# Redes Neuronales Recurrentes (RNN)

## ¿Qué es una RNN?

Una **Red Neuronal Recurrente (RNN)** es un tipo especial de red neuronal diseñada para trabajar con datos que tienen **dependencia temporal o secuencial**. Ejemplos de estos datos son:

* **Texto**: Predecir palabras, entender frases completas.
* **Series temporales**: Analizar patrones pasados para predecir el futuro.
* **Audio**: Reconocer palabras en función de las anteriores.

La **memoria interna** es lo que hace que las RNN sean diferentes. Esto significa que, en lugar de procesar cada dato de manera independiente, una RNN recuerda información sobre entradas anteriores y la utiliza al procesar la siguiente entrada.

La **dependencia del tiempo o del orden** significa que una RNN puede capturar relaciones entre eventos pasados y futuros. Por ejemplo:

* En un texto, entender una palabra puede depender de las palabras anteriores.
* En audio, un sonido puede depender de notas anteriores.

**Problema del gradiente desaparecido/explosivo**:

* Las RNN tradicionales tienen dificultades para manejar secuencias largas porque el gradiente puede desvanecerse o explotar durante el entrenamiento.

## RNN BASICA

model\_rnn = Sequential([ SimpleRNN(16, activation='tanh', input\_shape=(X\_train.shape[1], X\_train.shape[2])), Dropout(0.5), Dense(16, activation='relu'), Dense(3, activation='softmax') # 3 clases de salida ])

## RNN CON LSTM:

**LSTM (Long Short-Term Memory)** es un tipo de **Red Neuronal Recurrente (RNN)** diseñada para capturar dependencias a largo plazo en datos secuenciales. A diferencia de las RNN tradicionales, las LSTM están equipadas con **puertas** que controlan el flujo de información, lo que les permite recordar o olvidar información de manera más efectiva.

model\_lstm = Sequential([ LSTM(64, activation='tanh', return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])), Dropout(0.5), LSTM(32, activation='tanh', return\_sequences=True), Dropout(0.5), LSTM(16, activation='tanh'), Dropout(0.5), Dense(16, activation='relu'), Dense(3, activation='softmax') # 3 clases de salida ])

## 

## RNN CON LSTM Y BATRCH NROMALIZATION:

model\_lstm\_bn = Sequential([ LSTM(64, activation='tanh', return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])), BatchNormalization(), Dropout(0.5), LSTM(32, activation='tanh', return\_sequences=True), BatchNormalization(), Dropout(0.5), LSTM(16, activation='tanh'), BatchNormalization(), Dropout(0.5), Dense(32, activation='relu'), Dropout(0.5), Dense(3, activation='softmax') # 3 clases de salida ])

**Batch Normalization (Batch Norm)** es una técnica utilizada para normalizar las entradas de cada capa durante el entrenamiento. Ayuda a estabilizar y acelerar el proceso de entrenamiento, permitiendo el uso de tasas de aprendizaje más altas y reduciendo la sensibilidad a la inicialización de los pesos.

### **¿Cuándo y Cuánto Utilizar Batch Normalization?**

**Cuándo utilizar Batch Norm:**

* **Modelos Profundos:** En redes con múltiples capas, Batch Norm puede ayudar a mitigar el problema de desvanecimiento o explosión del gradiente.

## INPUT SHAPE

Las capas **SimpleRNN** en Keras esperan que los datos de entrada tengan una **forma específica**. Es crucial estructurar tus datos correctamente para que la red pueda procesarlos adecuadamente.

**Dimensión 1 (samples):** Número de muestras o instancias en tu conjunto de datos.

**Dimensión 2 (timesteps):** Número de pasos de tiempo en cada secuencia.

**Dimensión 3 (features):** Número de características o variables en cada paso de tiempo.

### **a. Clasificación Secuencial (Categoría Discreta)**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense, Embedding, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

# Datos de ejemplo

texts = [

"Me encanta este producto",

"Este es el peor servicio que he recibido",

"No está mal, pero podría mejorar",

"Fantástico! Superó mis expectativas",

"No me gustó nada",

# ... más oraciones

]

labels = [2, 0, 1, 2, 0] # 0: Negativo, 1: Neutral, 2: Positivo

# Parámetros

vocab\_size = 5000

max\_length = 20

embedding\_dim = 100

num\_classes = 3

# Tokenización y secuenciación

tokenizer = Tokenizer(num\_words=vocab\_size, oov\_token="<OOV>")

tokenizer.fit\_on\_texts(texts)

sequences = tokenizer.texts\_to\_sequences(texts)

# Padding

X = pad\_sequences(sequences, maxlen=max\_length, padding='post', truncating='post')

y = to\_categorical(labels, num\_classes=num\_classes)

# División en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=labels

)

# Construcción del modelo SimpleRNN

model = Sequential([

Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=max\_length),

SimpleRNN(64, activation='tanh'),

Dropout(0.5),

Dense(16, activation='relu'),

Dense(num\_classes, activation='softmax') # 3 clases de salida

])

# Compilación del modelo

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Resumen del modelo

model.summary()

# Entrenamiento del modelo

history = model.fit(

X\_train, y\_train,

epochs=50,

batch\_size=16,

validation\_split=0.2,

verbose=1

)

# Evaluación del modelo

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Precisión en el conjunto de prueba: {accuracy:.4f}')

### b. Regresión Secuencial (Valor Continuo)

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

# Generar datos de ejemplo

data = np.sin(np.linspace(0, 100, 1000)) # Datos simulados de temperatura

# Crear secuencias de 10 días para predecir el siguiente día

timesteps = 10

X = []

y = []

for i in range(len(data) - timesteps):

X.append(data[i:i+timesteps])

y.append(data[i+timesteps])

X = np.array(X)

y = np.array(y)

# Escalar los datos

scaler\_X = MinMaxScaler()

X = scaler\_X.fit\_transform(X)

scaler\_y = MinMaxScaler()

y = scaler\_y.fit\_transform(y.reshape(-1, 1)).flatten()

# Reshape para SimpleRNN: (samples, timesteps, features)

X = X.reshape((X.shape[0], timesteps, 1))

# División en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# Construcción del modelo SimpleRNN

model = Sequential([

SimpleRNN(50, activation='tanh', input\_shape=(timesteps, 1)),

Dense(1)

])

# Compilación del modelo

model.compile(optimizer='adam', loss='mse')

# Resumen del modelo

model.summary()

# Entrenamiento del modelo

model.fit(

X\_train, y\_train,

epochs=20,

batch\_size=32,

validation\_split=0.2,

verbose=1

)

# Evaluación del modelo

loss = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Loss en el conjunto de prueba: {loss:.4f}')

### c. Clasificación Multivariada Secuencial

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Datos de ejemplo

num\_videos = 100

timesteps = 30

features = 6 # e.g., 3 ejes de acelerómetro + 3 ejes de giroscopio

num\_classes = 3

# Generar datos simulados

X = np.random.rand(num\_videos, timesteps, features)

y = np.random.randint(0, num\_classes, num\_videos)

y\_categorical = to\_categorical(y, num\_classes=num\_classes)

# Escalar los datos

scaler = StandardScaler()

X\_flat = X.reshape(-1, features)

X\_scaled = scaler.fit\_transform(X\_flat).reshape(num\_videos, timesteps, features)

# División en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_scaled, y\_categorical, test\_size=0.2, random\_state=42, stratify=y

)

# Construcción del modelo SimpleRNN

model = Sequential([

SimpleRNN(64, activation='tanh', input\_shape=(timesteps, features)),

Dropout(0.5),

Dense(16, activation='relu'),

Dense(num\_classes, activation='softmax') # 3 clases de salida

])

# Compilación del modelo

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Resumen del modelo

model.summary()

# Entrenamiento del modelo

history = model.fit(

X\_train, y\_train,

epochs=50,

batch\_size=16,

validation\_split=0.2,

verbose=1

)

# Evaluación del modelo

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Precisión en el conjunto de prueba: {accuracy:.4f}')

### d. Regresión Multivariada Secuencial

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

# Generar datos de ejemplo

num\_samples = 500

timesteps = 20

features = 5 # e.g., indicadores técnicos: RSI, MACD, etc.

X = np.random.rand(num\_samples, timesteps, features)

y = np.random.rand(num\_samples, 2) # Predecir dos valores continuos, por ejemplo, precio de apertura y cierre

# Escalar los datos

scaler\_X = MinMaxScaler()

X\_flat = X.reshape(-1, features)

X\_scaled = scaler\_X.fit\_transform(X\_flat).reshape(num\_samples, timesteps, features)

scaler\_y = MinMaxScaler()

y\_scaled = scaler\_y.fit\_transform(y)

# División en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_scaled, y\_scaled, test\_size=0.2, random\_state=42

)

# Construcción del modelo SimpleRNN

model = Sequential([

SimpleRNN(128, activation='tanh', input\_shape=(timesteps, features)),

Dense(32, activation='relu'),

Dense(2) # Dos salidas para regresión

])

# Compilación del modelo

model.compile(optimizer='adam',

loss='mse')

# Resumen del modelo

model.summary()

# Entrenamiento del modelo

model.fit(

X\_train, y\_train,

epochs=100,

batch\_size=32,

validation\_split=0.2,

verbose=1

)

# Evaluación del modelo

loss = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Loss en el conjunto de prueba: {loss:.4f}')