagen tomada de [Series de Tiempo - Victor Morales](#).

Estacionalidad vs Serie estacionaria


- **Estacionalidad:**
 - Patrón que se repite en intervalos fijos (mensual, anual, etc.).
 - La media varía de forma cíclica.
- **Serie estacionaria:**
 - No tiene tendencia ni estacionalidad.
 - Su comportamiento estadístico es constante en el tiempo.

Característica	Estacionalidad	Serie estacionaria
Patrón que se repite cíclicamente	✓	✗
Media constante	✗	✓
Autocorrelación estable	✗	✓



Series estacionarias y tipos de error

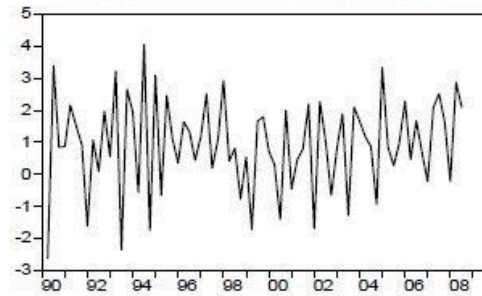
- Una serie es **estacionaria** si mantiene su **media**, **varianza** y **autocorrelación** constantes a lo largo del tiempo.
→ Muchos modelos clásicos lo requieren.
- **Error con estructura**: muestra dependencia temporal (por ejemplo, **autocorrelación**).
- **Ruido blanco**: error completamente aleatorio.
→ Media constante, varianza constante y sin autocorrelación.

 Una serie puede ser estacionaria aunque tenga error, siempre que sus propiedades estadísticas no cambien.

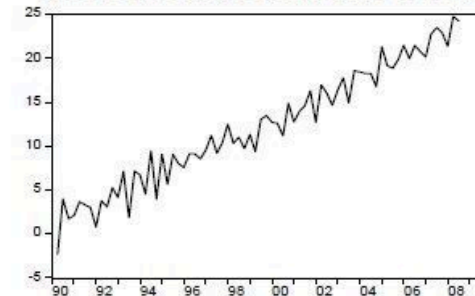


Ejemplos

Proceso estocástico estacionario



Proceso estocástico estacionario en media



Proceso estocástico no estacionario



Proceso estocástico no estacionario

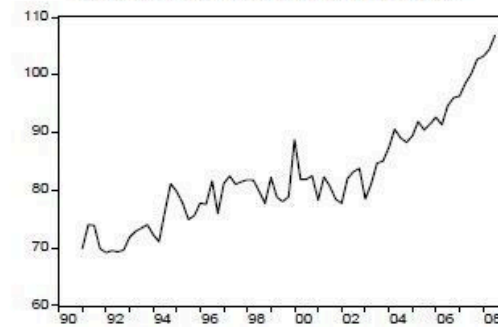
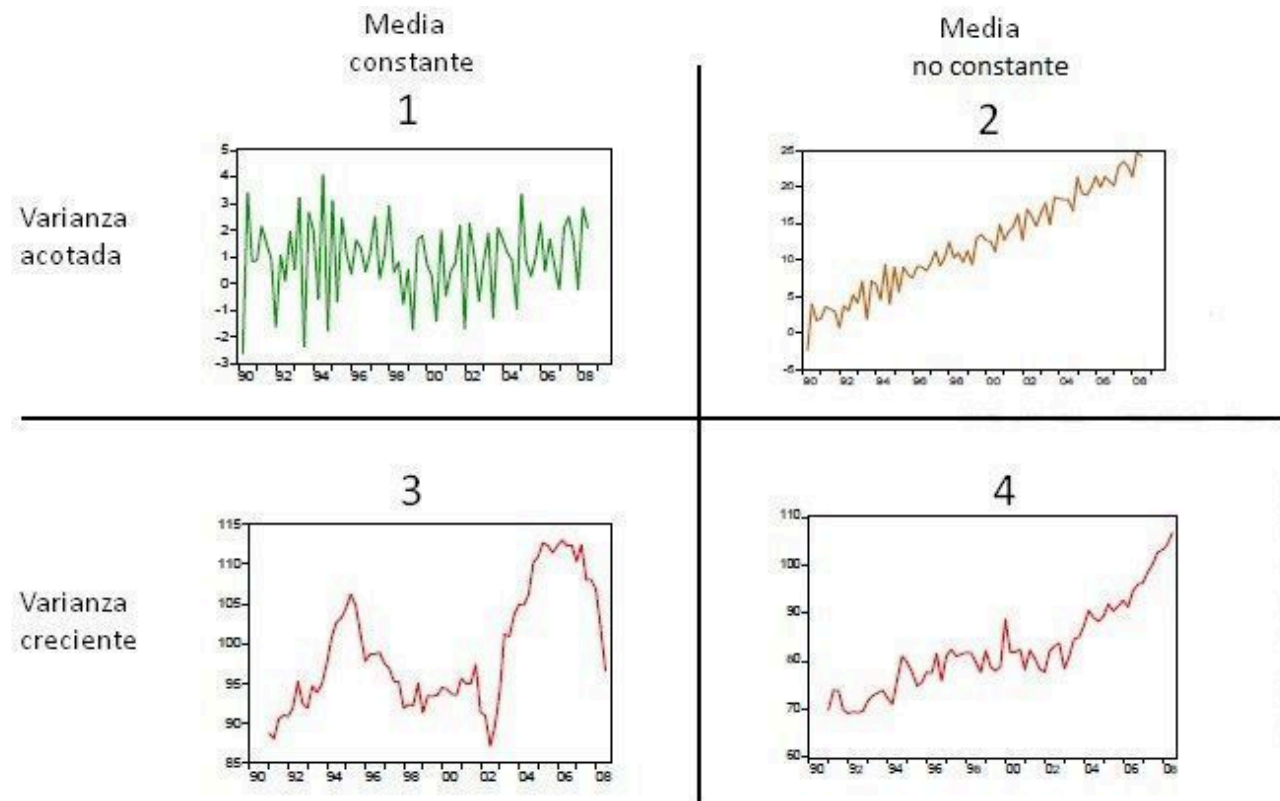


 Imagen tomada de [Economipedia](#).



Ejemplos



📌 Imagen tomada de [Economipedia](#).

¿Cómo transformar una serie a estacionaria?

1. Primera diferencia: elimina la tendencia

Se calcula restando el valor anterior (o el rezago que se seleccione):

$$X'_t = X_t - X_{t-1}$$

Ejemplo:

- Serie original: [10, 12, 15, 18, 20]
- Primera diferencia: [_, 2, 3, 3, 2]

La primera posición queda vacía porque no hay un X_0 .

Resultado: ahora los cambios entre periodos son más constantes.

2. Diferencia estacional: elimina la estacionalidad

Se calcula restando el valor del mismo periodo anterior (ej. hace 12 meses):

$$X_t'' = X_t - X_{t-s}$$

Ejemplo con $s = 3$:

- Serie: [20, 22, 19, 25, 27, 24]
- Diferencia estacional: [_, _, _, 5, 5, 5]

Se resta el valor de 3 pasos atrás.

■ Cuando el patrón se repite cada cierto tiempo, esta operación lo neutraliza.

¿Y si tiene tendencia y estacionalidad?

Se aplican **ambas** diferencias:

1. Primero: $X_t - X_{t-1}$ (tendencia)
2. Luego: $(X_t - X_{t-1}) - (X_{t-s} - X_{t-s-1})$ (estacionalidad)

¿Por qué transformamos una serie?

Modelos clásicos (AR, MA, ARIMA)

- Estos modelos suponen que la serie es **estacionaria**.
- Si la serie tiene tendencia o estacionalidad, **violan sus supuestos**.
- Resultado: el modelo **no predice bien** y sus estimaciones son sesgadas o inconsistentes.


→ Por eso, aplicamos diferencias para estabilizar la media y eliminar patrones sistemáticos.

¿Y en modelos de IA?

- En redes neuronales (RNN, LSTM, GRU, Transformers), **no es obligatorio transformar la serie a estacionaria**.
- Estos modelos pueden aprender **tendencias y estacionalidades complejas** por sí solos.

Aun así, **es clave preparar bien los datos**:


- Crear ventanas temporales.
- Escalar valores.
- Organizar correctamente las secuencias.

 Aunque no se requiere transformarla, volver una serie estacionaria puede ayudar a **resaltar patrones** y estructurar mejor los datos, según el caso.

Limitaciones de las redes neuronales tradicionales

Las redes neuronales feedforward (*MLP*) tienen limitaciones significativas al trabajar con datos secuenciales:

- No mantienen estado interno o "memoria".
- Asumen independencia entre observaciones.
- Procesan entradas de longitud fija.
- No capturan dependencias temporales.

 Para analizar series temporales, necesitamos redes que puedan "recordar" información previa y procesar secuencias de longitud variable.

Redes Neuronales Recurrentes (RNN)

Las RNN son redes diseñadas específicamente para procesar datos secuenciales, como las series temporales.

Idea clave

A diferencia de las redes tradicionales, las RNN tienen "memoria" mediante conexiones recurrentes, lo que permite que la información persista en el tiempo.

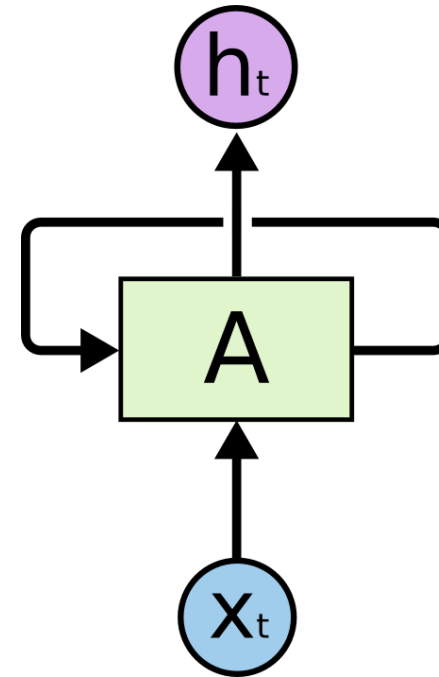
Ventajas

- Pueden procesar secuencias de longitud variable.
- Mantienen contexto a lo largo del tiempo.
- Comparten parámetros en cada paso temporal.

Arquitectura RNN: Componentes

Elementos principales

- **Entrada (X_t):** vector de datos en el tiempo t .
 - *Ejemplo: [precio, volumen, volatilidad].*
- **Estado oculto (h_t):** la "memoria" de la red.
 - *Almacena información contextual.*
- **Salida (y_t):** predicción en el tiempo t .
 - *Ejemplo: precio futuro, probabilidad de recesión.*



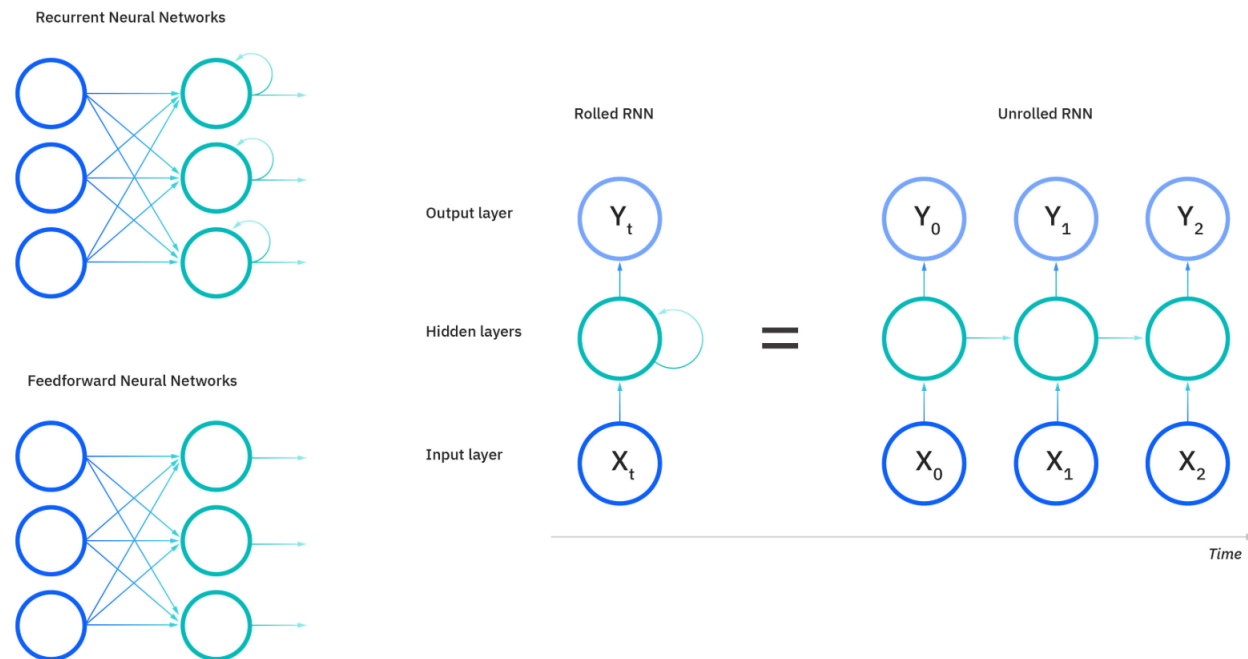
Fuente: [Colah's blog](#)

Funcionamiento de la RNN

Proceso secuencial

En cada paso temporal, la RNN:

1. **Recibe** una nueva entrada X_t .
2. **Combina** esta entrada con el estado anterior h_{t-1} .
3. **Actualiza** el estado oculto h_t .
4. **Genera** una predicción y_t .



Fuente: IBM

La misma función se aplica en cada paso, permitiendo procesar secuencias de cualquier longitud.

Matemática de las RNN: Simple pero potente

Ecuaciones fundamentales

Estado oculto:

$$h_t = \tanh(W_{xh}X_t + W_{hh}h_{t-1} + b_h)$$

Salida:

$$y_t = W_{hy}h_t + b_y$$

Donde:

- W_{xh} : pesos entrada-oculta.
- W_{hh} : pesos recurrentes.
- W_{hy} : pesos oculta-salida.
- b_h, b_y : vectores de sesgo.

Matemática de las RNN: Simple pero Potente

Ecuaciones Fundamentales

Estado oculto:

$$h_t = \tanh(W_{xh}X_t + W_{hh}h_{t-1} + b_h)$$

Salida:

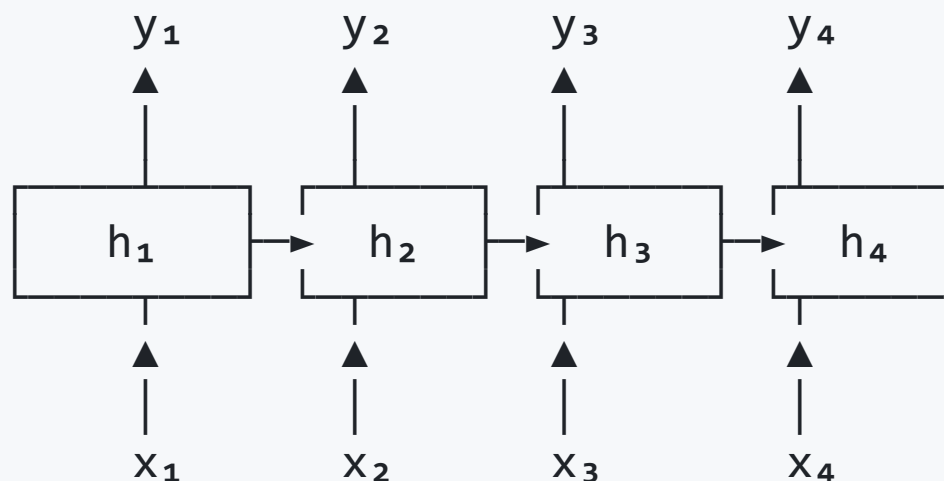
$$y_t = W_{hy}h_t + b_y$$

 **Dato importante:**

Los mismos parámetros se reutilizan en cada paso temporal.



Visualizando el Despliegue en el Tiempo



Observaciones clave

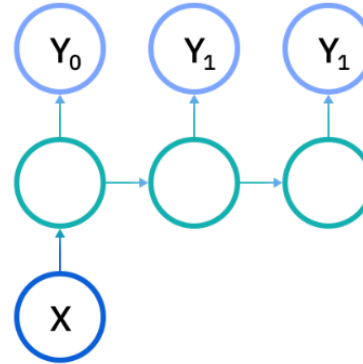
- Se aplica la **misma función** en cada paso temporal.
- Se usan los **mismos parámetros** en toda la secuencia.
- El estado h_t depende de h_{t-1} y x_t .

Tipos de RNN

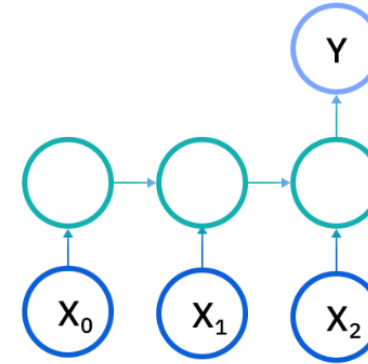
One-to-one



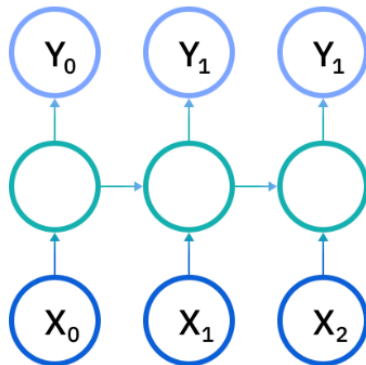
One-to-many



Many-to-one



Many-to-many



Many-to-many

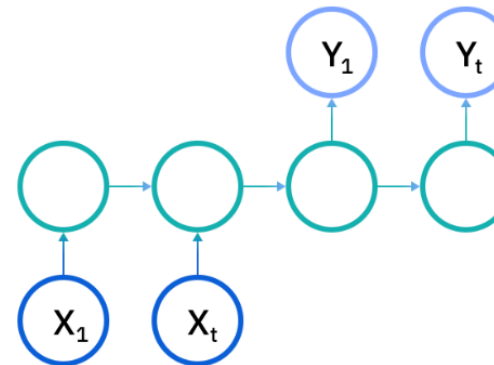


 Imagen tomada de [IBM](#).

MLP vs RNN: ¿Qué cambia?

Característica	MLP	RNN (One-to-One)	RNN (Many-to-One)
Tipo de entrada	Vector único	Un paso de una secuencia	Secuencia completa
Uso de memoria	✗ No	✓ Parcial (solo del paso actual)	✓ Acumula información del pasado
Ejemplo económico	Clasificar riesgo de una operación	Clasificar una transacción dentro de una secuencia	Predecir recesión con datos de los últimos 12 meses
Ventaja	Simple y rápida	Reconoce contexto inmediato	Captura relaciones temporales profundas


 Las MLP no manejan secuencias. Las RNN procesan tiempo paso a paso y retener contexto.

Tipos de RNN y Aplicaciones

Tipo	Entrada → Salida	Ejemplo
One-to-One	Un paso → una salida	Clasificación de una operación financiera como sospechosa (no secuencial).
Many-to-One	Secuencia → una salida	12 meses de variables macroeconómicas → probabilidad de recesión.
One-to-Many	Un paso → secuencia	Simular el efecto de un cambio en política fiscal sobre varias variables futuras.
Many-to-Many	Secuencia → secuencia	Forecast conjunto de inflación, tipo de cambio y tasas de interés. Este tipo también se utiliza en tareas de texto: traducción automática (ej. español → inglés).

¿Y cómo encaja en una red más grande?

- Una RNN suele ser **solo una capa**.
- Puede combinarse con otras capas como:
 - Capa densa (para regresión o clasificación)
 - Capa de embedding (en texto o categorías)
 - Otras RNN apiladas (stacked)

 Una RNN captura la dinámica temporal. El resto de la red transforma esa información en predicciones útiles.



Preparación de Datos para Series Temporales




División Train-Test

- **Objetivo:** evaluar la capacidad predictiva del modelo.
- **Método estándar:** división cronológica (no aleatoria).
 - **80%** de los datos para entrenamiento.
 - **20%** para prueba.

División cronológica de datos en series temporales.

```
# Ejemplo de código  
n = len(datos)  
train_size = int(n * 0.8)  
  
train_data = datos[:train_size]  
test_data = datos[train_size:]
```

 A diferencia de otros problemas de ML, en series temporales la división debe respetar el orden cronológico.

Ventanas Deslizantes: El Corazón del Entrenamiento

¿Qué son las ventanas deslizantes?

Es una técnica para convertir series temporales en un formato adecuado para redes RNN:

1. **Ventana de entrada:** n valores consecutivos.
2. **Valor objetivo:** siguiente valor a predecir.
3. **Deslizamiento:** mover la ventana un paso a la vez.

Ejemplo con PIB trimestral

Datos: [8.3, 8.6, 8.8, 9.1, 9.4, 9.8, 10.0, 10.3]

Ventana tamaño 4

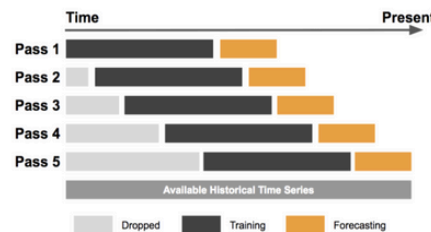
$X1 = [8.3, 8.6, 8.8, 9.1] \rightarrow Y1 = 9.4$

$X2 = [8.6, 8.8, 9.1, 9.4] \rightarrow Y2 = 9.8$

$X3 = [8.8, 9.1, 9.4, 9.8] \rightarrow Y3 = 10.0$

$X4 = [9.1, 9.4, 9.8, 10.0] \rightarrow Y4 = 10.3$

Sliding Window



Expanding Window

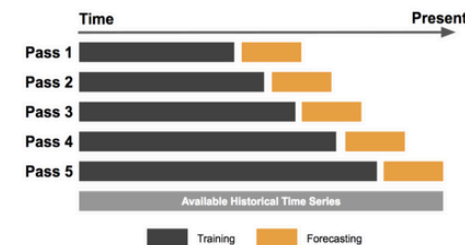


Ilustración del proceso de ventana deslizante

Métricas de Evaluación para Series Temporales

- **MSE** (Error Cuadrático Medio):

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

- **RMSE** (Raíz del Error Cuadrático Medio):

$$RMSE = \sqrt{MSE}$$

- **MAE** (Error Absoluto Medio):

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

 Se calculan comparando las predicciones con los valores reales del conjunto de prueba.



Proceso de Evaluación



Consideraciones importantes

- **Evaluación en datos no vistos:**
El modelo, entrenado con el 80% de los datos, genera predicciones sobre el 20% restante.
- **Conjunto de prueba como referencia:**
Se comparan las predicciones con los valores reales del conjunto de prueba.



Estrategias de predicción

- **Horizonte de predicción:**
 - **One-step:** predecir el siguiente valor.
 - **Multi-step:** predecir varios valores futuros.
- **Comparación con baseline:**
 - Método ingenuo (último valor).
 - Promedio móvil.
 - ARIMA.

Ejemplo de comparativa

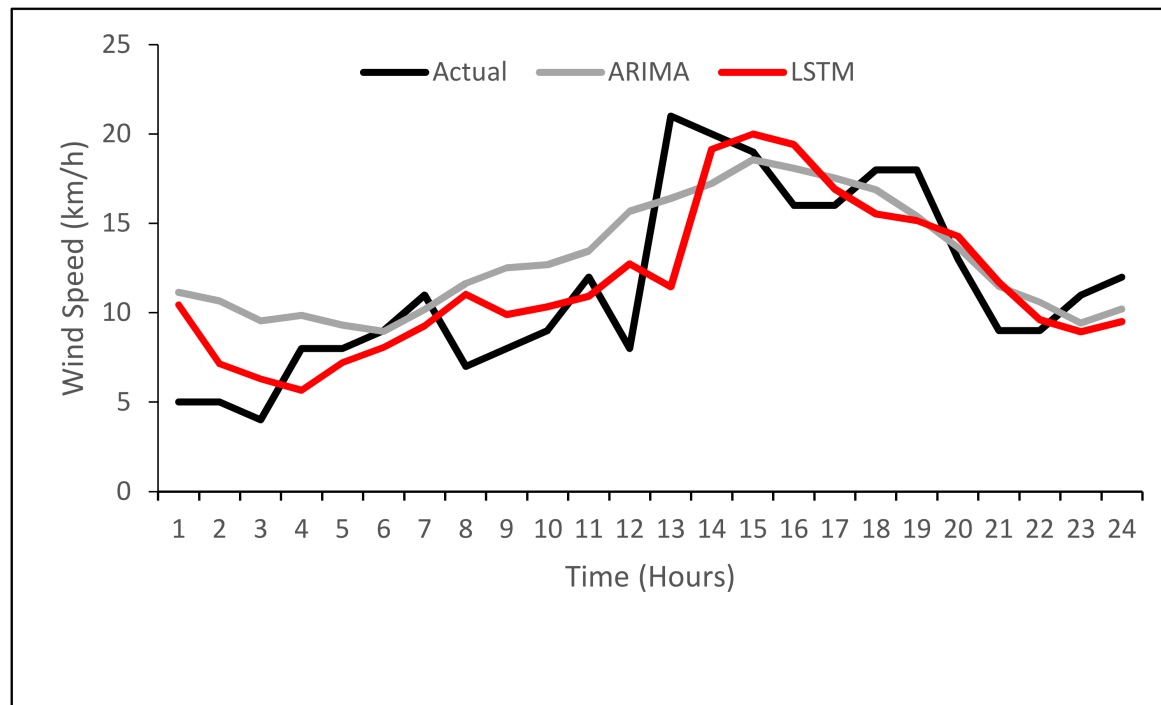


 Imagen tomada de [MDPI](#).



Proceso Completo de Modelado - Flujo de trabajo

1. Preparación:

- División train-test (80%-20%).
- Creación de ventanas deslizantes.
- Normalización de datos.

2. Entrenamiento:

- Alimentar ventanas al modelo.
- Ajustar pesos para minimizar el error.
- Validación con datos no vistos.

3. Evaluación:

- Predicción secuencial en test.
- Cálculo de métricas (RMSE, MAE).
- Comparación con baseline.

4. Producción:

- Entrenamiento final con toda la serie.
- Predicciones sobre nuevos datos.
- Monitoreo y reentrenamiento continuo.

El Problema del Gradiente Desvaneciente

¿Qué ocurre?

Al entrenar RNNs con backpropagation, los gradientes:

- Se multiplican repetidamente por W_{hh} .
- Si los valores propios de $W_{hh} < 1$, el gradiente **se desvanece**.
- Si los valores propios de $W_{hh} > 1$, el gradiente **explota**.

Consecuencias

- **Memoria a corto plazo:** la red olvida información pasada.
- **Dificultad para capturar dependencias a largo plazo.**
- **Entrenamiento inestable.**

El Problema del Gradiente Desvaneciente

Implicaciones

- En secuencias largas, los gradientes tienden a desvanecerse, limitando la capacidad de aprendizaje.
- A medida que retropropagamos en el tiempo, los gradientes se vuelven tan pequeños que la red deja de aprender efectivamente.

! Explorando la Teoría del Gradiente Desvaneciente

1.1. Training recurrent networks

A generic recurrent neural network, with input \mathbf{u}_t and state \mathbf{x}_t for time step t , is given by equation (1). In the theoretical section of this paper we will sometimes make use of the specific parametrization given by equation (11) ¹ in order to provide more precise conditions and intuitions about the everyday use-case.

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta) \quad (1)$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \quad (2)$$

The parameters of the model are given by the recurrent weight matrix \mathbf{W}_{rec} , the biases \mathbf{b} and input weight matrix \mathbf{W}_{in} , collected in θ for the general case. \mathbf{x}_0 is provided by the user, set to zero or learned, and σ is an element-wise function (usually the *tanh* or *sigmoid*). A cost \mathcal{E} measures the performance of the network on some given task and it can be broken apart into individual costs for each step $\mathcal{E} = \sum_{1 \leq t \leq T} \mathcal{E}_t$, where $\mathcal{E}_t = \mathcal{L}(\mathbf{x}_t)$.

One approach that can be used to compute the necessary gradients is Backpropagation Through Time (BPTT), where the recurrent model is represented as

¹ This formulation is equivalent to the more widely known equation $\mathbf{x}_t = \sigma(\mathbf{W}_{rec}\mathbf{x}_{t-1} + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b})$, and it was chosen for convenience.

a deep multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model (see Fig. 2).

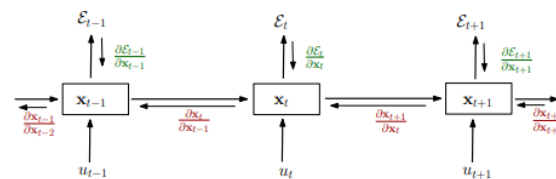


Figure 2. Unrolling recurrent neural networks in time by creating a copy of the model for each time step. We denote by \mathbf{x}_t the hidden state of the network at time t , by \mathbf{u}_t the input of the network at time t and by \mathcal{E}_t the error obtained from the output at time t .

We will diverge from the classical BPTT equations at this point and re-write the gradients (see equations (3), (4) and (5)) in order to better highlight the exploding gradients problem. These equations were obtained by writing the gradients in a sum-of-products form.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T \text{diag}(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

 Paper: *On the difficulty of training Recurrent Neural Networks*

LSTM: La solución al gradiente desvaneciente

Las **Long Short-Term Memory** (LSTM) son una variante avanzada de RNN, diseñadas para:

- Mantener información durante periodos prolongados.
- Evitar el problema del gradiente desvaneciente.
- Aprender qué información conservar y cuál descartar.

Componente clave

Las LSTM utilizan **compuertas** (*gates*) que controlan el flujo de información a lo largo del tiempo.

¿Qué son las compuertas?

¿Para qué sirven?


- Deciden qué información **recordar**, **actualizar** u **olvidar** en cada paso de tiempo.
- Son clave para que la red maneje dependencias **de largo plazo**.


En redes neuronales recurrentes como LSTM:

- Son **mecanismos matemáticos** que controlan el flujo de información dentro de la red.
- Se implementan como **funciones diferenciables** con **pesos entrenables** y **activación sigmoide**.
- Devuelven valores entre **0 y 1**, que indican **cuánta información puede pasar** (0 = nada, 1 = todo).

¿Qué son las compuertas?

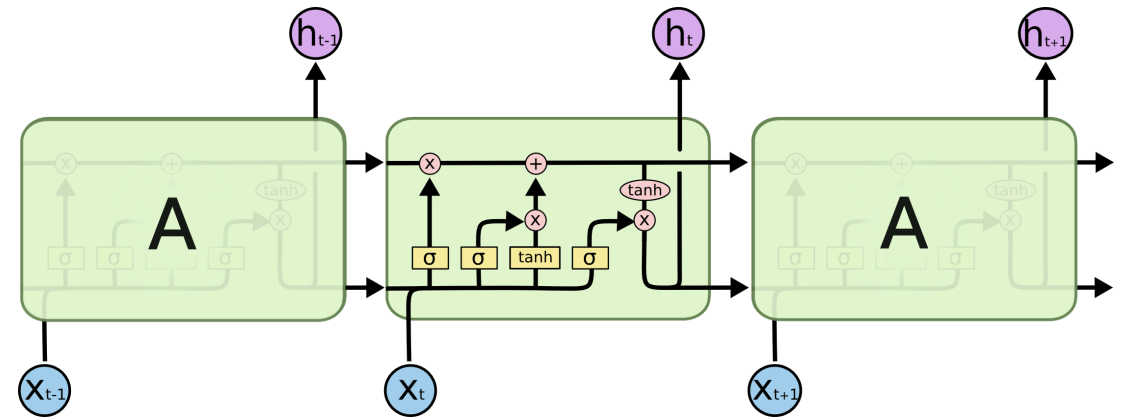
Durante el entrenamiento:

- Las compuertas **se aprenden automáticamente** con backpropagation.
-  No se configuran como hiperparámetros.
- Funcionan como **válvulas inteligentes** que se abren o cierran según el contexto.

 Sin compuertas, una RNN olvida rápido. Las compuertas permiten que la red elija lo relevante.

🔍 Los tipos de compuertas LSTM

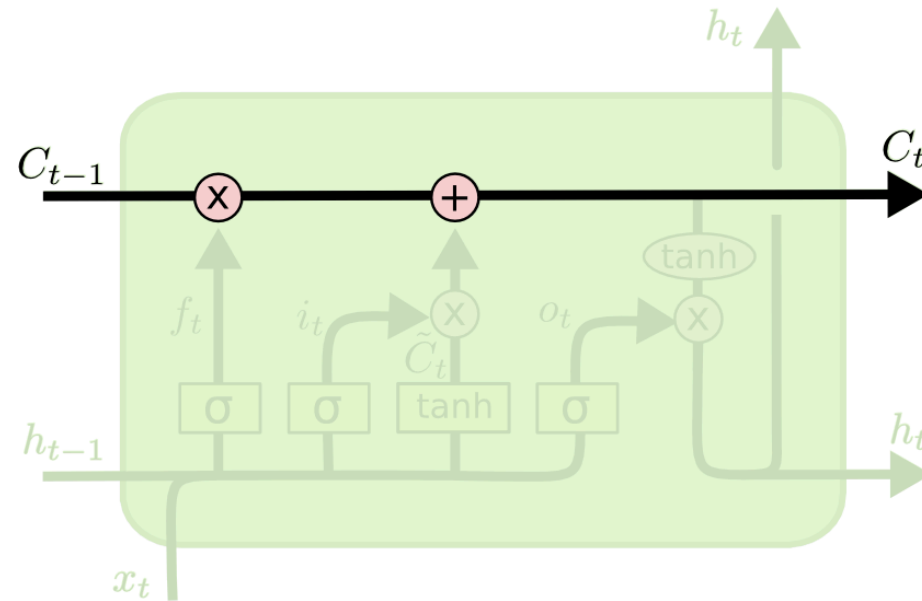
1. **Compuerta de olvido (f_t):**
Decide qué información descartar.
2. **Compuerta de entrada (i_t):**
Determina qué nueva información almacenar.
3. **Compuerta de salida (o_t):**
Controla qué información se transmite al siguiente paso.
4. **Celda de memoria (c_t):**
Almacena información a largo plazo.



Fuente: [Colah's blog](#)

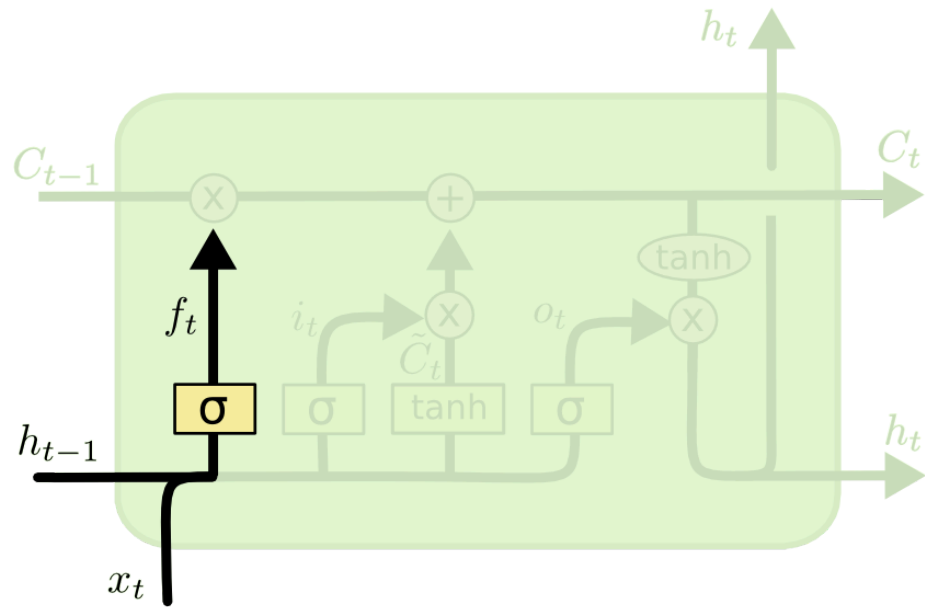
Las compuertas utilizan funciones *sigmoid* y *tanh* para controlar el flujo de información.

▢ Matemática de las LSTM: Cell State / Celda de memoria



Fuente: [Colah's blog](#)

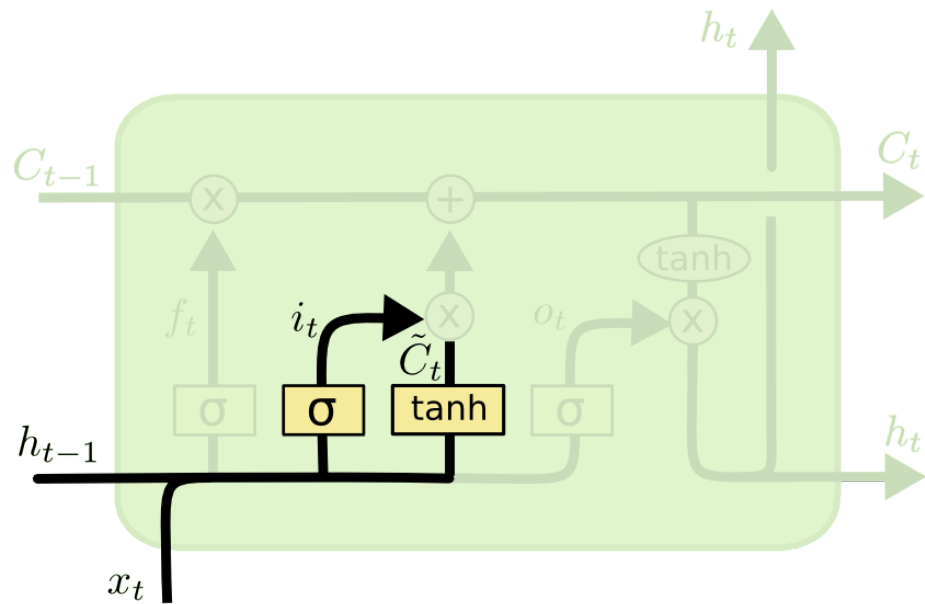
▢ Matemática de las LSTM: Compuerta de olvido (f_t)



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Fuente: *Colah's blog*

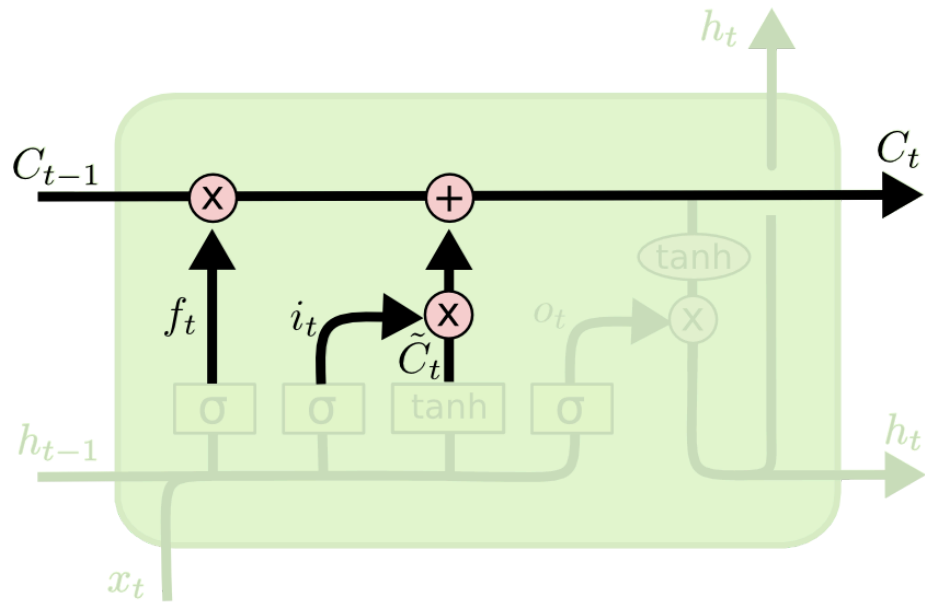
▶ Matemática de las LSTM: Compuerta de entrada (i_t):



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Fuente: [Colah's blog](#)

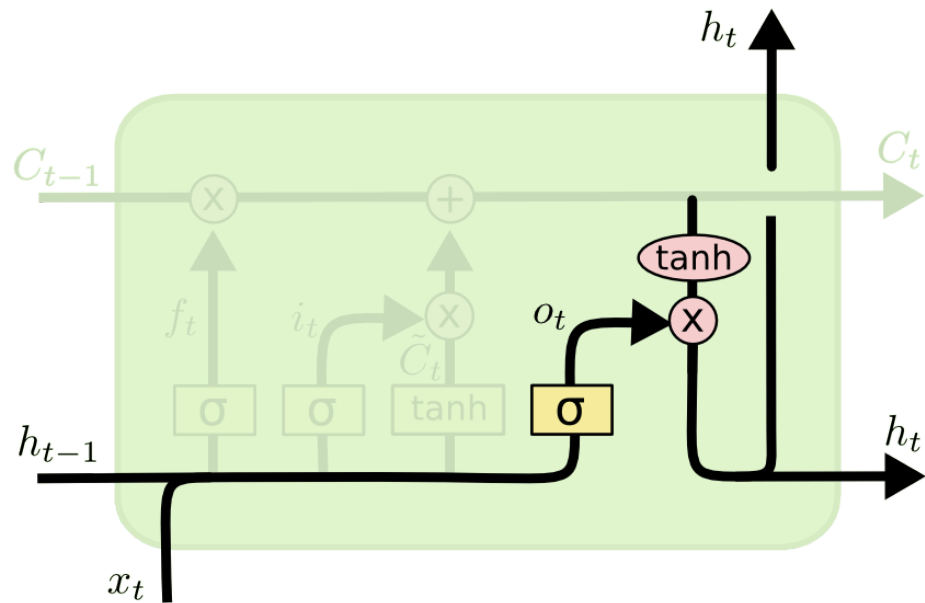
▢ Matemática de las LSTM: Cell State / Celda de memoria



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

Fuente: [Colah's blog](#)

▢ Matemática de las LSTM: Compuerta de salida (o_t)



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

Fuente: [Colah's blog](#)

Analogía: LSTM como Analista Financiero

1 Compuerta de Olvido

- Como descartar informes obsoletos.
- *"El mercado ha cambiado, estos datos ya no son relevantes"*.
- Ejemplo: Ignorar patrones pre-pandemia en análisis actuales.

2 Compuerta de Entrada

- Como archivar información crucial.
- *"Esta alza de tasas es importante, debo guardarla"*.

3 Celda de Memoria

- Como una caja fuerte de información.
- *"Aquí guardo las tendencias importantes a largo plazo"*.
- Ejemplo: Mantener patrones de ciclos económicos.

4 Compuerta de Salida

- Como preparar un informe actual
- *"Para la predicción de hoy, selecciono esta información"*.

GRU: La Alternativa Eficiente

Las **Gated Recurrent Unit** (GRU) son una variante más simple de las LSTM, introducidas por Cho et al. en 2014.

Características Principales

- Fusionan las compuertas de olvido y entrada.
- Eliminan la celda de memoria separada.
- Menos parámetros que LSTM.
- Rendimiento similar en muchas tareas.
- Entrenamiento más rápido y eficiente.

Matemática de las GRU: Simplicidad Efectiva

Las GRU utilizan sólo **dos compuertas** para controlar el flujo de información:

1. Compuerta de actualización (z_t):

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

2. Compuerta de reinicio (r_t):

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

3. Candidato a nuevo estado (\tilde{h}_t):

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

4. Estado oculto final:

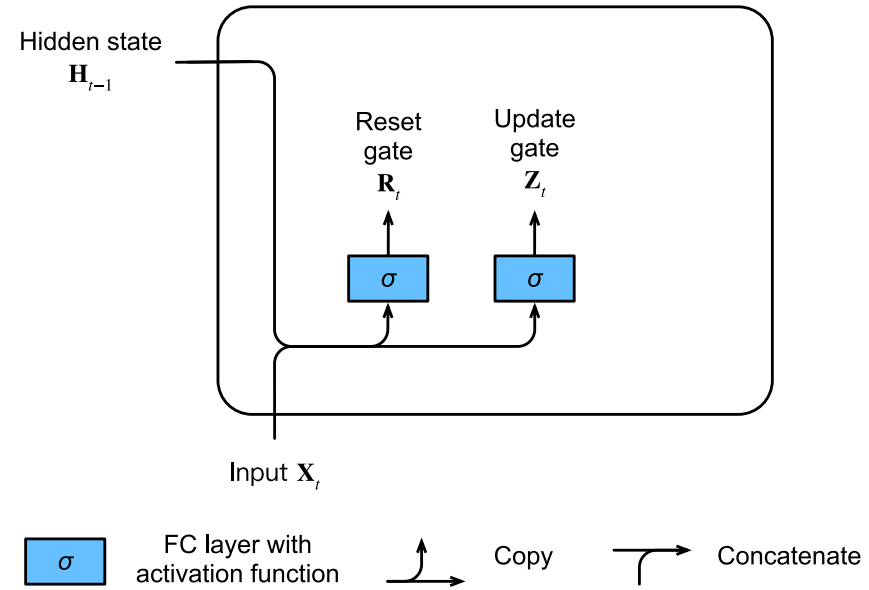
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

🔄 Analogía para entender las compuertas GRU

📖 Primera compuerta

1. Compuerta de Actualización (z_t)

- Decide cuánto confiar en información nueva vs. antigua.
- *"¿Sigo mi estrategia anterior o doy peso al anuncio reciente?"*
- Si hay un anuncio importante → prioriza la nueva información.
- Si es ruido de mercado → se aferra a su estrategia previa.



Tomado de: [Dive into Deep Learning](#)

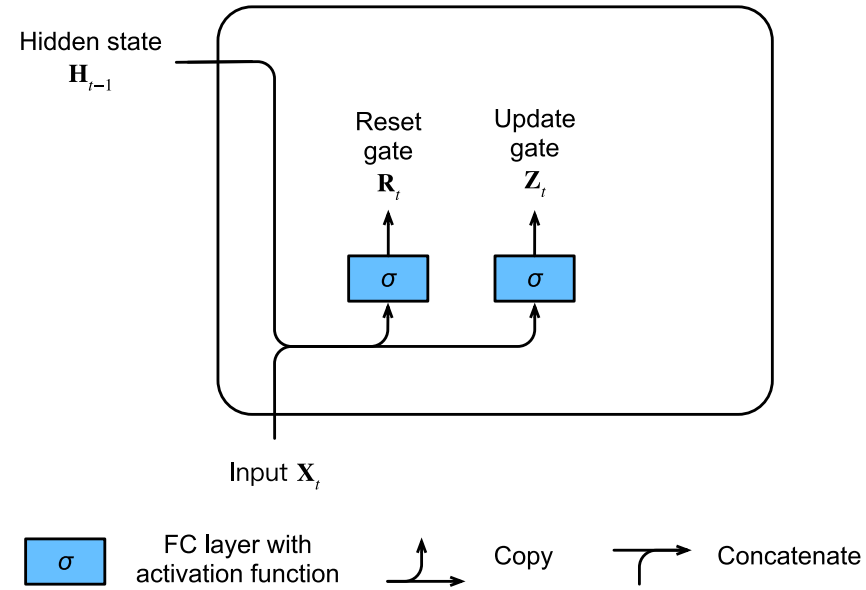
La GRU es como un operador de mercado ágil: sabe cuándo cambiar de estrategia y cuándo ignorar el ruido.

🔄 Analogía para entender las compuertas GRU

📖 Segunda compuerta

2. Compuerta de Reinicio (r_t)

- Decide cuándo "empezar de cero" tras un cambio de paradigma.
-- "*¿Este cambio en política monetaria invalida mis análisis previos?*"



Tomado de: [Dive into Deep Learning](#)

La GRU es como un operador de mercado ágil: sabe cuándo cambiar de estrategia y cuándo ignorar el ruido.



Comparativa de Modelos

Baso en experiencias de predicción de mercados financieros.

Modelo	Precisión	Velocidad	Memoria	Recursos necesarios
RNN	★ ★	★ ★ ★ ★ ★	★ ★	★ ★
LSTM	★ ★ ★ ★ ★	★ ★ ★	★ ★ ★ ★ ★	★ ★ ★ ★ ★
GRU	★ ★ ★ ★	★ ★ ★ ★	★ ★ ★ ★	★ ★ ★



Casos de uso ideales

RNN Simple

- Mercados estables sin interrupciones.
- Predicciones a muy corto plazo.
- Sistemas con recursos limitados.
- Procesamiento en tiempo real sencillo.
- Alto volumen de datos simples.

LSTM/GRU

- Mercados volátiles o con interrupciones.
- Predicciones a medio/largo plazo.
- Patrones económicos complejos.
- Detección de cambios de régimen.
- Análisis de crisis financieras.

Ejemplo real: Durante la crisis del 2008, los modelos LSTM predijeron correctamente patrones de recuperación basándose en crisis anteriores, mientras que las RNN simples fallaron consistentemente.

Recursos del Curso

Plataformas y Enlaces Principales

GitHub del curso

 github.com/CamiloVga/IA_Aplicada

Asistente IA para el curso

 [Google Notebook LLM](#)