

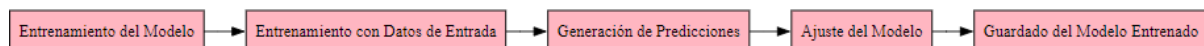
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de ciencias y sistemas
Inteligencia Artificial 1

Chat Bot con tensorflow seq2seq

Nombre	Carnet
Brayan Alexander Mejia Barrientos	201900576
Roberto Carlos Gomez Donis	202000544
Sergio Mynor David Felipe Zapeta	200715274

Descripción General

Este proyecto consiste en la creación de un chatbot utilizando técnicas de procesamiento de lenguaje natural (NLP) y redes neuronales recurrentes (RNNs) con LSTM (Long Short-Term Memory). El chatbot está diseñado para aprender interacciones de texto a partir de un conjunto de datos de entrenamiento en formato JSON, y puede predecir respuestas a las preguntas que el usuario haga.



Requerimientos

Python: 3.12.6

Librerías:

TensorFlow: Para construir y entrenar el modelo de redes neuronales.

NumPy: Para operaciones numéricas.

JSON: Para la carga y almacenamiento de datos de entrenamiento.

Teoría del modelo:

Entrada del Usuario (Secuencia de Entrada): El chatbot recibe una secuencia de entrada, que es la pregunta del usuario. Esta secuencia puede ser una frase o conjunto de palabras.

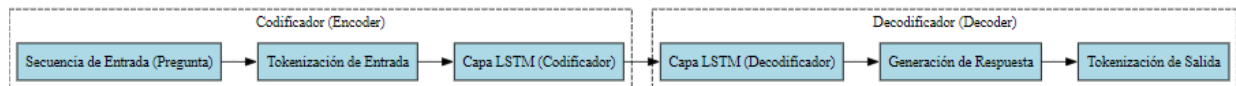
Tokenización de Entrada: La secuencia de entrada se tokeniza, es decir, se convierte en una secuencia de números enteros que representan palabras o tokens.

Capa LSTM (Codificador): El codificador utiliza una capa LSTM (Long Short-Term Memory) para procesar la secuencia de entrada. LSTM es adecuado para trabajar con secuencias de texto, ya que puede aprender dependencias a largo plazo.

Capa LSTM (Decodificador): El decodificador recibe la salida del codificador, procesando esta información para generar la secuencia de salida, que será la respuesta del chatbot.

Generación de Respuesta: A partir de la información procesada, el decodificador genera una distribución de probabilidad sobre las posibles palabras de salida.

Tokenización de Salida: La secuencia generada por el decodificador se convierte de nuevo en palabras o tokens legibles, lo que permite que el chatbot muestre la respuesta al usuario.



Estructura del Proyecto

Carga y Almacenamiento de Datos:

Los datos de entrenamiento se cargan desde un archivo JSON (`training_data.json`) que contiene pares de preguntas y respuestas. Estos datos son utilizados para entrenar el modelo del chatbot.

También se permite agregar nuevos datos de entrenamiento mediante una interfaz interactiva.

Modelo de Red Neuronal:

El modelo está basado en LSTM (Long Short-Term Memory), una arquitectura de red neuronal recurrente que se usa comúnmente para tareas de procesamiento de lenguaje natural.

La red neuronal toma las secuencias de texto de entrada y salida (pregunta y respuesta), las tokeniza y las convierte en vectores de números enteros para ser procesadas por la red.

Funcionalidades:

Entrenamiento: Se entrena el modelo utilizando los datos cargados. Después del entrenamiento, el modelo se guarda en un archivo HDF5 para su reutilización.

Interacción con el chatbot: Permite que los usuarios interactúen con el chatbot para obtener respuestas basadas en las interacciones previas almacenadas.

Guardado y carga de modelo: El modelo entrenado puede ser guardado y cargado para evitar la pérdida de progreso. Si no existe un modelo previamente entrenado, se creará uno nuevo al iniciar el chatbot.

Explicación Detallada del Código

1. Carga de Datos de Entrenamiento

Función `load_training_data`

```
def load_training_data(file_path='training_data.json'):
    try:
        # Abre el archivo en modo lectura ('r')
```

```

        with open(file_path, 'r') as file:
            # Carga el contenido del archivo JSON y lo convierte en
            un objeto Python (lista, diccionario, etc.)
            return json.load(file)
    except FileNotFoundError:
        # Si no se encuentra el archivo, se maneja el error y se
        retorna una lista vacía
        return []

```

Objetivo: Esta función carga los datos de entrenamiento desde un archivo JSON.:

La función usa `open(file_path, 'r')` para abrir el archivo de datos en modo de lectura.

Usa `json.load(file)` para cargar el contenido del archivo JSON y convertirlo en una estructura de datos de Python (normalmente una lista de diccionarios).

Si el archivo no se encuentra, se atrapa la excepción `FileNotFoundError` y retorna una lista vacía.

Función `save_training_data`

```

def save_training_data(training_data,
file_path='training_data.json'):
    # Abre el archivo en modo escritura ('w')
    with open(file_path, 'w') as file:
        # Guarda los datos de entrenamiento en formato JSON, con una
        indentación de 4 espacios para mejor legibilidad
        json.dump(training_data, file, indent=4)

```

Objetivo: Esta función guarda los datos de entrenamiento en un archivo JSON.

Se usa `open(file_path, 'w')` para abrir el archivo en modo escritura.

`json.dump(training_data, file, indent=4)` convierte el objeto `training_data` en formato JSON y lo guarda en el archivo con una indentación de 4 espacios, lo que facilita la lectura.

2. Creación y Configuración del Modelo

```

def create_model(input_shape=(9,), vocab_size=1000,
embedding_dim=128, target_len=9):

```

```

    # Crea una capa de entrada con la forma especificada (dimensión
    de la entrada)
    input_layer = Input(shape=input_shape)
    # Crea una capa de embedding que convierte las secuencias de
    índices en vectores densos
    embedding = Embedding(input_dim=vocab_size,
    output_dim=embedding_dim)(input_layer)
    # Añade una capa LSTM de 256 unidades, la salida es una
    secuencia de vectores
    lstm = LSTM(256, return_sequences=True)(embedding)
    # Añade una segunda capa LSTM para capturar más relaciones entre
    las secuencias
    lstm_2 = LSTM(256, return_sequences=True)(lstm)
    # Añade una capa densa para generar una distribución de
    probabilidad sobre el vocabulario
    dense = Dense(vocab_size, activation='softmax')(lstm_2)
    # Define el modelo con la capa de entrada y la capa de salida
    model = Model(inputs=input_layer, outputs=dense)
    # Compila el modelo especificando el optimizador, la función de
    pérdida y las métricas
    model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])
    return model

```

Objetivo: Esta función crea el modelo de red neuronal utilizando Keras.

`Input(shape=input_shape)` define la entrada del modelo, que en este caso tiene una forma de 9 elementos.

`Embedding(input_dim=vocab_size, output_dim=embedding_dim)` convierte los índices de palabras en vectores densos (embedding).

Se agregan dos capas LSTM para procesar secuencias de texto.

`return_sequences=True` indica que las capas LSTM devolverán una secuencia de resultados, no solo el último valor.

La capa densa con activación `softmax` genera probabilidades de las palabras en el vocabulario.

El modelo se compila con el optimizador `adam`, la función de pérdida `categorical_crossentropy` y la métrica de precisión.

3. Preprocesamiento de Datos

```

tokenizer = tf.keras.preprocessing.text.Tokenizer()
input_texts = [pair['input'] for pair in training_data]
target_texts = [pair['output'] for pair in training_data]
# Ajusta el tokenizador a los textos de entrada y salida
tokenizer.fit_on_texts(input_texts + target_texts)
# Convierte los textos de entrada y salida en secuencias de números
enteros
input_sequences = tokenizer.texts_to_sequences(input_texts)
target_sequences = tokenizer.texts_to_sequences(target_texts)
# Rellena las secuencias para que todas tengan la misma longitud
input_sequences = pad_sequences(input_sequences,
maxlen=max_input_len, padding='post')
target_sequences = pad_sequences(target_sequences,
maxlen=max_input_len, padding='post')

```

Objetivo: Preprocesar los textos de entrada y salida para convertirlos en secuencias numéricas que puedan ser procesadas por el modelo.:

`Tokenizer()` crea un objeto que ayudará a convertir los textos en secuencias de números enteros.

`fit_on_texts(input_texts + target_texts)` ajusta el tokenizador a los textos de entrada y salida combinados, creando un índice de palabras.

`texts_to_sequences(input_texts)` convierte cada palabra en un número entero basado en el índice creado por el tokenizador.

`pad_sequences` asegura que todas las secuencias tengan la misma longitud (usando relleno al final si es necesario).

4. Entrenamiento del Modelo

```

def train_model(epochs=10):
    global model
    input_texts = [pair['input'] for pair in training_data]
    target_texts = [pair['output'] for pair in training_data]
    # Ajusta el tokenizador a los textos de entrada y salida
    tokenizer.fit_on_texts(input_texts + target_texts)
    # Convierte los textos a secuencias
    input_sequences = tokenizer.texts_to_sequences(input_texts)
    target_sequences = tokenizer.texts_to_sequences(target_texts)
    # Rellena las secuencias para que tengan la misma longitud

```

```

    input_sequences = pad_sequences(input_sequences,
maxlen=max_input_len, padding='post')
    target_sequences = pad_sequences(target_sequences,
maxlen=max_input_len, padding='post')
    # Convierte las secuencias de salida a formato one-hot
    target_sequences_one_hot = np.array(
        [to_categorical(seq, num_classes=len(tokenizer.word_index) +
1) for seq in target_sequences]
    )
    # Entrena el modelo utilizando las secuencias de entrada y
salida
    model.fit(input_sequences, target_sequences_one_hot,
epochs=epochs, batch_size=1)
    print(f"Entrenamiento completado por {epochs} épocas.")
    # Guarda el modelo entrenado
    save_model(model)

```

Objetivo: Entrenar el modelo con los datos de entrada y salida preprocesados.

Los textos se tokenizan y se convierten en secuencias numéricas.

`to_categorical` convierte las secuencias de salida en un formato one-hot (vector binario).

El modelo se entrena con `model.fit()`, donde se especifica el número de épocas y el tamaño del lote.

Al finalizar el entrenamiento, se guarda el modelo.

5. Interacción con el Chatbot

```

def chat_with_bot(input_text):
    # Convierte el texto de entrada en una secuencia numérica
    input_seq = tokenizer.texts_to_sequences([input_text])
    # Rellena la secuencia para que tenga la misma longitud que las
entradas del modelo
    input_seq = pad_sequences(input_seq, maxlen=max_input_len,
padding='post')
    # Predice la secuencia de salida con el modelo
    prediction = model.predict(input_seq)
    # Obtiene el índice de la palabra con la mayor probabilidad
    predicted_sequence = np.argmax(prediction, axis=-1)[0]
    # Convierte la secuencia de índices en palabras
    response = []

```

```

for word_index in predicted_sequence:
    for word, index in tokenizer.word_index.items():
        if index == word_index:
            response.append(word)
            break
# Devuelve la respuesta como una cadena de texto
return ' '.join(response)

```

Objetivo: Interactuar con el chatbot, donde se toma un texto de entrada, se procesa, y se obtiene una respuesta.

La entrada del usuario se convierte en una secuencia numérica.

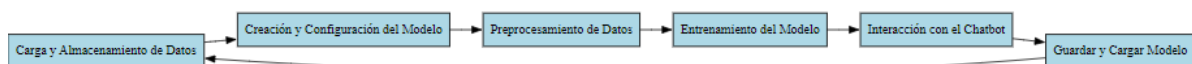
Se usa el modelo para predecir la secuencia de salida.

`np.argmax(prediction, axis=-1)` obtiene el índice del valor máximo en la predicción, lo que corresponde a la palabra más probable.

La secuencia de índices se convierte de nuevo a palabras usando el tokenizador.

Funciones del Menú Interactivo

1. **Agregar Datos de Entrenamiento:** Permite al usuario agregar nuevas interacciones de texto (entrada y salida).
2. **Entrenar el Modelo:** Permite entrenar el modelo con los datos de entrenamiento cargados.
3. **Chatear con el Chatbot:** Permite al usuario interactuar con el chatbot en tiempo real.
4. **Guardar el Modelo:** Permite guardar el modelo entrenado en un archivo `.h5` para su reutilización.




El archivo que se utiliza para el entrenamiento es el siguiente:


```
opcion3.py x training_data.json x
179     "input": "cual es tu opinion sobre el futuro de la tecnologia?",
180     "output": "creo que la tecnologia seguira avanzando rapidamente y transformando el mundo."
181 },
182 {
183     "input": "como te sientes con respecto a la tecnologia?",
184     "output": "me siento emocionado por las posibilidades que ofrece."
185 },
186 {
187     "input": "que opinas de las redes sociales?",
188     "output": "creo que son una buena manera de mantenerse conectado, pero deben usarse con responsabilidad."
189 },
190 {
191     "input": "cuanto tiempo pasas en internet al dia?",
192     "output": "paso varias horas al dia en internet, especialmente para aprender."
193 },
194 {
195     "input": "que eres?",
196     "output": "soy un chatbot que aprende el lenguaje"
197 },
198 {
199     "input": "me puedes ayudar?",
200     "output": "si claro en que te puedo ayudar"
201 },
202 {
203     "input": "que sabes?",
204     "output": "se solo lo que me ense\u00f1an"
```

debe de tener el mismo nombre y el el mismo formato, con forme se le amplie mas este json mejores seran sus respuestas y mientras mas se entrene entendera mejor el lenguaje humano:

El modelo ya entrenado queda en un archivo con extencion .h5

 chatbot_model.h5