Final_Chatbot_Model2_Kaustubh_Prakash_Deolase.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

Share

Files

- ..
- models
  - best_seq2seq.keras
  - decoder_model.keras
  - encoder_model.keras
  - seq2seq_model.keras
- sample_data
- dialogs.txt

Disk 73.28 GB available

Final_Project_Chatbot_Model_by_Kaustubh_Prakash_Deolase

==========

ChatBot_Model_Seq2Seq_Chatbot_(LSTM)_by_Kaustubh_Prakash_Deolase

==========

[16] ✓ 0s

```
import os
import re
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import (Embedding, LSTM, Dense, LayerNormalization, Dropout, TextVectorization)
import matplotlib.pyplot as plt
```

==========

TEXT CLEANING FUNCTION

==========

[17] ✓ 0s

```
def clean_text(text: str) -> str:
    """Basic text cleaning."""
    if not isinstance(text, str):
        text = str(text)
    text = text.lower()
    text = re.sub(r"-", " ", text)
    text = re.sub(r"([?.!,])", r" \1 ", text)
    text = re.sub(r"([0-9])", r" \1 ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

==========

LOAD AND PREPROCESS DIALOGS

==========

[18] ✓ 0s

```
def load_dialogs(path="dialogs.txt"):
    if not os.path.exists(path):
        raise FileNotFoundError(f"dialogs file not found at: {path}")
    df = pd.read_csv(path, sep="\t", names=["question", "answer"], encoding="utf-8")
    df.dropna(inplace=True)
    df["question"] = df["question"].map(clean_text)
    df["answer"] = df["answer"].map(clean_text)
    return df
```

==========

TEXT VECTORIZATION

==========

[19] ✓ 0s

```
def build_vectorizer(texts, max_tokens=5000, seq_len=30):
    vectorizer = TextVectorization(
        max_tokens=max_tokens,
        output_mode="int",
        output_sequence_length=seq_len,
        standardize=None  # We already cleaned text
    )
    vectorizer.adapt(texts)
    return vectorizer
```

==========

ENCODER MODEL

==========

[20] ✓ 0s

```
class Encoder(tf.keras.Model):
    def __init__(self, units, embedding_dim, vocab_size):
        super().__init__()
        self.embedding = Embedding(vocab_size, embedding_dim, mask_zero=True)
        self.norm = LayerNormalization()
        self.dropout = Dropout(0.4)
        self.lstm = LSTM(units, return_sequences=True, return_state=True, dropout=0.4)

    def call(self, x, training=False):
        x = self.embedding(x)
        x = self.norm(x, training=training)
        x = self.dropout(x, training=training)
        outputs, h, c = self.lstm(x, training=training)
        return outputs, h, c
```

==========

DECODER MODEL

==========

[21] ✓ 0s

```
class Decoder(tf.keras.Model):
    def __init__(self, units, embedding_dim, vocab_size):
```

```
        super().__init__()
        self.embedding = Embedding(vocab_size, embedding_dim, mask_zero=True)
        self.norm1 = LayerNormalization()
        self.lstm = LSTM(units, return_sequences=True, return_state=True, dropout=0.4)
        self.norm2 = LayerNormalization()
        self.dropout = Dropout(0.4)
        self.fc = Dense(vocab_size, activation="softmax")

    def call(self, x, initial_state=None, training=False):
        x = self.embedding(x)
        x = self.norm1(x, training=training)
        x = self.dropout(x, training=training)
        outputs, h, c = self.lstm(x, initial_state=initial_state, training=training)
        outputs = self.norm2(outputs, training=training)
        outputs = self.dropout(outputs, training=training)
        logits = self.fc(outputs)
        return logits, h, c
```

==========

BUILD TRAINING MODEL

==========

[22] ✓ 0s

```
def build_training_model(encoder, decoder, seq_len):
    enc_inp = tf.keras.Input(shape=(seq_len,), name="encoder_input")
    dec_inp = tf.keras.Input(shape=(seq_len,), name="decoder_input")
    enc_out, h, c = encoder(enc_inp)
    dec_out, _, _ = decoder(dec_inp, initial_state=[h, c])
    model = tf.keras.Model([enc_inp, dec_inp], dec_out)
    return model
```

==========

BUILD INFERENCE MODELS (for chatting)

==========

[23] ✓ 0s

```
def build_inference_models(encoder, decoder, seq_len, units):
    enc_input = tf.keras.Input(shape=(seq_len,), name="enc_input")
    _, h, c = encoder(enc_input)
    enc_model = tf.keras.Model(enc_input, [h, c])

    dec_input = tf.keras.Input(shape=(1,), name="dec_input")
    h_in = tf.keras.Input(shape=(units,), name="h_in")
    c_in = tf.keras.Input(shape=(units,), name="c_in")
    dec_out, h_out, c_out = decoder(dec_input, initial_state=[h_in, c_in])
    dec_model = tf.keras.Model([dec_input, h_in, c_in], [dec_out, h_out, c_out])
    return enc_model, dec_model
```

==========

GREEDY DECODING

==========

[24] ✓ 0s

```
def greedy_decode(input_text, vectorizer, enc_model, dec_model, id2word, word2id, max_len=20):
    # Convert text to sequence using TextVectorization
    seq = vectorizer([input_text])  # returns tensor (1, seq_len)

    # Encode input sequence to get states
    state_h, state_c = enc_model(seq)

    # Start token id
    start_id = word2id.get('<start>', 1)
    end_id = word2id.get('<end>', 2)

    target_seq = tf.expand_dims([start_id], 0)  # shape (1,1)
    decoded_sentence = []

    for _ in range(max_len):
        output_tokens, h, c = dec_model([target_seq, state_h, state_c])
        sampled_token_index = tf.argmax(output_tokens[0, -1, :]).numpy()
        sampled_word = id2word.get(sampled_token_index, '')

        if sampled_word == '<end>':
            break

        decoded_sentence.append(sampled_word)
        target_seq = tf.expand_dims([sampled_token_index], 0)
        state_h, state_c = h, c

    return ' '.join(decoded_sentence)


```

==========

MAIN TRAINING PIPELINE

==========

[25] ✓ 0s

```
def train_chatbot(data_path="dialogs.txt"):
    # Hyperparameters
    VOCAB_SIZE = 5000
    SEQ_LEN = 30
    EMBED_DIM = 128
    LSTM_UNITS = 256
    EPOCHS = 30
    BATCH_SIZE = 128
    LR = 1e-3

```

```
    # Load data
    df = load_dialogs(data_path)
    df["decoder_input"] = "<start> " + df["answer"]
    df["decoder_target"] = df["answer"] + " <end>"

    # Build vocabulary
    combined_text = pd.concat([
        df["question"], df["decoder_input"], df["decoder_target"]
    ])
    vectorizer = build_vectorizer(combined_text, max_tokens=VOCAB_SIZE, seq_len=SEQ_LEN)

    vocab = vectorizer.get_vocabulary()
    word2id = {t: i for i, t in enumerate(vocab)}
    id2word = {i: t for i, t in enumerate(vocab)}

    # Vectorize sequences
    enc_inp = vectorizer(df["question"])
    dec_inp = vectorizer(df["decoder_input"])
    dec_tar = vectorizer(df["decoder_target"])

    # Build model
    encoder = Encoder(LSTM_UNITS, EMBED_DIM, VOCAB_SIZE)
    decoder = Decoder(LSTM_UNITS, EMBED_DIM, VOCAB_SIZE)
    model = build_training_model(encoder, decoder, SEQ_LEN)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(LR),
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=["accuracy"]
    )

    # Train
    ckpt = tf.keras.callbacks.ModelCheckpoint(
        "models/best_seq2seq.keras", monitor="val_loss", save_best_only=True
    )

    history = model.fit(
        [enc_inp, dec_inp], dec_tar,
        batch_size=BATCH_SIZE,
        epochs=EPOCHS,
        validation_split=0.1,
        callbacks=[ckpt]
    )

    # Save models
    enc_model, dec_model = build_inference_models(encoder, decoder, SEQ_LEN, LSTM_UNITS)
    os.makedirs("models", exist_ok=True)
    model.save("models/seq2seq_model.keras")
    enc_model.save("models/encoder_model.keras")
    dec_model.save("models/decoder_model.keras")

    # Plot metrics
    fig, ax = plt.subplots(1, 2, figsize=(15, 5))
    ax[0].plot(history.history["loss"], label="train_loss", color="red")
    ax[0].plot(history.history["val_loss"], label="val_loss", color="blue")
    ax[0].set_title("Loss")
    ax[0].legend()
    ax[1].plot(history.history["accuracy"], label="train_acc", color="green")
    ax[1].plot(history.history["val_accuracy"], label="val_acc", color="orange")
    ax[1].set_title("Accuracy")
    ax[1].legend()
    plt.show()

    print("✅ Training complete and models saved in 'models/' folder.")
    return vectorizer, enc_model, dec_model, id2word, word2id
```

==========

CHAT LOOP (AFTER TRAINING)

==========

[26] ✓ 0s

```
def chat_loop(vectorizer, enc_model, dec_model, id2word, word2id):
    print("\n🤖 Chatbot is ready! Type 'exit' to quit.\n")
    while True:
        msg = input("You: ").strip()
        if msg.lower() in ["exit", "quit", "bye"]:
            print("Chatbot: Goodbye!")
            break
        reply = greedy_decode(clean_text(msg), vectorizer, enc_model, dec_model, id2word, word2id)
        print("Chatbot:", reply)
```

==========

ENTRY POINT

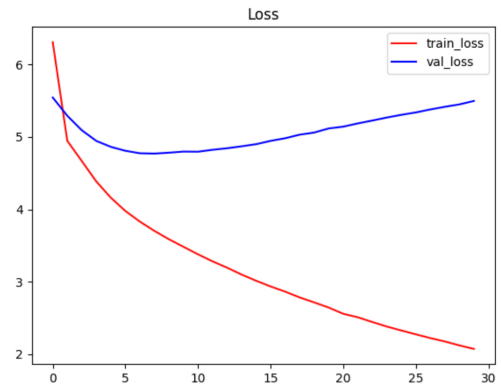
==========

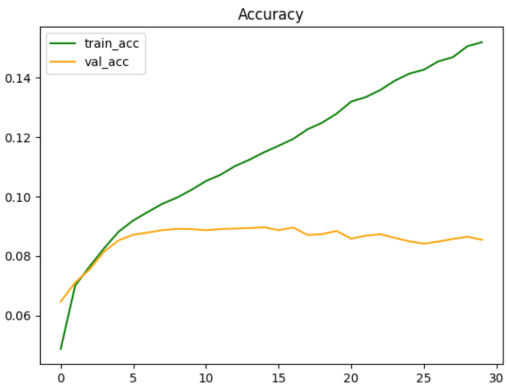
[27] ✓ 56s

```
if __name__ == "__main__":
    vectorizer, enc_model, dec_model, id2word, word2id = train_chatbot("dialogs.txt")
    #chat_loop(vectorizer, enc_model, dec_model, id2word, word2id)
```

```
Epoch 15/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1151 - loss: 2.9962 - val_accuracy: 0.0897 - val_loss: 4.8993
Epoch 16/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1187 - loss: 2.9018 - val_accuracy: 0.0887 - val_loss: 4.9450
Epoch 17/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1195 - loss: 2.8625 - val_accuracy: 0.0896 - val_loss: 4.9812
Epoch 18/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1224 - loss: 2.7705 - val_accuracy: 0.0871 - val_loss: 5.0308
Epoch 19/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.1249 - loss: 2.7118 - val_accuracy: 0.0874 - val_loss: 5.0600
Epoch 20/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.1293 - loss: 2.6244 - val_accuracy: 0.0885 - val_loss: 5.1169
Epoch 21/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.1335 - loss: 2.5278 - val_accuracy: 0.0859 - val_loss: 5.1414
Epoch 22/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.1336 - loss: 2.4935 - val_accuracy: 0.0869 - val_loss: 5.1859
Epoch 23/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.1378 - loss: 2.4160 - val_accuracy: 0.0874 - val_loss: 5.2255
Epoch 24/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.1406 - loss: 2.3588 - val_accuracy: 0.0861 - val_loss: 5.2673
Epoch 25/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1425 - loss: 2.2994 - val_accuracy: 0.0850 - val_loss: 5.3048
Epoch 26/30
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1428 - loss: 2.2534 - val_accuracy: 0.0842 - val_loss: 5.3378
Epoch 27/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.1462 - loss: 2.1978 - val_accuracy: 0.0849 - val_loss: 5.3791
Epoch 28/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.1474 - loss: 2.1679 - val_accuracy: 0.0858 - val_loss: 5.4156
Epoch 29/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.1528 - loss: 2.0927 - val_accuracy: 0.0865 - val_loss: 5.4488
Epoch 30/30
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.1556 - loss: 2.0482 - val_accuracy: 0.0855 - val_loss: 5.4963
```






```
✅ Training complete and models saved in 'models/' folder.
```

==========

ChatBot Converstion:

==========

[28] ✓ 0s

```python
# Function to reply to a single user input
def chatbot_reply(user_input, vectorizer, enc_model, dec_model, id2word, word2id):
    user_input = clean_text(user_input)
    return greedy_decode(user_input, vectorizer, enc_model, dec_model, id2word, word2id)

# Function to simulate a conversation from a list of messages
def print_conversation(messages, vectorizer, enc_model, dec_model, id2word, word2id):
    for text in messages:
        print(f'You: {text}')
        reply = chatbot_reply(text, vectorizer, enc_model, dec_model, id2word, word2id)
        print(f'Bot: {reply}')
        print('=======================')

# ✅ Example usage
if __name__ == "__main__":
    # Make sure the models are loaded / trained
    # vectorizer, enc_model, dec_model, id2word, word2id = train_chatbot("dialogs.txt")

    test_messages = [
        "hi",
        "how are you?",
        "what is your name?",
        "tell me a joke",
        "bye"
    ]

    print_conversation(test_messages, vectorizer, enc_model, dec_model, id2word, word2id)

```

```
You: hi
Bot: i'm not sleeping .
=======================
You: how are you?
Bot: i don't know .
=======================
You: what is your name?
Bot: i don't know .
=======================
You: tell me a joke
Bot: i hope it doesn't rain .
=======================
You: bye
Bot: not yet .
=======================
```