Institución Universitaria Digital de Antioquia

Evidencia de Aprendizaje: S20 - Evidencia de aprendizaje 3: Implementación básica de una red neuronal

Asignatura: Principios de Deep Learning y Redes Neuronales

Estudiante: Jean Carlos Páez Ramírez

Grupo: PREICA2501B020140

Docente: Sharon Karin Camacho Guzman (Ingeniera Administradora Especialista en ingeniería Financiera de la Universidad Nacional de Colombia)

Fecha: 08 de junio de 2025

Introducción

Este trabajo implementa una **red neuronal recurrente básica (RNN)** utilizando **TensorFlow**, enfocada en el **análisis de sentimiento de comentarios de producto**s. El objetivo principal es **clasificar los comentarios en categorías positivas o negativas**, siguiendo los lineamientos establecidos en la evidencia de aprendizaje.

Para ello, **se construyó un modelo sencillo** que incluye una **capa de embedding** para la representación vectorial de las palabras, **una capa RNN** para el procesamiento secuencial de los datos y **una capa densa final** con activación **sigmoide** para la clasificación binaria.

Aunque el conjunto de datos empleado fue reducido y simulado, la implementación permite demostrar el funcionamiento general de este tipo de modelos. **Para mejorar el rendimiento en aplicaciones reales**, se recomienda utilizar **conjuntos de datos más amplios y representativos**, así como explorar arquitecturas más avanzadas como **LSTM o GRU**. Esta evidencia constituye una base sólida para **continuar el desarrollo y profundización en el proyecto integrador.**

```
# 1. Importación de librerías necesarias
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
```

```
# 2. Datos de ejemplo (comentarios de texto simulados)
comentarios = [
    "me encantó el producto",
    "muy mala calidad",
    "excelente servicio",
    "no lo recomiendo",
    "estoy satisfecho con la compra",
    "horrible experiencia",
    "muy buen producto",
    "no me gustó nada",
    "perfecto para lo que necesitaba",
    "pésimo servicio"
]
```

```
# 3. Etiquetas: 1 = positivo, 0 = negativo
etiquetas = np.array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0])
```

```
# 4. Tokenización del texto: convertir palabras a enteros
tokenizer = Tokenizer()
tokenizer.fit_on_texts(comentarios)
secuencias = tokenizer.texts_to_sequences(comentarios)
```

```
# 5. Padding para unificar longitud de secuencias
max_len = max(len(x) for x in secuencias)
entradas = pad_sequences(secuencias, maxlen=max_len, padding='post')
```

Nota sobre la capa Pooling

En modelos de procesamiento de texto con redes neuronales recurrentes (RNN), **la capa pooling no es necesaria**, ya que la RNN procesa toda la secuencia y genera una representación global en su salida. Por eso, en esta arquitectura no se incluye una capa pooling. Si se usara una arquitectura basada en CNN o solo embeddings, la capa pooling sería fundamental para resumir la información de la secuencia.

¿Qué pasa si la agrego?

Si intento poner una capa pooling entre el Embedding y la RNN, el modelo no funcionará correctamente, porque la pooling elimina la dimensión temporal que la RNN necesita.

```
# 6. Definición del modelo RNN
modelo = Sequential()
# Capa de embedding: convierte enteros en vectores densos
modelo.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=8, input_length=max_len))
# Capa RNN simple
modelo.add(SimpleRNN(units=16, activation='tanh'))
# Capa densa de salida para clasificación binaria
modelo.add(Dense(1, activation='sigmoid'))
```

```
# 7. Compilación del modelo
modelo.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# 8. Construir el modelo explícitamente para que summary muestre los parámetros correctamente
modelo.build(input_shape=(None, max_len))
```

```
# 9. Resumen de la arquitectura
modelo.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 5, 8)	224
simple_rnn_2 (SimpleRNN)	(None, 16)	400
dense_3 (Dense)	(None, 1)	17

```
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 B)
```

```
# 10. Entrenamiento del modelo
modelo.fit(entradas, etiquetas, epochs=10, verbose=1)
```

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.6000 - loss: 0.6897
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.6000 - loss: 0.6857
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.6000 - loss: 0.6816
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.6000 - loss: 0.6775
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6000 - loss: 0.6733
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.6000 - loss: 0.6690
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.7000 - loss: 0.6646
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.7000 - loss: 0.6601
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8000 - loss: 0.6555
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - accuracy: 0.8000 - loss: 0.6507
<keras.src.callbacks.history.History at 0x7ed54edc1a50>
```

```
# 11. Prueba con una oración nueva
nuevos_comentarios = ["me gustó mucho", "terrible atención"]
```

```
secuencias_nuevas = tokenizer.texts_to_sequences(nuevos_comentarios)
entradas_nuevas = pad_sequences(secuencias_nuevas, maxlen=max_len, padding='post')
predicciones = modelo.predict(entradas_nuevas)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
```

```
# 12. Mostrar predicciones
for comentario, pred in zip(nuevos_comentarios, predicciones):
    print(f"Comentario: '{comentario}' -> Sentimiento positivo: {pred[0]:.2f}")
```

```
Comentario: 'me gustó mucho' -> Sentimiento positivo: 0.48
Comentario: 'terrible atención' -> Sentimiento positivo: 0.49
```

Conclusiones y Recomendaciones

Conclusiones:

El modelo de red neuronal recurrente (RNN) implementado para el análisis de sentimientos en comentarios de productos logró entrenar correctamente y alcanzó una precisión de hasta **0.80** en el conjunto de entrenamiento. La disminución progresiva de la función de pérdida indica que ***el modelo fue capaz de aprender patrones a partir de los datos proporcionados.***

Sin embargo, al evaluar el modelo con nuevos comentarios, ***las predicciones obtenidas estuvieron muy cercanas a 0.5, lo que refleja que el modelo no está completamente seguro al clasificar estos ejemplos como positivos o negativos.*** Esto es esperable debido a que ***el conjunto de datos utilizado para el entrenamiento es muy pequeño*** y las frases de prueba contienen palabras o estructuras que no estaban presentes en los datos originales.

Recomendaciones:

Ampliar el conjunto de datos de entrenamiento, incorporando una mayor cantidad y variedad de comentarios, para que el modelo pueda aprender patrones más generales y robustos.

Realizar un preprocesamiento más avanzado del texto, como la eliminación de palabras vacías (stopwords), normalización y uso de embeddings preentrenados, para mejorar la representación de los comentarios. ***Probar con un mayor número de épocas de entrenamiento si se dispone de más datos,*** para permitir que el modelo ajuste mejor sus parámetros. Evaluar el modelo con frases de prueba que sean más similares a las del entrenamiento, o bien, asegurarse de que el vocabulario de los datos de prueba esté bien representado en el entrenamiento.