Language Translation with Sequence Models

A PROJECT REPORT

for

AI Project(AI101B)

Session (2024-25)

Submitted by

Manish Kumar Singh

(202410116100114)

Divyanshu Mishra

(202410116100068)

Divyansh Pathak

(202410116100067)

Submitted in partial fulfilment of the Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of

Mr. Apoorv Jain

Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS

KIET Group of Institutions, Ghaziabad

Uttar Pradesh-201206

TABLE OF CONTENTS

Introduction

Language translation is one of the most important applications of natural language processing (NLP), allowing us to convert text or speech from one language to another. This has become increasingly useful in today's global world, where people speak and write in hundreds of different languages. From translating websites to helping people communicate across cultures, automatic language translation plays a key role in bridging the language gap.

Traditional translation methods relied heavily on rules and dictionaries or statistical approaches that matched word patterns. However, these techniques often produced inaccurate or awkward results because they could not fully understand the structure or meaning of sentences. To overcome these limitations, deep learning models—especially Sequence-to-Sequence (Seq2Seq) models—have become popular for building more accurate and fluent translation systems.

A Sequence-to-Sequence model is a type of neural network that processes input as a sequence (like a sentence) and generates an output sequence (like the translated sentence). It has two main parts: an encoder that reads and understands the input, and a decoder that generates the translated output. These models are especially powerful because they can handle sentences of different lengths and capture the context and meaning of each word, rather than translating word by word.

In recent years, more advanced versions of sequence models, like Transformers, have further improved translation quality by allowing models to look at all parts of a sentence at once. These models, such as MarianMT, BERT, and GPT, have set new standards for machine translation. Whether it's through apps, websites, or chatbots, language translation using sequence models continues to grow, helping people connect and understand each other across language barriers.

As the demand for multilingual communication grows, the use of sequence models in language translation is expanding into various real-world applications. From customer support systems and educational tools to voice assistants and travel apps, these models are enhancing user experiences across industries. With continued research and access to large multilingual datasets, translation models are becoming more accurate, faster, and even capable of handling low-resource languages. As a result, they are not just translating words—they are helping build a more connected and inclusive world.

Methodology

1. Approach

The task of language translation, specifically from English to Hindi, is approached using Supervised Deep Learning techniques. In this method, models are trained on paired sentences—where each English sentence has a corresponding Hindi translation. The model learns the underlying relationships between the sequences and generates accurate translations for new input. The overall approach involves the following steps:

1. Data Acquisition:
   The dataset used for training is a parallel corpus of English-Hindi sentence pairs. Sources may include publicly available datasets like the Tatoeba Project, OpenSubtitles, or custom datasets in CSV format (e.g., Dataset_English_Hindi.csv).

2. Data Preprocessing:
   Text preprocessing is essential for preparing clean input data. This involves removing missing values, converting text to lowercase, adding special tokens (e.g., <start> and <end> for target sentences), tokenizing the text using subword or word-level tokenizers, and padding sequences to ensure uniform input dimensions.

3. Exploratory Data Analysis (EDA):
   Basic EDA helps understand the structure and vocabulary of the dataset. This includes analyzing sentence lengths, unique token counts, and sample sentence pairs to assess translation complexity.

4. Model Selection and Training:
   The Sequence-to-Sequence (Seq2Seq) architecture is employed for this task. The model comprises:

   - An Encoder that processes the input English sentence.
   - A Decoder that generates the output Hindi sentence.
   - Both are typically built using LSTM (Long Short-Term Memory) layers or GRU (Gated Recurrent Unit) layers.
     Alternatively, pretrained Transformer-based models like MarianMT can be used for faster implementation and better accuracy.

5. Hyperparameter Tuning:
   Key parameters like embedding dimension, LSTM units, learning rate, batch size, and maximum sequence length are tuned. Techniques like Grid Search or manual tuning based on validation performance are used.

6. Model Evaluation:
   The model is evaluated on a validation/test dataset using metrics such as:
   - BLEU Score (Bilingual Evaluation Understudy) – to measure the quality of translation
   - Loss (Categorical Crossentropy) – during training and validation
   - Inference Accuracy – comparing predicted translations with actual Hindi sentences

This structured approach ensures that the trained model can translate unseen English text to Hindi with a high degree of fluency and grammatical correctness.

2. Algorithm Used

To accomplish the translation task, the following algorithms and architectures were used and analyzed:

1. Sequence-to-Sequence (Seq2Seq) using LSTM:
   This model includes an encoder-decoder structure. The encoder compresses the English sentence into a fixed context vector, which the decoder then uses to generate the Hindi translation. LSTM layers are preferred due to their ability to handle long dependencies.

2. Attention Mechanism (optional but effective):
   The attention layer allows the decoder to focus on relevant words from the input sequence while generating each word of the output. This improves translation accuracy, especially for longer sentences.

3. Transformer Models (Pretrained - MarianMT):
   For faster and more accurate results, pretrained transformer models like Helsinki-NLP/opus-mt-en-hi are used. These models use multi-head self-attention mechanisms and position-wise feed-forward layers to learn relationships across sequences without relying on recurrence.

4. Tokenization (Subword or Word-Level):
   Tokenizers such as MarianTokenizer or Tokenizer from Keras are used to convert text into sequences of tokens or subwords. Limiting the vocabulary size helps reduce memory usage and training time.

5. Embedding Layer:
   A dense vector representation of each token is learned during training, which helps the model understand semantic similarities between words.

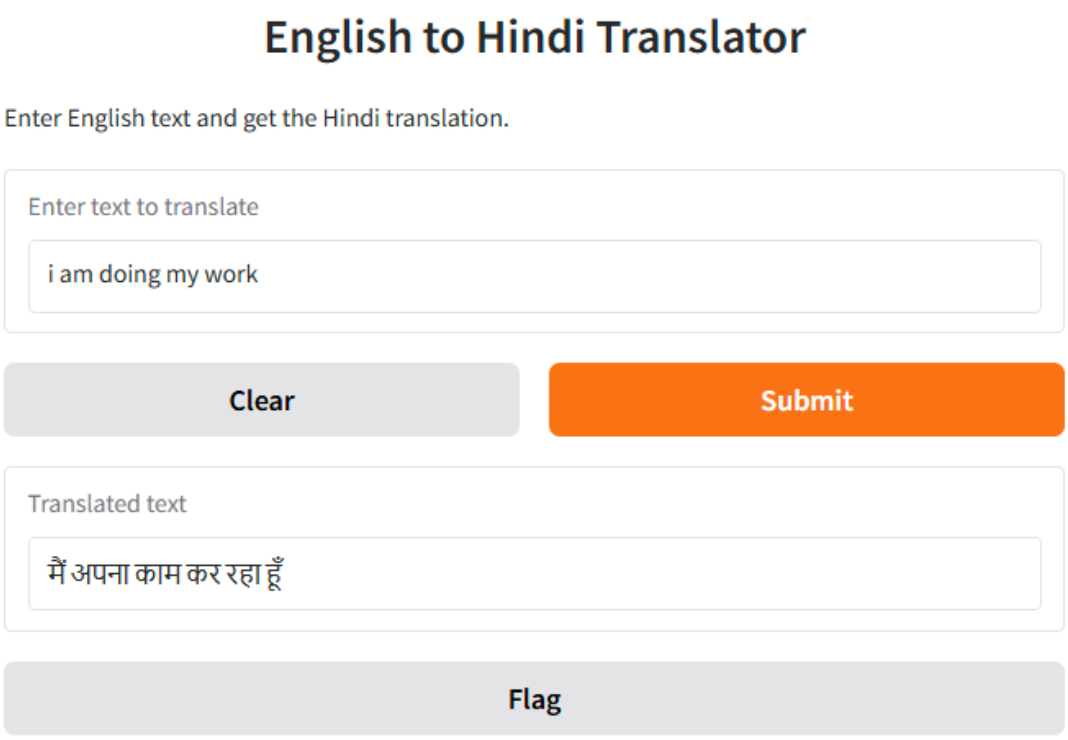
6. Beam Search Decoding (optional):
   Instead of greedy decoding, beam search helps improve translation quality by considering multiple likely output sequences and choosing the best one.

Each of these components plays a critical role in producing high-quality translations. The final model is selected based on performance in terms of BLEU score, generalization ability, and fluency of translated output.

Code

```
# Install required packages
# pip install gradio transformers torch
!pip install gradio
import torch
import torch.nn as nn
import torch.optim as optim
from transformers import MarianMTModel, MarianTokenizer
import gradio as gr

# Load model and tokenizer
model_name = 'Helsinki-NLP/opus-mt-en-hi'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Define the translation function
def translate(text):
    tokenized_text = tokenizer.prepare_seq2seq_batch([text], return_tensors='pt')
    translation = model.generate(**tokenized_text)
    translated_text = tokenizer.decode(translation[0], skip_special_tokens=True)
    return translated_text

# Create the Gradio interface
iface = gr.Interface(
    fn=translate,
    inputs=gr.Textbox(label="Enter text to translate"),
    outputs=gr.Textbox(label="Translated text"),
    title="English to Hindi Translator",
    description="Enter English text and get the Hindi translation."
)

# Launch the interface
iface.launch()
```

Outputs


English to Hindi Translator
Enter English text and get the Hindi translation.
Enter text to translate
i am doing my work
Clear
Submit
Translated text
मैं अपना काम कर रहा हूँ
Flag

OUTPUT EXPLANATION

1. Library Installation

- The first step ensures that all necessary Python packages are installed in the environment.
- The command installs:
    - transformers: This library, developed by Hugging Face, provides access to powerful pretrained language models like MarianMT.
    - gradio: A Python library that allows you to create simple web interfaces for machine learning models without needing frontend coding skills.
    - sentencepiece: Required by some transformer tokenizers for processing subword units, which help handle unknown or rare words more effectively.

---

2. Importing Required Libraries

- The code imports the following:
    - torch: A deep learning framework used to run models efficiently.
    - MarianMTModel and MarianTokenizer from the transformers library: These are specific classes used for machine translation.
    - gradio: For building and displaying the web interface.
- These imports prepare the environment to load the model and process user input.

---

3. Loading the Pretrained Model and Tokenizer

- The model name Helsinki-NLP/opus-mt-en-hi is specified. It is a pretrained translation model trained on English-Hindi language pairs.
- The tokenizer is responsible for:
    - Splitting the input text into manageable pieces (tokens).
    - Converting these tokens into numerical format (IDs) so the model can process them.
- The model is responsible for:
    - Taking the encoded English tokens as input.
    - Generating token IDs for the translated Hindi sentence.
- This step ensures the model is ready to perform real-time translation without any additional training.

---

4. Defining the Translation Function

- A function named translate() is defined to handle translation from English to Hindi.
- Inside the function:
    - The input English sentence is first checked to ensure it is not empty.
    - The sentence is then tokenized using the prepare_seq2seq_batch() method (which prepares the sentence for translation).
    - The model generates a translation using the .generate() method.
    - The output tokens are converted back into Hindi text using .decode().
- This function encapsulates all the logic for translation and is connected to the Gradio interface.

---

5. Creating the Gradio Interface

- The Gradio Interface() function is used to create a simple and user-friendly web interface.
- It includes:
    - An input textbox where the user can type English sentences.
    - An output textbox to display the translated Hindi sentence.
    - A title and description to explain what the app does.
- This interface makes the model accessible to anyone, even users without programming knowledge.

---

6. Launching the Application

- The launch() function starts the app and opens it in the browser.
- The interface allows users to:
    - Type an English sentence.
    - Press a button to get a translated Hindi output.
    - See results instantly in a clean and simple format.

Conclusion

 Language Translation with Sequence Models:

- Sequence models, particularly in NLP, have revolutionized language translation.
- The project focuses on using such models to create an English-to-Hindi translator.

 Pretrained Transformer Model:

- The MarianMT model, which is Transformer-based, is used in this project.
- This model is pretrained, meaning it's already trained on vast amounts of multilingual data, allowing it to translate accurately.

 No Need for Extensive Dataset Collection:

- By using the pretrained MarianMT model, there's no requirement for collecting and annotating large translation datasets.
- This saves significant time and effort, as the model already knows how to translate based on previous training.

 Gradio Interface Integration:

- The MarianMT model is integrated into a Gradio interface, providing a user-friendly platform.
- Users can input English text and instantly receive Hindi translations in real time.

 Demonstrating Deep Learning and NLP Tools:

- The project highlights how deep learning and modern NLP technologies (like Transformers) can address real-world problems like language translation.
- It demonstrates the potential of combining advanced models with accessible interfaces.

 Real-World Application and Accessibility:

- The project showcases the practical application of sequence-to-sequence learning and transfer learning in real-world use.
- It makes technology more accessible to people from different linguistic backgrounds by enabling smoother communication across languages.

 Opportunities for Further Development:

- The success of this project opens up further opportunities to expand multilingual applications.
- There is room for improving the system to handle more languages or offer enhanced features such as domain-specific translations.

 Impact on Multilingual Communities: