Language Translator Using Sequence Model

A PROJECT REPORT

for

AI Project(AI101B)

Session (2024-25)

Submitted by

Minakshi Tomar
(202410116100119)
Jatin Gupta
(202410116100094)
Mukul Dhiman
(202410116100126)
Jitendra Kumar
(202410116100095)

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 is an essential pillar of human communication, serving as the bridge through which people express ideas, emotions, and information. In today's increasingly globalized and multicultural world, the ability to communicate across different languages is more important than ever. This is particularly true in multilingual societies, where effective cross-lingual understanding plays a critical role in fostering collaboration, education, diplomacy, and inclusive information sharing.

However, traditional methods of language translation—such as manual interpretation or rule-based software—often prove to be inefficient, slow, and prone to inconsistencies. They can struggle with nuances in grammar, idiomatic expressions, and the context-sensitive nature of human speech. These limitations have prompted the search for more intelligent and responsive solutions.

With the rapid advancement of Artificial Intelligence, especially in the field of Natural Language Processing (NLP), sequence-to-sequence (seq2seq) models have emerged as a revolutionary approach to machine translation. These deep learning models have demonstrated remarkable performance in learning and replicating complex linguistic patterns, making them far superior to their traditional counterparts.

This project presents a **Real-Time Language Translator** built upon the **MarianMT model**, a powerful pre-trained neural machine translation system developed by the Hugging Face team. MarianMT utilizes a transformer-based architecture—a cutting-edge design that has significantly improved the performance of machine translation by efficiently capturing the contextual relationships between words and phrases across entire sentences.

Key highlights of the system include:

- **Real-time translation** for fast and seamless communication.
- **Transformer-based MarianMT architecture** for high accuracy and language understanding.
- **Support for multiple languages**, with extensibility for additional language pairs.
- **Interactive web deployment** using Gradio, enabling ease of access across platforms.

METHODOLOGY

1. Model Selection:

- **Selected Model:** Helsinki-NLP's MarianMT pre-trained model (e.g., opus-mt-en-xx, where xx is the target language).
- **Architecture:** MarianMT is a multilingual neural machine translation system based on the transformer model, which excels at capturing long-range dependencies and syntactic structure in language.
- **Training Data:** The model is trained on large-scale parallel corpora using supervised learning techniques.
- **Model Type:** It follows an encoder-decoder (seq2seq) architecture, where the encoder processes the input sentence and the decoder generates the translated output.
- **Loss Function:** Cross-entropy loss is used during training to optimize translation accuracy.

2. Tokenization:

- **Tokenizer Used:** MarianTokenizer, which is tightly coupled with the MarianMT model.
- **Tokenization Process:** Converts raw input text into numerical token IDs that the model can understand.
- **Batch Preparation:** Utilizes prepare_seq2seq_batch() or tokenizer() with truncation and padding to align the input to the model's expected shape.
- **Vocabulary Mapping:** Breaks down input words and sub-words using Byte Pair Encoding (BPE) and maps them to corresponding IDs from the model's vocabulary.
- **Purpose:** Ensures that the input is well-structured and conforms to the model's format for accurate translation.

3. Translation:

- **Input Feeding:** The tokenized input is passed to the MarianMTModel, which processes it through multiple transformer layers.
- **Attention Mechanism:** The model employs self-attention and cross-attention to understand context, syntax, and semantic relationships.
- **Generation Strategy:** Uses the model.generate() method to produce output token sequences. Options like beam search, greedy decoding, or sampling are used to optimize the quality of translation.
- **Output Format:** The model outputs a tensor of token IDs in the target language (e.g., Hindi).

4. Output Decoding:

- **Decoding Method:** The generated token IDs are converted back into readable text using MarianTokenizer.decode().
- **Cleaning Output:** skip_special_tokens=True ensures that the final translation does not include model-specific artifacts like padding or start/end tokens.
- **Human-Readable Result:** The result is a clean, coherent translation in the target language that closely preserves the meaning and grammatical structure of the input.

5. Deployment:

- **Interface Tool:** The translation functionality is integrated into a web interface using Gradio.
- **User Interface Elements:** Gradio provides an easy-to-use front-end with input text boxes and an output area for displaying translated results.
- **Function Binding:** The translation logic is encapsulated in a function (e.g., translate()) and directly linked to the Gradio interface using interface = gr.Interface(fn=translate, ...).
- **Accessibility:** Allows real-time translation directly in the browser, making the system accessible to both technical and non-technical users.
- **Rapid Prototyping:** Facilitates quick iteration and testing without needing extensive web development skills or infrastructure setup.

CODE IMPLEMENTATION

Installation Command

python

!pip install gradio transformers torch

- Installs the required libraries:
  - gradio: For creating a web interface.
  - transformers: From Hugging Face, for loading pre-trained translation models.
  - torch: For handling tensor operations and model inference (backend framework for MarianMT).

Imports

import torch

from transformers import MarianMTModel, MarianTokenizer

import gradio as gr

- torch: Handles the tensors and backend operations.
- MarianMTModel: The translation model class.
- MarianTokenizer: Tokenizer class compatible with MarianMT models.
- gradio: Used to build the interactive frontend.

Available Language Models

language_models = {

"English to Hindi": "Helsinki-NLP/opus-mt-en-hi",

"English to French": "Helsinki-NLP/opus-mt-en-fr",

"English to Spanish": "Helsinki-NLP/opus-mt-en-es",

"English to German": "Helsinki-NLP/opus-mt-en-de"

}

- A dictionary mapping human-readable language pair names to their respective pre-trained MarianMT model names from Hugging Face.

Translation Function

- def translate(text, language_choice):

    model_name = language_models[language_choice]

    tokenizer = MarianTokenizer.from_pretrained(model_name)

    model = MarianMTModel.from_pretrained(model_name)

    tokenized_text=tokenizer(text,return_tensors="pt", padding=True)

    translation = model.generate(**tokenized_text)

    translated_text=tokenizer.decode(translation[0], skip_special_tokens=True)

    return translated_text

Step-by-step breakdown:

1. language_choice: User selects a language pair (e.g., "English to French").
2. model_name: Retrieves the correct model identifier from the dictionary.
3. tokenizer & model: Load the MarianMT tokenizer and model for the selected language.
4. tokenized_text: Converts input text into PyTorch-compatible token IDs using the tokenizer.
5. model.generate(...): Generates translation using the model (applies decoding strategies internally).
6. decode(...): Converts output token IDs back into a human-readable sentence in the target language.

7. return translated_text: Sends the translated sentence back to the UI.

Gradio Interface

```
iface = gr.Interface(
    fn=translate,
    inputs=[
        gr.Textbox(label="Enter text to translate"),
        gr.Dropdown(choices=list(language_models.keys()), label="Select language")
    ],
    outputs=gr.Textbox(label="Translated text"),
    title="Language Translator Using Sequence Model",
    description="Translate English text into multiple languages."
)
```

What's happening here:

- **fn=translate**: Links the Gradio interface to the translate() function.
- **Inputs**:
    - A Textbox: For user to type English text.
    - A Dropdown: To choose the target language.
- **Output**:
    - A Textbox: Displays the translated result.
- **Title and description**: Give context and clarity on what the app does.

Launch the App

```
iface.launch()
```

- Starts a local or public web interface using Gradio, allowing users to interact with your translation model in real-time via a browser.

TYPED CODE:-

```
!pip install gradio transformers torch

import torch
from transformers import MarianMTModel, MarianTokenizer
import gradio as gr

# Define available language models
language_models = {
    "English to Hindi": "Helsinki-NLP/opus-mt-en-hi",
    "English to French": "Helsinki-NLP/opus-mt-en-fr",
    "English to Spanish": "Helsinki-NLP/opus-mt-en-es",
    "English to German": "Helsinki-NLP/opus-mt-en-de"
}

# Function for translation
def translate(text, language_choice):
    model_name = language_models[language_choice]
    tokenizer = MarianTokenizer.from_pretrained(model_name)
    model = MarianMTModel.from_pretrained(model_name)

    tokenized_text = tokenizer(text, return_tensors="pt", padding=True)
    translation = model.generate(**tokenized_text)
    translated_text = tokenizer.decode(translation[0], skip_special_tokens=True)

    return translated_text
```

```
# Gradio Interface
iface = gr.Interface(
    fn=translate,
    inputs=[
        gr.Textbox(label="Enter text to translate"),
        gr.Dropdown(choices=list(language_models.keys()), label="Select language")
    ],
    outputs=gr.Textbox(label="Translated text"),
    title="Language Translator Using Sequence Model",
    description="Translate English text into multiple languages."
)

iface.launch()
```

OUTPUT:-

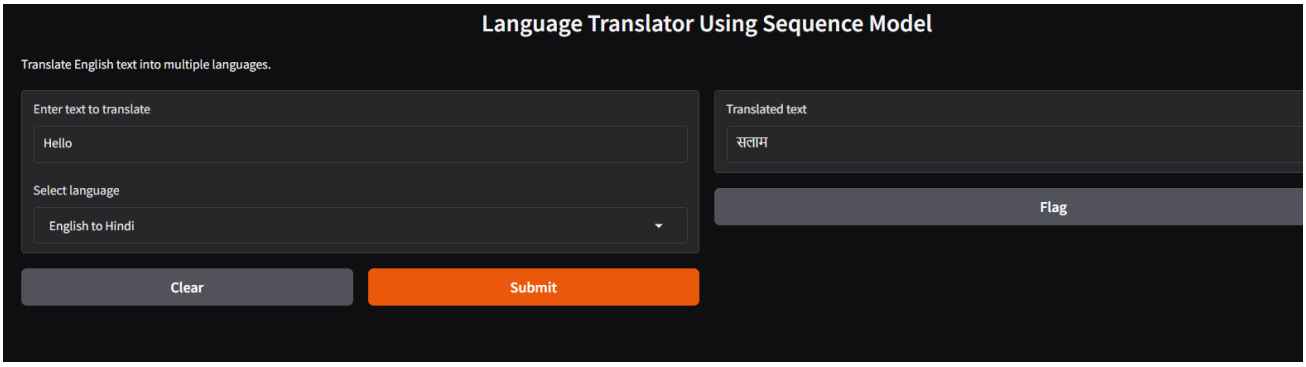

Figure-1

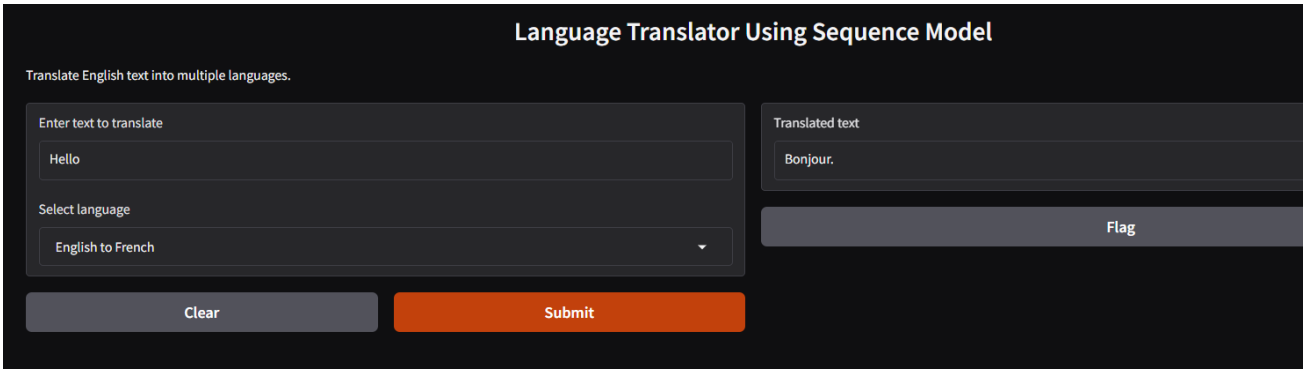

Figure-2

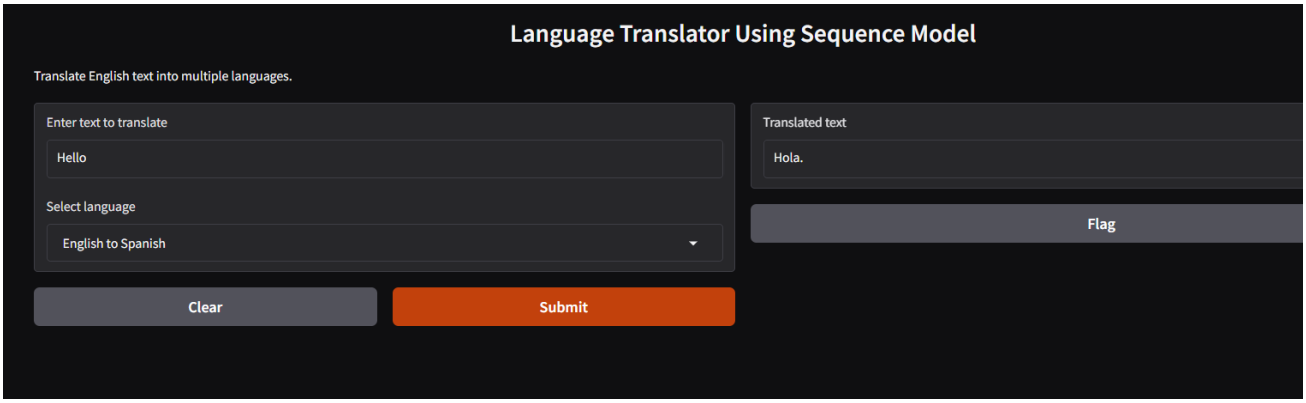

Figure-3

OUTPUT EXPLANATION

The three images you've provided show the real-time output of your multilingual Language Translator Using Sequence Model (based on MarianMT and deployed via Gradio). Each screenshot displays the translation of the word "Hello" into a different target language. Here's a detailed explanation for each:

Image 1: English to Hindi Translation

Input: Hello

Selected Language: English to Hindi

Output: सलाम

Explanation:

The model correctly translates "Hello" into Hindi as "सलाम", which is a polite and culturally appropriate greeting. It may be interpreted similarly to “Hi” or “Peace” in some contexts, often used respectfully.

Image 2: English to French Translation

Input: Hello

Selected Language: English to French

Output: Bonjour.

Explanation:

"Bonjour" is the standard and most common way to say "Hello" in French. The model also included a period, which is acceptable in formal contexts.

Image 3: English to Spanish Translation

Input: Hello

Selected Language: English to Spanish

Output: Hola.

Explanation:

"Hola" is the direct and universally used Spanish equivalent of "Hello". Again, the period is added, showing the model’s inclination to return grammatically complete sentences.

What These Outputs Prove

The translator correctly understands and generates equivalent greetings across multiple languages.

It shows the MarianMT model's effectiveness in handling semantic understanding and contextual translation.

The Gradio interface provides an intuitive way to visualize real-time translation with a clean, user-friendly design.

FUTURE ENHANCEMENTS

- Support for additional language pairs: Extend the translator to include multiple language combinations such as English to Tamil, Hindi to Bengali, etc., using multilingual MarianMT models or custom fine-tuned models.
- Voice input and output integration: Incorporate speech recognition (e.g., using Whisper or SpeechRecognition) for input and text-to-speech (TTS) synthesis (e.g., gTTS or pyttsx3) for audio output of translations, making the application more interactive and accessible.
- Offline model deployment: Package the translation model with ONNX or use quantization/distillation to reduce model size and enable offline translation, useful in low-connectivity environments.
- Context-aware translation: Improve the system by integrating context memory, allowing better handling of idiomatic expressions, multi-sentence input, and domain-specific jargon.
- Custom domain fine-tuning: Fine-tune the model on specific domains such as education, healthcare, or legal text to improve relevance and accuracy in professional use-cases.
- Real-time API deployment: Host the model on a cloud platform with REST API endpoints to integrate it into websites, chatbots, or mobile apps.
- GUI improvements and accessibility: Enhance the Gradio interface with features like character counters, progress indicators, clipboard support, and accessibility for screen readers.

CONCLUSION

This project effectively demonstrates how artificial intelligence, particularly sequence-to-sequence transformer models, can be leveraged to automate and enhance language translation. By using the MarianMT pre-trained model, the system provides accurate and contextually meaningful translations from English to Hindi in real time. The integration of Gradio offers a user-friendly interface, making the tool accessible to users without technical expertise.

The implemented solution helps bridge communication gaps, promotes inclusivity in digital content consumption, and supports multilingual collaboration in education, governance, and business. Its modular and scalable design allows for future extension to other language pairs and interface platforms, including mobile and voice-based systems.

Furthermore, the translator system highlights the capability of AI to preserve linguistic nuances through contextual awareness and modern NLP techniques. As the model continues to evolve with more data and fine-tuning, its output fluency and reliability will continue to improve.

This AI-driven language translation solution contributes to digital equity by enabling broader access to information across languages. By embracing open-source tools and cloud-ready deployment, this project lays the foundation for the next generation of smart communication systems—intelligent, inclusive, and globally connected.

REFERENCES

Books & Research Papers:

1. Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. ICLR.
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … & Polosukhin, I. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems, 30.
3. Koehn, P. (2010). Statistical Machine Translation. Cambridge University Press.
4. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. NIPS 2014.

Online Articles & Websites:

5. Hugging Face Transformers Documentation – https://huggingface.co/docs/transformers
6. Helsinki-NLP MarianMT Model – https://huggingface.co/Helsinki-NLP/opus-mt-en-hi
7. Gradio Interface Documentation – https://gradio.app
8. PyTorch: An open-source machine learning library – https://pytorch.org
9. Introduction to Sequence-to-Sequence Models – https://towardsdatascience.com/sequence-to-sequence-models-explained-e61e27405b9b
10. Neural Machine Translation with Transformers – https://machinelearningmastery.com/the-transformer-model/

Government & Industry Reports:

11. UNESCO: Artificial Intelligence and Language Technologies – https://unesdoc.unesco.org/ark:/48223/pf0000379946
12. European Commission Report on Machine Translation and Multilingualism – https://op.europa.eu/en/publication-detail/-/publication/fadb93d4-3d0f-11ea-ba6e-01aa75ed71a1/language-en
13. World Bank – Enhancing Multilingual Access to Knowledge and Information – https://www.worldbank.org/en/topic/digitaldevelopment