Chatbot using Seq2Seq with LSTM

Overview

This project implements a chatbot based on a sequence-to-sequence (Seq2Seq) architecture with Long Short-Term Memory (LSTM) units. It is designed to process input sentences, understand their meaning, and generate appropriate responses. The model uses an encoder-decoder setup with embedded layers for feature representation and LSTM for sequential processing.

Features

Implements Seq2Seq architecture using LSTM layers.
Trains the model to generate human-like responses from the input data.
Utilizes word embeddings for richer representation of text.
Processes and tokenizes input data for optimal performance.

System Requirements

Programming Language: Python 3.8+
Libraries:
TensorFlow/Keras (2.x+)
NumPy
Pandas
scikit-learn
NLTK

Hardware:

Recommended: NVIDIA GPU with CUDA for faster training.
Minimum: CPU with sufficient RAM (16GB recommended).

Dataset

Name: Cornell Movie Dialogs Dataset
Description: A collection of movie dialogs to train conversational models.
Usage: Preprocessed to create paired sequences of questions and responses.

Architecture Details

Encoder

Takes input sequences, encodes them into fixed-size states using LSTM.
Components:-
Embedding Layer: Converts word indices into dense vectors.
LSTM Layer: Processes embedded input sequentially.

code:

```
encoder_inputs = Input(shape=(None,))
encoder_embedding = Embedding(vocab_size_input, embedding_dim)(encoder_inputs)
encoder_lstm = LSTM(units, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]
```

Decoder
Decodes the encoded state into the target sequence.
Components:-
Embedding Layer: Converts target word indices into dense vectors.
LSTM Layer: Sequentially predicts output tokens.
Dense Layer: Maps LSTM outputs to vocabulary probabilities.

code:
```
decoder_inputs = Input(shape=(None,))
decoder_embedding = Embedding(vocab_size_output, embedding_dim)(decoder_inputs)
decoder_lstm = LSTM(units, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(vocab_size_output, activation='softmax')
output = decoder_dense(decoder_outputs)
```

Compilation and Training
Loss Function: Categorical crossentropy.
Optimizer: Adam for adaptive learning.
Data Split: 80% training, 20% validation.
Batch Size: Adjustable based on system resources.

Challenges
Resource Exhaustion:-
Addressed by reducing batch size and vocabulary size or applying truncation to input/output sequences.
High Dimensionality:-
Optimized embedding dimensions and LSTM units to balance performance and memory requirements.

Usage Instructions
Preprocess Data:-
Tokenize sentences, build vocabulary, and pad sequences to ensure uniform input length.

code:
```
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(sentences)
sequences = tokenizer.texts_to_sequences(sentences)
```

Train Model:
Fit the Seq2Seq model on preprocessed training data.

code:
```
model.fit(
    [encoder_input_data, decoder_input_data],
    decoder_output_data,
    batch_size=batch_size,
    epochs=20,
    validation_split=0.2
)
```

Inference:
Convert user input into tokenized sequences and use the trained model to generate responses.