Universidad Experimental Nacional del Táchira
Vicerectorado Académico
Decanato de Docencia
Departamento de Ingeniería Electrónica
Redes neuronales y lógica difusa- 0233809T

Red Neuronal Recurrente

Autor:
Porras Axel C.I 29545523

San Cristóbal, enero de 2024

Importación de librerías

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense
import numpy as np
```

Este bloque de código es bastante sencillo, se encarga de importar todos los requerimientos para poder realizar de forma sencilla el proceso de construcción de la red LSTM, cada una de ellas tiene las siguientes funcionalidades:

- tensorflow y keras: Proporcionan las herramientas para construir y entrenar redes neuronales.
- Tokenizer y pad_sequences: Herramientas para convertir texto en números y normalizar las secuencias de entrada.
- Embedding, LSTM, Dense: Componentes principales del modelo neuronal.
- numpy: Biblioteca para manejar cálculos numéricos y manipulación de datos.

Cargado del archivo de texto con los datos de entrenamiento

```
# Cargar el contenido del archivo
def cargar_texto(ruta):
    with open(ruta, 'r', encoding='utf-8') as archivo:
        return archivo.read()

texto = cargar_texto('dataset.txt')
```

Toda red neuronal requiere de un set de datos necesarios para el correcto funcionamiento de la misma, es por esto que en el archivo dataset.txt se encuentra un texto corto para entrenar la red y poder producir texto de forma natural y creíble. Es importante recalcar, que de ser más largo el set se verían mejores resultados ya que la red tendría mucho más contexto sobre como continuar un texto.

Tokenizador de palabras y conteo de las mismas

```
# Configurar el tokenizador y contar palabras
def preparar_tokenizador(texto):
    tokenizador = Tokenizer()
    tokenizador.fit_on_texts([texto])
    return tokenizador

tokenizador = preparar_tokenizador(texto)
vocab_size = len(tokenizador.word_index) + 1
```

Las redes neuronales no pueden procesar texto directamente, por lo tanto, esta función se encarga de generar una clase de índices para poder entregarle los datos de forma adecuada a la red para su entrenamiento. Así en base a las librerías importadas nos encargamos “normalizar” los datos para su correcto funcionamiento. Cada función u objeto de este bloque tiene el siguiente funcionamiento específico:

- Tokenizer(): Inicializa el objeto tokenizador.
- fit_on_texts: Genera un diccionario de palabras y sus índices.
- word_index: Devuelve el diccionario de palabras-token.

Crear secuencias para entrenamiento

```
# Crear secuencias para entrenamiento
def generar_secuencias(texto, tokenizador):
    secuencias = []
    for i in range(1, len(texto)):
        secuencia = texto[:i + 1]
        secuencias.append(secuencia)
    return pad_sequences(tokenizador.texts_to_sequences(secuencias))

secuencias = generar_secuencias(texto, tokenizador)
X, y = secuencias[:, :-1], secuencias[:, -1]
```

El objetivo de este paso es crear secuencias de texto (subsecuencias) que se utilizarán como entradas para el modelo. Estas secuencias representan cómo el modelo aprenderá a predecir la siguiente palabra basándose en un contexto dado.

Supongamos que el contenido del texto en el archivo es:

El sol brilla fuerte

Cada letra del texto se considera individualmente durante la iteración.

- Iteración 1: text[:2] → "El"
- Iteración 2: text[:3] → "El "
- Iteración 3: text[:4] → "El s"
- Iteración 4: text[:5] → "El so"
- Iteración 5: text[:6] → "El sol"
- ...
- Iteración final: text[:21] → "El sol brilla fuerte"

Construir y entrenar el modelo

```
# Definir y compilar el modelo
def construir_modelo(vocab_size, input_length):
    modelo = tf.keras.Sequential([
        Embedding(input_dim=vocab_size, output_dim=64, input_length=input_length),
        LSTM(100),
        Dense(vocab_size, activation='softmax')
    ])
    modelo.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
    return modelo

modelo = construir_modelo(vocab_size, X.shape[1])

# Entrenar el modelo
modelo.fit(X, y, epochs=20, verbose=1)
```

Mediante este proceso se tomaron distintas fuentes para ajustar de forma adecuada el modelo, dándole todos los argumentos necesarios para que según los requerimientos del problema funcione de la forma más óptima posible. Esto se realiza de la siguiente forma:

- Capa Embedding: Mapea cada palabra (índice) a un vector denso de tamaño fijo, lo que permite que el modelo entienda relaciones semánticas.
    1. input_dim: Tamaño del vocabulario.
    2. output_dim: Dimensión de los vectores de palabra.
    3. input_length: Longitud de las secuencias de entrada.

- Capa LSTM: Procesa secuencias aprendiendo dependencias temporales (contexto) en el texto.
    1. 100 unidades: Indica el tamaño de la memoria interna de la LSTM.

2. Capa Dense: Produce la probabilidad para cada palabra en el vocabulario usando softmax.

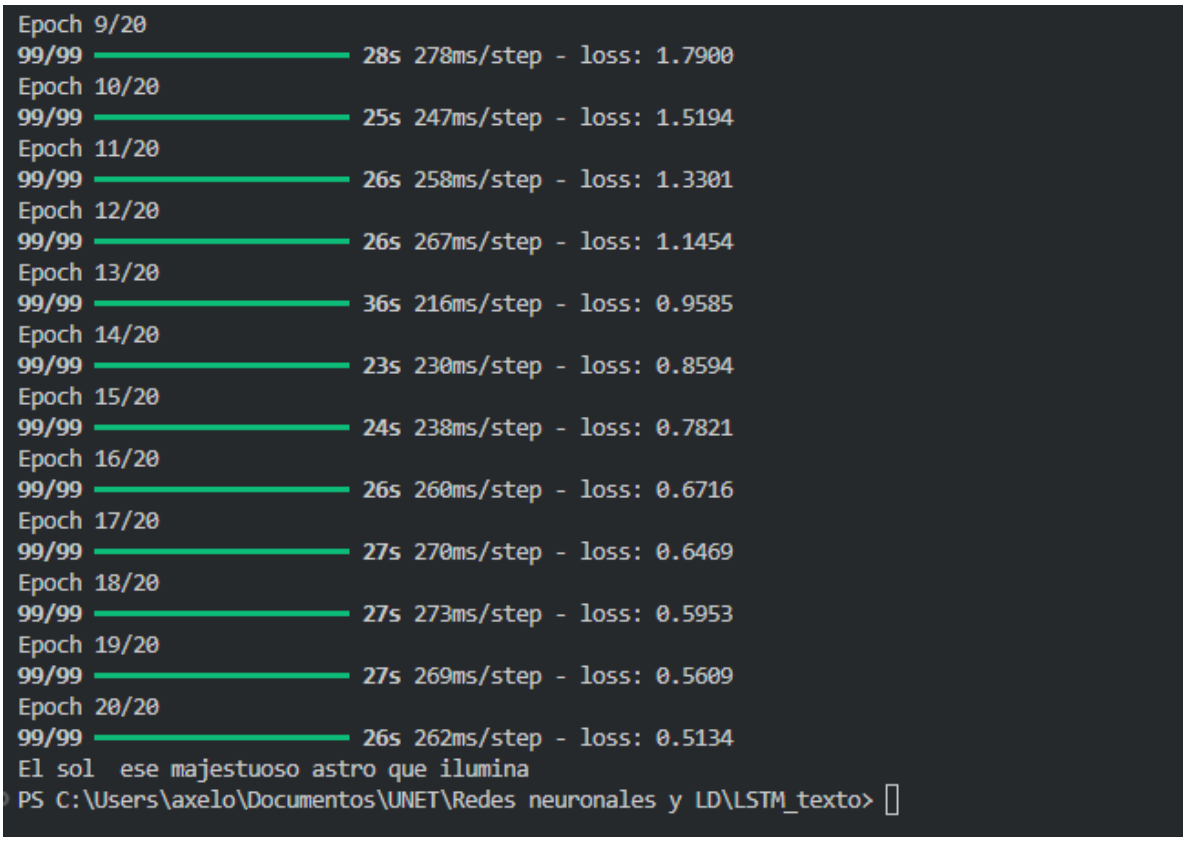
- El modelo se compila con:
    1. sparse_categorical_crossentropy: Métrica de pérdida para clasificación de múltiples categorías.
    2. adam: Optimizador para ajustar los pesos del modelo.

Generación de texto a partir de un texto inicial como contexto

```
# Generar texto a partir de un texto inicial
def generar_texto(modelo, tokenizador, texto_inicial, num_palabras, longitud_max):
    for _ in range(num_palabras):
        secuencia = tokenizador.texts_to_sequences([texto_inicial])[0]
        secuencia = pad_sequences([secuencia], maxlen=longitud_max, padding='pre')
        prediccion = np.argmax(modelo.predict(secuencia, verbose=0))
        palabra_predicha = tokenizador.index_word.get(prediccion, '')
        texto_inicial += ' ' + palabra_predicha
    return texto_inicial

texto_generado = generar_texto(modelo, tokenizador, "El sol ", 5, X.shape[1])
print(texto_generado)
```

Esta fase, tal cual como su nombre lo explica es la utilización de la red previamente entrenada para poder generar un texto en este caso de 5 palabras más de las 2 que ya se tienen para completar la frase. Se utilizan el tokenizador, las secuencias y en base al modelo entrenado se comienza a generar la palabra adecuada para los datos correspondientes, dando así un resultado como el siguiente:



Es importante recalcar que con el número de épocas seleccionado la duración del entrenamiento no es tan corta como en otros modelos de redes neuronales. Por lo que con paciencia se debe esperar el final del mismo para continuar.

Conclusiones

Es interesante ver los resultados obtenidos con este tipo de redes y como debido a la retroalimentación que cada capa le provee a la otra, se puede contextualizar de mucha mejor forma un problema y generar respuestas coherentes para en este caso elementos como posibles conversaciones, o en su defecto de forma más introductoria como en este problema, completar un texto. Obteniendo así un avance importante en el cual no solo importan los pesos de las posibles entradas para tomar decisiones sino a su vez el contexto anterior para que estas decisiones puedan ser más completas e informadas.