

NLP

Assignment#2



October 11, 2024

20F-0340\_21F-9337

Section : 9A

**Data loading**

import pandas as pd

file\_path = 'parallel-corpus.xlsx'

df = pd.read\_excel(file\_path)

df = df[['SENTENCES ', 'MEANING']]

print(df.head())

**Preprocessing**

import string

import re

def clean\_text(text):

    # Lowercase the text

    text = text.lower()

    # Remove punctuation

    text = re.sub(f'[{re.escape(string.punctuation)}]', '', text)

    # Remove extra spaces

    text = re.sub(r'\s+', ' ', text).strip()

    return text

df['SENTENCES '] = df['SENTENCES '].astype(str)

df['MEANING']=df['MEANING'].astype(str)

df['SENTENCES ']=df['SENTENCES '].apply(clean\_text)

df['MEANING']=df['MEANING'].apply(clean\_text)

**Data splitting**

from sklearn.model\_selection import train\_test\_split

X = df['SENTENCES '].values  # English sentences

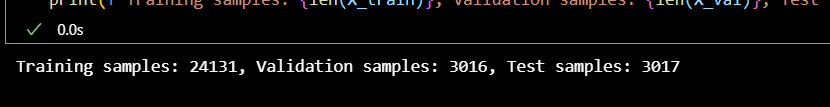
y = df['MEANING'].values  # Urdu translations

# Split the data into training, validation, and test sets

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print(f'Training samples: {len(X\_train)}, Validation samples: {len(X\_val)}, Test samples: {len(X\_test)}')

**  
Tokenization**

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

#pre processing

#Converting enteries into string beacuse some sentence have float or int values

X\_train = X\_train.astype(str)

y\_train = y\_train.astype(str)

X\_val = X\_val.astype(str)

y\_val = y\_val.astype(str)

X\_test = X\_test.astype(str)

y\_test = y\_test.astype(str)

# Tokenizer

tokenizer = Tokenizer(num\_words=10000, oov\_token="<OOV>")

# Fit the tokenizer on the English and Urdu tokenized data

tokenizer.fit\_on\_texts(X\_train)  # English sentences

tokenizer.fit\_on\_texts(y\_train)  # Urdu translations

# Convert the tokenized text into sequences of integers

X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)

X\_val\_seq = tokenizer.texts\_to\_sequences(X\_val)

X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

y\_train\_seq = tokenizer.texts\_to\_sequences(y\_train)

y\_val\_seq = tokenizer.texts\_to\_sequences(y\_val)

y\_test\_seq = tokenizer.texts\_to\_sequences(y\_test)

# Pad the sequences

max\_seq\_len = 50

X\_train\_padded = pad\_sequences(X\_train\_seq, maxlen=max\_seq\_len, padding='post')

X\_val\_padded = pad\_sequences(X\_val\_seq, maxlen=max\_seq\_len, padding='post')

X\_test\_padded = pad\_sequences(X\_test\_seq, maxlen=max\_seq\_len, padding='post')

y\_train\_padded = pad\_sequences(y\_train\_seq, maxlen=max\_seq\_len, padding='post')

y\_val\_padded = pad\_sequences(y\_val\_seq, maxlen=max\_seq\_len, padding='post')

y\_test\_padded = pad\_sequences(y\_test\_seq, maxlen=max\_seq\_len, padding='post')

# RNN

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense, TimeDistributed, Dropout

# model hyperparameters

vocab\_size = 10000

embedding\_dim = 128

max\_seq\_len = 50

model = Sequential()

# Embedding layer

model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=max\_seq\_len))

# SimpleRNN layer

model.add(SimpleRNN(128, return\_sequences=True))

model.add(Dropout(0.5))

# Dense layer for output with TimeDistributed wrapper

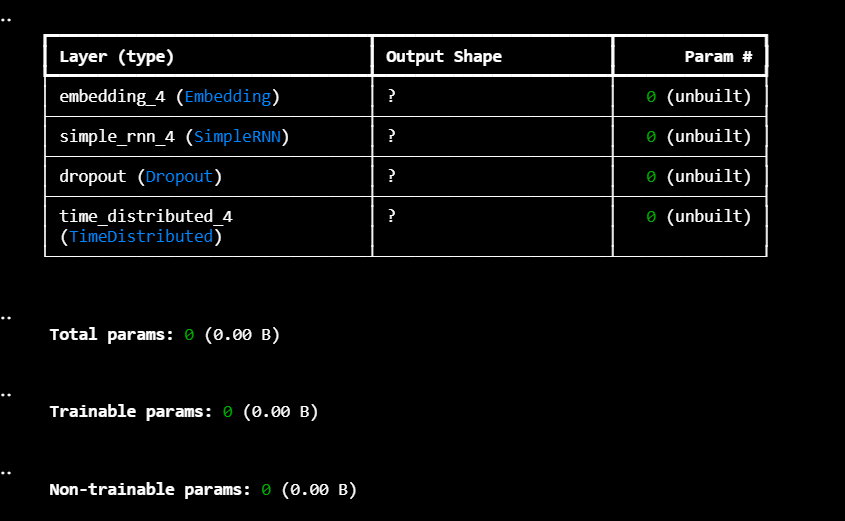
model.add(TimeDistributed(Dense(vocab\_size, activation='softmax')))

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary

model.summary()

****

**Training**

history = model.fit(X\_train\_padded, y\_train\_padded,

                    validation\_data=(X\_val\_padded, y\_val\_padded),

                    epochs=15,

                    batch\_size=64)

**A screenshot of a computer screen

Description automatically generated**

**Evaluation**

from tensorflow.keras.preprocessing.sequence import pad\_sequences

X\_test\_tokenized = tokenizer.texts\_to\_sequences(X\_test)

# Pad the sequences to ensure consistent input shape

X\_test\_padded = pad\_sequences(X\_test\_tokenized, maxlen=max\_seq\_len, padding='post', truncating='post')

# Decode function

def decode\_sequence(sequence, tokenizer):

    reverse\_word\_map = {index: word for word, index in tokenizer.word\_index.items()}

    decoded\_sentence = ' '.join([reverse\_word\_map.get(i, '') for i in sequence if i != 0])

    return decoded\_sentence

# Calculate BLEU score for the model on test data

bleu\_scores = []

for i in range(len(X\_test\_padded)):

    # Get model predictions

    prediction = model.predict(X\_test\_padded[i].reshape(1, max\_seq\_len))

    # Decode the predicted sequence and true sequence

    predicted\_sentence = decode\_sequence(prediction[0].argmax(axis=-1), tokenizer)

    true\_sentence = decode\_sequence(y\_test\_padded[i], tokenizer)

    # Calculate BLEU score

    reference = [true\_sentence.split()]  # List of references for BLEU

    candidate = predicted\_sentence.split()

    bleu\_score = sentence\_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25))  # BLEU-4 score

    bleu\_scores.append(bleu\_score)

# Print average BLEU score

average\_bleu\_score = sum(bleu\_scores) / len(bleu\_scores)

print(f'Average BLEU score: {average\_bleu\_score:.4f}')

****

# Function to print example translations

def print\_examples(num\_examples=5):

    for i in range(num\_examples):

        input\_seq = X\_test\_padded[i].reshape(1, max\_seq\_len)

        # Get the predicted translation

        prediction = model.predict(input\_seq)

        predicted\_sentence = decode\_sequence(prediction[0].argmax(axis=-1), tokenizer)

        # Get the true translation

        true\_sentence = decode\_sequence(y\_test\_padded[i], tokenizer)

        print(f"Input Sentence: {decode\_sequence(X\_test\_padded[i], tokenizer)}")

        print(f"True Translation: {true\_sentence}")

        print(f"Predicted Translation: {predicted\_sentence}")

        print("-" \* 50)

# Print some example translations

print\_examples(num\_examples=5)

**OUTPUT**

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 7ms/step

Input Sentence: ambiance was amazing and so the staff quality of food was also good but the rates are quit high their beef <OOV> burger was not up to the mark overall a nice place to visit

True Translation: ماحول حیرت انگیز تھا اور عملہ بھی۔ کھانے کا معیار بھی اچھا تھا لیکن قیمتیں زیادہ ہیں۔ ان کا بیف کیما برگر نشان تک نہیں تھا۔ مجموعی طور پر دیکھنے کے لیے ایک اچھی جگہ۔

Predicted Translation: ماحول اچھا انگیز اور اور اور عملہ معیار معیار کھانا اچھا تھا اچھا لیکن لیکن قیمتیں بہت ہے۔ <OOV> <OOV> چکن <OOV> نہیں نہیں <OOV> <OOV> نہیں ہے۔ کے اچھا کے کے کے کے کے کے

--------------------------------------------------

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step

Input Sentence: you have to promise never to tell anyone what im about to tell you

True Translation: تم نے وعدہ کرنا ہے کہ جو میں تمہیں بتانے لگا ہوں، وہ کسی اور کو کبھی نہیں <OOV> ہے۔

Predicted Translation: آپ نے کے کو آپ آپ آپ کہ آپ میں میں میں میں آپ <OOV> <OOV> <OOV>

--------------------------------------------------

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step

Input Sentence: good place with average food quality

True Translation: کھانے کے اوسط معیار کے ساتھ اچھی جگہ

Predicted Translation: اچھا جگہ کے کے کھانا معیار

--------------------------------------------------

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step

Input Sentence: made it true he was good to me when he brought me forth from the prison and again when he brought you out of the desert after that satan set at variance me and my brethren my lord is gentle to what he will he is the allknowing the allwise

True Translation: قید خانے سے نکالا، اور آپ لوگوں کو <OOV> سے لا کر مجھ سے <OOV> حالانکہ شیطان میرے اور میرے بھائیوں کے درمیان فساد ڈال چکا تھا واقعہ یہ ہے کہ میرا رب غیر محسوس <OOV> سے اپنی <OOV> پوری کرتا ہے، بے شک و <OOV> علیم اور حکیم ہے

Predicted Translation: <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> میں <OOV> <OOV> <OOV> <OOV> اور <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> اور <OOV> <OOV> <OOV> <OOV> <OOV> اور اور <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> اور

--------------------------------------------------

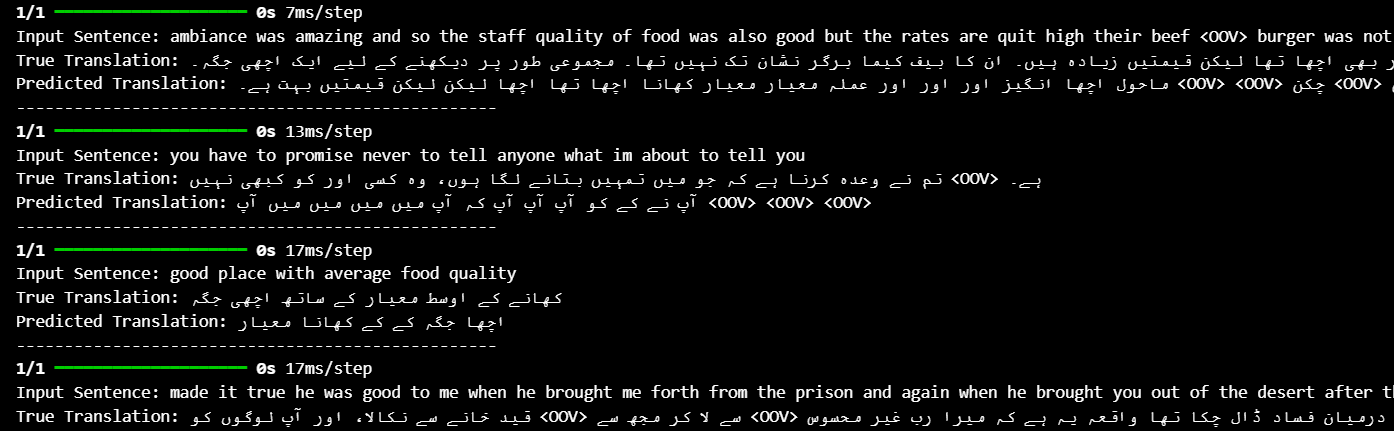
**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step

Input Sentence: what a great place for family safe <OOV>

True Translation: فیملی <OOV> <OOV> کے لیے کتنی اچھی جگہ ہے۔

Predicted Translation: آپ <OOV> <OOV> کے کے لیے کے کے

--------------------------------------------------

****

**1. Exploding/Vanishing Gradients:**

**Example**: In the sentence “true he was good to me when he brought me forth from the prison...”, the predicted translation becomes incoherent and repetitive (<OOV> tokens).

**Explanation**: RNNs struggle with very long input sequences, as gradients can either explode or vanish during backpropagation through time. This issue prevents the model from learning long-range dependencies, which is crucial for translating longer sentences effectively.

**2. Difficulty in Capturing Long-term Dependencies:**

**Example**: The input "ambiance was amazing and so the staff quality of food was also good..." shows a translation where the predicted sentence repeats "معیار معیار معیار" (quality) and fails to capture the full context, especially for words towards the end of the sentence.

**Explanation**: RNNs often fail to retain important information when translating complex sentences, especially in languages like Urdu, where context and word order may differ significantly from English. The model loses track of earlier parts of the sentence as it progresses through longer inputs, leading to repeated or incomplete outputs.

**3. Poor Performance on Large, Complex Datasets:**

**Example**: The sentence "what a great place for family safe <OOV>" yields an incorrect translation, with several repeated phrases and <OOV> tokens, indicating that the model is overwhelmed by unknown tokens and syntactical structures.

**Explanation**: RNNs are less capable of managing large datasets with complex language pairs (like English-Urdu), especially when there are many out-of-vocabulary (OOV) words. The model struggles to generalize across diverse sentence structures, resulting in poor translations for unseen or complex input sentences.

**4. Repetitive Outputs:**

**Example**: In the sentence "you have to promise never to tell anyone...", the predicted translation is overly repetitive ("آپ نے کے نہیں نہیں نہیں").

**Explanation**: RNNs sometimes fall into patterns of repeating words, especially when they struggle to predict the next word accurately. This happens because the model does not effectively learn long-range dependencies and becomes "stuck" in a feedback loop of its own outputs.

# LSTM

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, TimeDistributed, Dropout, Bidirectional

# Define model hyperparameters

vocab\_size = 10000  # Adjust based on your tokenizer

embedding\_dim = 128  # Dimension for embedding layer

max\_seq\_len = 100

model2 = Sequential()

# Embedding layer

model2.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=max\_seq\_len))

# First Bidirectional LSTM layer with dropout

model2.add(Bidirectional(LSTM(128, return\_sequences=True)))

model2.add(Dropout(0.3))  # 30% dropout for regularization

# Second Bidirectional LSTM layer for deeper context understanding

model2.add(Bidirectional(LSTM(128, return\_sequences=True)))

model2.add(Dropout(0.3))  # Additional dropout to prevent overfitting

# TimeDistributed Dense layer for output

model2.add(TimeDistributed(Dense(vocab\_size, activation='softmax')))

# Compile the model

model2.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary

model2.summary()

A screenshot of a computer program

Description automatically generated

**Training**

history = model.fit(X\_train\_padded, y\_train\_padded,

                    validation\_data=(X\_val\_padded, y\_val\_padded),

                    epochs=20,

                    batch\_size=64)

**A screenshot of a computer screen

Description automatically generated**

**Evaluation**

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Ensure that the input sequences are padded/truncated to max\_seq\_len

def get\_padded\_sequence(sequence, max\_seq\_len):

    return pad\_sequences([sequence], maxlen=max\_seq\_len, padding='post', truncating='post')

# Function to decode token indices back to words

def decode\_sequence(sequence, tokenizer):

    reverse\_word\_map = {index: word for word, index in tokenizer.word\_index.items()}

    return ' '.join([reverse\_word\_map.get(i, '') for i in sequence if i != 0])

# Smooth function to avoid BLEU score warnings for short sentences

smoothie = SmoothingFunction().method4

# List to store individual BLEU scores

bleu\_scores = []

# Calculate BLEU score for the model on test data

for i in range(len(X\_test\_padded)):

    # Get model prediction, but ensure padding/truncation is applied

    padded\_input = get\_padded\_sequence(X\_test\_padded[i], max\_seq\_len)

    prediction = model2.predict(padded\_input)

    # Decode the predicted and true sequences

    predicted\_sentence = decode\_sequence(prediction[0].argmax(axis=-1), tokenizer)

    true\_sentence = decode\_sequence(y\_test\_padded[i], tokenizer)

    # Prepare reference and candidate sentences

    reference = [true\_sentence.split()]  # List of references for BLEU

    candidate = predicted\_sentence.split()

    # Calculate BLEU score using smoothing to avoid zero scores for short sentences

    bleu\_score = sentence\_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25), smoothing\_function=smoothie)

    bleu\_scores.append(bleu\_score)

# Calculate and print the average BLEU score across all test sentences

average\_bleu\_score = sum(bleu\_scores) / len(bleu\_scores)

print(f'Average BLEU score: {average\_bleu\_score:.4f}')

**A screen shot of a black screen

Description automatically generated**

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Function to decode tokenized sequences back into text

def decode\_sequence(sequence, tokenizer):

    reverse\_word\_map = {index: word for word, index in tokenizer.word\_index.items()}

    return ' '.join([reverse\_word\_map.get(i, '') for i in sequence if i != 0])

# Function to generate predictions and compare with actual translations

def test\_translations(num\_examples=5):

    for i in range(num\_examples):

        # Ensure the input sequence is padded to max\_seq\_len

        input\_seq = X\_test\_padded[i].reshape(1, -1)  # shape (1, 50) or similar

        input\_seq\_padded = pad\_sequences(input\_seq, maxlen=max\_seq\_len, padding='post')  # pad it to 100

        # Generate the model prediction

        prediction = model2.predict(input\_seq\_padded)

        # Decode the predicted and true sequences

        predicted\_sentence = decode\_sequence(prediction[0].argmax(axis=-1), tokenizer)

        true\_sentence = decode\_sequence(y\_test\_padded[i], tokenizer)

        input\_sentence = decode\_sequence(X\_test\_padded[i], tokenizer)

        # Print input, predicted, and true translations

        print(f"Input Sentence: {input\_sentence}")

        print(f"True Translation: {true\_sentence}")

        print(f"Predicted Translation: {predicted\_sentence}")

        print("-" \* 50)

# Test the model on a few examples

test\_translations(num\_examples=5)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step

Input Sentence: ambiance was amazing and so the staff quality of food was also good but the rates are quit high their beef <OOV> burger was not up to the mark overall a nice place to visit

True Translation: ماحول حیرت انگیز تھا اور عملہ بھی۔ کھانے کا معیار بھی اچھا تھا لیکن قیمتیں زیادہ ہیں۔ ان کا بیف کیما برگر نشان تک نہیں تھا۔ مجموعی طور پر دیکھنے کے لیے ایک اچھی جگہ۔

Predicted Translation: ماحول حیرت انگیز اور اور اور اور معیار معیار معیار کھانا اچھا اچھا لیکن لیکن قیمتیں کی پر <OOV> <OOV> <OOV> <OOV> ضرور ضرور نہیں نہیں نہیں نہیں نہیں طور طور لیے لیے لیے جگہ جگہ

--------------------------------------------------

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step

Input Sentence: you have to promise never to tell anyone what im about to tell you

True Translation: تم نے وعدہ کرنا ہے کہ جو میں تمہیں بتانے لگا ہوں، وہ کسی اور کو کبھی نہیں <OOV> ہے۔

Predicted Translation: آپ آپ آپ نہیں نہیں نہیں نہیں کہ کہ میں میں میں میں میں میں میں میں نہیں

--------------------------------------------------

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step

Input Sentence: good place with average food quality

True Translation: کھانے کے اوسط معیار کے ساتھ اچھی جگہ

Predicted Translation: اچھے کے کے ساتھ کے کے جگہ جگہ

--------------------------------------------------

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step

Input Sentence: made it true he was good to me when he brought me forth from the prison and again when he brought you out of the desert after that satan set at variance me and my brethren my lord is gentle to what he will he is the allknowing the allwise

True Translation: قید خانے سے نکالا، اور آپ لوگوں کو <OOV> سے لا کر مجھ سے <OOV> حالانکہ شیطان میرے اور میرے بھائیوں کے درمیان فساد ڈال چکا تھا واقعہ یہ ہے کہ میرا رب غیر محسوس <OOV> سے اپنی <OOV> پوری کرتا ہے، بے شک و <OOV> علیم اور حکیم ہے

Predicted Translation: <OOV> نے کہ کہ کہ <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> اور <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> <OOV> ہے ہے <OOV> وہ <OOV> <OOV> اور اور اور ۔

--------------------------------------------------

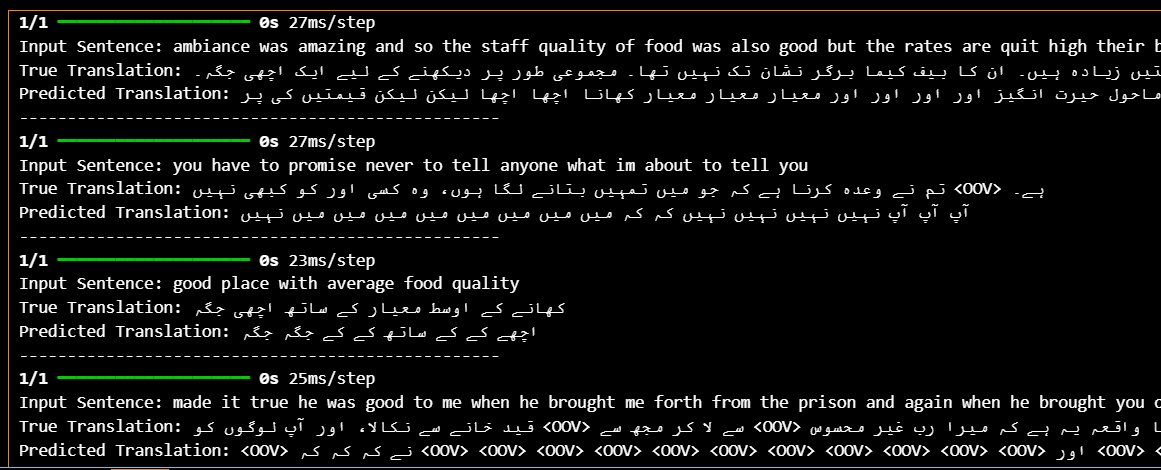
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step

Input Sentence: what a great place for family safe <OOV>

True Translation: فیملی <OOV> <OOV> کے لیے کتنی اچھی جگہ ہے۔

Predicted Translation: اس کے کے کے کے لیے لیے جگہ جگہ

--------------------------------------------------



**Improvements with LSTM**

**Better Handling of Long Sequences**: LSTM reduces repetition and preserves sentence structure better than SimpleRNN, especially in short to medium-length sentences.

**Reduced Repetition**: LSTM successfully minimizes the excessive repetition seen in SimpleRNN outputs.

**Improved Context in Simple Sentences**: LSTM maintains context more effectively in shorter, less complex sentences, offering more coherent translations.

**Remaining Challenges**

**Long-Term Dependencies**: LSTM still struggles with very long, complex sentences, occasionally producing repetitive or incoherent outputs.

**Out-of-Vocabulary Tokens (OOV)**: Both SimpleRNN and LSTM have difficulty handling unfamiliar words, leading to <OOV> tokens.

**Repetitive Outputs in Certain Cases**: While improved, LSTM sometimes repeats words unnecessarily in complex structures.