


```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from nltk.translate.bleu_score import sentence_bleu
import random
import tensorflow as tf
```


```
from google.colab import drive
drive.mount('/content/drive')
```



 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

```
# Import the pandas library and assign it the alias 'pd'
import pandas as pd
```

```
# 1. Load the dataset
df = pd.read_excel('/content/drive/MyDrive/AI_Assignment/tamil_sentences.xlsx')
```

```
# Display the first few rows
df.head()
```




	Ungrammatical Statement	Standard Tamil	
0	அவளே குரலுடன் பாடினான்.	அவன் குரலுடன் பாடினான்.	
1	அவன் விளையாட்டில் சிறந்தான்.	அவன் விளையாட்டில் சிறந்தான்.	
2	அவர்கள் கதையை கேட்டான்.	அவர்கள் கதையை கேட்டார்கள்.	
3	அவள் மொபைலை பிடித்தான்.	அவள் மொபைலை பிடித்தாள்.	
4	அவளை பார்த்தான் அவன் நண்பர்கள்.	அவளை அவன் நண்பர்கள் பார்த்தார்கள்.	

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
# Clean dataset to handle missing or unexpected values
df = df.dropna(subset=['Ungrammatical Statement', 'Standard Tamil'])
df['Ungrammatical Statement'] = df['Ungrammatical Statement'].astype(str)
df['Standard Tamil'] = df['Standard Tamil'].astype(str)
```

```
# Separate the ungrammatical statements and standard Tamil sentences
input_sentences = df['Ungrammatical Statement'].values
target_sentences = ['<start> ' + sentence + ' <end>'] for sentence in df['Standard Tamil'].values]
```

 `<ipython-input-143-fbde7730036d>:3: SettingWithCopyWarning:`  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-c](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c)  
df['Ungrammatical Statement'] = df['Ungrammatical Statement'].astype(str)

`<ipython-input-143-fbde7730036d>:4: SettingWithCopyWarning:`  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-c](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c)  
df['Standard Tamil'] = df['Standard Tamil'].astype(str)

```
# Tokenize the input and target sentences
input_tokenizer = Tokenizer(filters='')
input_tokenizer.fit_on_texts(input_sentences)
input_sequences = input_tokenizer.texts_to_sequences(input_sentences)

output_tokenizer = Tokenizer(filters='')
output_tokenizer.fit_on_texts(target_sentences)
target_sequences = output_tokenizer.texts_to_sequences(target_sentences)
```

```

# Find the maximum sequence lengths
max_input_length = max(len(seq) for seq in input_sequences)
max_target_length = max(len(seq) for seq in target_sequences)

# Pad sequences to ensure equal length
encoder_input_data = pad_sequences(input_sequences, maxlen=max_input_length, padding='post')
decoder_input_data = pad_sequences(target_sequences, maxlen=max_target_length, padding='post')

# Prepare decoder output data (shifted by 1 position)
decoder_output_data = np.zeros((len(target_sequences), max_target_length, len(output_tokenizer.word_index) + 1), dtype='float32')
for i, seq in enumerate(target_sequences):
    for t, word_id in enumerate(seq):
        if t > 0: # Skip the first token (start token)
            decoder_output_data[i, t - 1, word_id] = 1.0

# Train-Test Split
encoder_input_train, encoder_input_val, decoder_input_train, decoder_input_val, decoder_output_train, decoder_output_val = train_test_split(
    encoder_input_data, decoder_input_data, decoder_output_data, test_size=0.2, random_state=42)

# Build the Seq2Seq Model
embedding_dim = 256
hidden_units = 512

# Encoder
encoder_inputs = Input(shape=(max_input_length,))
encoder_embedding = Embedding(input_dim=len(input_tokenizer.word_index) + 1, output_dim=embedding_dim)(encoder_inputs)
encoder_lstm = LSTM(hidden_units, return_state=True)
_, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(max_target_length,))
decoder_embedding = Embedding(input_dim=len(output_tokenizer.word_index) + 1, output_dim=embedding_dim)(decoder_inputs)
decoder_lstm = LSTM(hidden_units, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
decoder_dense = Dense(len(output_tokenizer.word_index) + 1, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the Model
batch_size = 64
epochs = 50
history = model.fit(
    [encoder_input_train, decoder_input_train],
    decoder_output_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=([encoder_input_val, decoder_input_val], decoder_output_val)
)

```



```

Epoch 34/50
5/5 ----- 3s 388ms/step - accuracy: 0.5970 - loss: 0.2343 - val_accuracy: 0.5262 - val_loss: 0.5133
Epoch 35/50
5/5 ----- 2s 490ms/step - accuracy: 0.5961 - loss: 0.2220 - val_accuracy: 0.5316 - val_loss: 0.4962
Epoch 36/50
5/5 ----- 2s 495ms/step - accuracy: 0.5995 - loss: 0.2069 - val_accuracy: 0.5371 - val_loss: 0.4896
Epoch 37/50
5/5 ----- 2s 327ms/step - accuracy: 0.5929 - loss: 0.1940 - val_accuracy: 0.5389 - val_loss: 0.4939
Epoch 38/50
5/5 ----- 2s 319ms/step - accuracy: 0.6027 - loss: 0.2051 - val_accuracy: 0.5389 - val_loss: 0.4760
Epoch 39/50
5/5 ----- 3s 314ms/step - accuracy: 0.6075 - loss: 0.1833 - val_accuracy: 0.5371 - val_loss: 0.4715
Epoch 40/50
5/5 ----- 3s 322ms/step - accuracy: 0.6008 - loss: 0.1708 - val_accuracy: 0.5407 - val_loss: 0.4625
Epoch 41/50
5/5 ----- 3s 511ms/step - accuracy: 0.6068 - loss: 0.1781 - val_accuracy: 0.5389 - val_loss: 0.4639
Epoch 42/50
5/5 ----- 3s 479ms/step - accuracy: 0.6031 - loss: 0.1692 - val_accuracy: 0.5425 - val_loss: 0.4515
Epoch 43/50
5/5 ----- 2s 405ms/step - accuracy: 0.6166 - loss: 0.1612 - val_accuracy: 0.5407 - val_loss: 0.4535
Epoch 44/50
5/5 ----- 2s 310ms/step - accuracy: 0.6136 - loss: 0.1500 - val_accuracy: 0.5443 - val_loss: 0.4437
Epoch 45/50
5/5 ----- 2s 311ms/step - accuracy: 0.6117 - loss: 0.1463 - val_accuracy: 0.5479 - val_loss: 0.4336
Epoch 46/50
5/5 ----- 2s 315ms/step - accuracy: 0.6148 - loss: 0.1498 - val_accuracy: 0.5479 - val_loss: 0.4272
Epoch 47/50
5/5 ----- 2s 308ms/step - accuracy: 0.6123 - loss: 0.1370 - val_accuracy: 0.5497 - val_loss: 0.4234
Epoch 48/50
5/5 ----- 3s 450ms/step - accuracy: 0.6201 - loss: 0.1286 - val_accuracy: 0.5479 - val_loss: 0.4252
Epoch 49/50
5/5 ----- 3s 511ms/step - accuracy: 0.6146 - loss: 0.1282 - val_accuracy: 0.5479 - val_loss: 0.4299
Epoch 50/50
5/5 ----- 2s 329ms/step - accuracy: 0.6136 - loss: 0.1304 - val_accuracy: 0.5479 - val_loss: 0.4211

```

# Inference

```

encoder_model = Model(encoder_inputs, encoder_states)
decoder_state_input_h = Input(shape=(hidden_units,))
decoder_state_input_c = Input(shape=(hidden_units,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_embedding_inf = Embedding(input_dim=len(output_tokenizer.word_index) + 1, output_dim=embedding_dim)(decoder_inputs)
decoder_outputs, state_h, state_c = decoder_lstm(decoder_embedding_inf, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)

```

```

def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1))
    target_seq[0, 0] = output_tokenizer.word_index['<start>']
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_word = None
        for word, index in output_tokenizer.word_index.items():
            if index == sampled_token_index:
                sampled_word = word
                break
        if sampled_word == '<end>' or len(decoded_sentence.split()) > max_target_length:
            stop_condition = True
        else:
            decoded_sentence += ' ' + sampled_word
            target_seq = np.zeros((1, 1))
            target_seq[0, 0] = sampled_token_index
            states_value = [h, c]
    return decoded_sentence.strip()

```

# Test the model with random sentences

```

random_indices = random.sample(range(len(input_sentences)), 5)
selected_test_inputs = [input_sentences[i] for i in random_indices]
selected_test_targets = [target_sentences[i] for i in random_indices]
predictions = []
references = []
for input_sentence, target_sentence in zip(selected_test_inputs, selected_test_targets):
    input_seq = pad_sequences(input_tokenizer.texts_to_sequences([input_sentence]), maxlen=max_input_length, padding='post')
    predicted_sentence = decode_sequence(input_seq)

```

```

predictions.append(predicted_sentence.split())
references.append([target_sentence.split()])

```

```

1/1 _____ 0s 163ms/step
1/1 _____ 0s 176ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 26ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 22ms/step
1/1 _____ 0s 26ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 22ms/step
1/1 _____ 0s 25ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 35ms/step
1/1 _____ 0s 28ms/step
1/1 _____ 0s 29ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 22ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 22ms/step

```

```

# Test the model with random sentences
random_indices = random.sample(range(len(input_sentences)), 5)
selected_test_inputs = [input_sentences[i] for i in random_indices]
selected_test_targets = [target_sentences[i] for i in random_indices]
predictions = []
references = []
for input_sentence, target_sentence in zip(selected_test_inputs, selected_test_targets):
    input_seq = pad_sequences(input_tokenizer.texts_to_sequences([input_sentence]), maxlen=max_input_length, padding='post')
    predicted_sentence = decode_sequence(input_seq)
    predictions.append(predicted_sentence.split())
    references.append([target_sentence.split()])

# Calculate BLEU Scores for the selected sentences
bleu_scores = [sentence_bleu(ref, pred, weights=(0.5, 0.5)) for pred, ref in zip(predictions, references)]

# Compute the average BLEU score
avg_bleu_score = sum(bleu_scores) / len(bleu_scores)

# Display BLEU scores
for i, score in enumerate(bleu_scores):
    print(f"Sentence {i+1} BLEU Score: {score:.4f}")
print(f"Average BLEU Score for 5 sentences: {avg_bleu_score:.4f}")

```

```

1/1 _____ 0s 113ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 25ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 26ms/step
1/1 _____ 0s 28ms/step
1/1 _____ 0s 33ms/step
1/1 _____ 0s 25ms/step
1/1 _____ 0s 22ms/step
1/1 _____ 0s 22ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 23ms/step
1/1 _____ 0s 24ms/step
1/1 _____ 0s 32ms/step
Sentence 1 BLEU Score: 0.0000
Sentence 2 BLEU Score: 0.0000
Sentence 3 BLEU Score: 0.2231
Sentence 4 BLEU Score: 0.5134
Sentence 5 BLEU Score: 0.3679
Average BLEU Score for 5 sentences: 0.2209
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of

```

```

how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
warnings.warn(_msg)

```

```

# Define the list of ungrammatical sentences
input_sentences = [
    "அவள் வேலை செய்தான்.",
    "நான் நேற்று சாப்பிடுவேன்.",
    "நான் இன்று பாடம் படிக்கிறேன்.",
    "அவர்கள் கதையை கேட்டான்.",
    "நான் நேற்று பாடம் படிக்கிறேன்.",
]

# Loop through each sentence, preprocess, and get the predicted grammatical sentence
for idx, input_sentence in enumerate(input_sentences):
    # Preprocess the input sentence
    input_seq = pad_sequences(input_tokenizer.texts_to_sequences([input_sentence]), maxlen=max_input_length, padding='post')

    # Get the predicted sentence using the model
    predicted_sentence = decode_sequence(input_seq)

    # Output the result for each sentence
    print(f"\n--- Sentence {idx+1} ---")
    print("Input Sentence (Ungrammatical):", input_sentence)
    print("Predicted Sentence (Grammatical):", predicted_sentence)

```

```

1/1 _____ 0s 132ms/step
1/1 _____ 0s 74ms/step
1/1 _____ 0s 50ms/step
1/1 _____ 0s 47ms/step
1/1 _____ 0s 48ms/step

```

```

--- Sentence 1 ---
Input Sentence (Ungrammatical): அவள் வேலை செய்தான்.
Predicted Sentence (Grammatical): அவள் வேலை செய்தாள்.
1/1 _____ 0s 50ms/step
1/1 _____ 0s 38ms/step
1/1 _____ 0s 41ms/step
1/1 _____ 0s 61ms/step
1/1 _____ 0s 44ms/step

```

```

--- Sentence 2 ---
Input Sentence (Ungrammatical): நான் நேற்று சாப்பிடுவேன்.
Predicted Sentence (Grammatical): நான் நேற்று சாப்பிட்டேன்.
1/1 _____ 0s 43ms/step
1/1 _____ 0s 43ms/step
1/1 _____ 0s 44ms/step
1/1 _____ 0s 42ms/step
1/1 _____ 0s 48ms/step

```

```

--- Sentence 3 ---
Input Sentence (Ungrammatical): நான் இன்று பாடம் படிக்கிறேன்.
Predicted Sentence (Grammatical): நான் இன்று பாடம்
1/1 _____ 0s 47ms/step
1/1 _____ 0s 50ms/step
1/1 _____ 0s 38ms/step
1/1 _____ 0s 39ms/step

```

```

--- Sentence 4 ---
Input Sentence (Ungrammatical): அவர்கள் கதையை கேட்டான்.
Predicted Sentence (Grammatical): அவர்கள் கேட்டார்கள்.
1/1 _____ 0s 40ms/step
1/1 _____ 0s 50ms/step
1/1 _____ 0s 42ms/step
1/1 _____ 0s 57ms/step
1/1 _____ 0s 54ms/step
1/1 _____ 0s 49ms/step

```

```

--- Sentence 5 ---
Input Sentence (Ungrammatical): நான் நேற்று பாடம் படிக்கிறேன்.
Predicted Sentence (Grammatical): நான் நேற்று பாடம் படித்தேன்.

```

