Machine Translation

English to Telugu

Chaitanya Yadavally
Cwid : 50051123

Contents:

Problem Statement: Machine translation is a critical task in natural language processing, facilitating communication across linguistic boundaries. The project at hand aims to bridge the language gap between English and Telugu by leveraging a pre-trained English to Hindi translation model and fine-tuning it for English to Telugu translation. The goal is to develop an effective translation system that can accurately and contextually convert English sentences into their Telugu equivalents.

Abstract:

This project aims to adapt a pre-trained English-Hindi machine translation model to translate between English and Telugu. Leveraging neural networks and transfer learning techniques,I fine-tuned the Helsinki-NLP's MarianMTModel, a part of the Transformers library, for the specific language pair. The successful adaptation of this model demonstrates a novel approach to addressing the scarcity of resources in less-commonly spoken language pairs, like English-Telugu.

1. Introduction

Objective:

The primary objective of this project is to develop a machine translation model capable of accurately translating sentences from English to Telugu, a language pair with limited direct translation resources.

Background:

Machine translation has been a key area of research in the field of natural language processing (NLP). While significant progress has been made in commonly spoken languages, less prevalent language pairs often suffer from inadequate resources. The project addresses this gap by adapting an existing model to a new language pair, thus contributing to the broader goal of making language translation technology more inclusive.

Methodology:

Data Processing of Bilingual Dataset

In my project, I focused on a bilingual dataset containing pairs of sentences in English and Telugu. This dataset is crucial for understanding the differences between English and Telugu in terms of syntax and semantics.

```
# Specify input file path
input_file_path = 'english_telugu_data.txt'

# Specify output file path
output_file_path = 'teluguenglishseparate11.csv'

# Read data from the input file
with open(input_file_path, 'r', encoding='utf-8') as infile:
    telugudata = [line.strip().split('++++$++++') for line in infile]

# Write data to CSV with separate columns for Telugu and English
with open(output_file_path, 'w', newline='', encoding='utf-8') as csvfile:
    csv_writer = csv.writer(csvfile)

    # Write header
    csv_writer.writerow(['English Text', 'Telugu Text'])

    # Write data rows
    csv_writer.writerows(telugudata)


print(f'CSV file "{output_file_path}" created successfully.')
```

Data Format and Preparation: The original data was stored in a text file (english_telugu_data.txt), with each line containing an English sentence and its Telugu counterpart, separated by the delimiter ++++$++++. I wrote a Python script to process this data, splitting each line at the delimiter to separate the English and Telugu sentences.

Conversion to CSV: To make the data more accessible and easier to handle, I converted it into a CSV format. Using Python's csv module, I created a file named teluguenglishseparate11.csv with two columns: 'English Text' and 'Telugu Text'. This structured format is not only more manageable but also ready for any further analysis or application in machine learning models.

This process of data transformation is a key step in my project, ensuring that the dataset is in a format that is suitable for the analysis and modeling work that follows.

Importing required packages:
In the implementation phase of my project, I utilized several Python libraries to handle data processing, model training, and evaluation. Firstly, the csv library was employed for reading and writing CSV files, which was essential for managing our bilingual dataset. For deep learning tasks, I used torch, the core library of **PyTorch**, known for its flexibility and efficiency in building and training neural network models.

Data handling and manipulation were streamlined using pandas, a powerful data analysis and manipulation tool, ideal for working with structured data. To efficiently process our dataset, I utilized **tqdm**, a library that provides a progress bar, enhancing the visibility of long-running operations. The **torch.utils.data** module, specifically **DataLoader** and Dataset, was instrumental in providing an easy-to-use interface for dataset iteration and batching, crucial for training neural network models.

For the model's foundation, I chose BERT (Bidirectional Encoder Representations from Transformers), leveraging the transformers library by Hugging Face. This included **BertTokenizer** for tokenizing the text data and **BertForSequenceClassification** for the sequence classification task. Lastly, I

used **AdamW** from the same library, an optimizer specifically tuned for training deep learning models. The **sklearn.model_selection** module, particularly **train_test_split**, was utilized to divide our dataset into training and testing sets, ensuring a robust evaluation of the model's performance.

Dataset:

```
# Load the dataset
dataset = pd.read_csv('teluguenglishseparate11.csv')

# Take the first 80000 records
dataset = dataset.head(80000)
```

Initially, the dataset, which consists of 150,000 samples, was accessed and loaded into the Python environment using the Pandas library, a standard tool for data manipulation and analysis. To begin with, the focus was on a subset of this dataset. Specifically, the first 80,000 records were extracted for use. This initial selection represents less than half of the entire dataset and was likely chosen to balance between having a sizable amount of data for meaningful analysis and training, and the constraints of computational resources.

Model:

```
from transformers import MarianMTModel, MarianTokenizer
import sentencepiece as spm

model_name = "Helsinki-NLP/opus-mt-en-hi"
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name)
```

For my project, I've taken the "Helsinki-NLP/opus-mt-en-hi" model, which is originally trained for translating from English to Hindi. My aim is to adapt it

for translating English to Telugu. This involves retraining the model with a dataset specifically for English and Telugu, essentially teaching it a new language pair. It's like taking a skilled translator who knows Hindi and teaching them Telugu instead. The strength of this approach lies in using the advanced capabilities of the existing model and molding it to understand and translate Telugu.

```
# Set up the model for fine-tuning
device = torch.device("cuda")
model.to(device)
optimizer = AdamW(model.parameters(), lr=5e-5)
```

I've fine-tuned the model's learning process by selecting the "AdamW" optimizer, known for handling complex neural networks effectively. I've carefully set the learning rate to 5e-5 to strike the right balance between learning speed and accuracy, similar to finding the ideal speed for a vehicle to navigate efficiently.

Data Preparation:

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_length=128):
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        english_text = str(self.data.iloc[idx]['English Text'])
        telugu_text = str(self.data.iloc[idx]['Telugu Text'])

        # Tokenize and encode the inputs
        inputs = self.tokenizer(
            english_text,
            max_length=self.max_length,
            padding="max_length",
            truncation=True,
            return_tensors="pt",
            return_attention_mask=True
        )
```

```
        # Tokenize and encode the labels
        labels = self.tokenizer(
            telugu_text,
            max_length=self.max_length,
            padding="max_length",
            truncation=True,
            return_tensors="pt",
            return_attention_mask=True
        )

        # Return the tokenized inputs and labels
        return {
            "input_ids": inputs["input_ids"].squeeze(),
            "labels": labels["input_ids"].squeeze()  # Treat Telugu text as labels
        }


# Split the dataset into training and validation sets
train_data, temp_data = train_test_split(dataset, test_size=0.2, random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42)

# Use your CustomDataset class
train_dataset = CustomDataset(train_data, tokenizer)
val_dataset = CustomDataset(val_data, tokenizer)
test_dataset = CustomDataset(test_data, tokenizer)

# Create PyTorch DataLoaders
train_dataloader = DataLoader(train_dataset, batch_size=4, shuffle=True)
```

I've taken significant steps to prepare and organize the data for training my machine translation model effectively.

Tokenization and Encoding: Inside the CustomDataset class, I tokenize and encode both the English and Telugu texts. Tokenization is like breaking down sentences into smaller, meaningful units that the model can work with. I've used the specified max_length to ensure that the sequences are of a consistent length. Additionally, I've taken care of padding and truncation to handle sentences of varying lengths.

Data Splitting: I've divided the dataset into three subsets: training, validation, and testing. The train_test_split function was used to achieve this split. The training set is used to train the model, the validation set helps us fine-tune and monitor the model's performance during training, and the test set is reserved for evaluating the model's final performance.

Creating PyTorch DataLoaders: To efficiently feed the data into the model during training, I've created PyTorch DataLoaders for each subset (training, validation, and testing). DataLoaders handle tasks like batching and shuffling, which are crucial for efficient model training.

Batching and Shuffling: I've set the batch size to 4, which means that the model will process four sequences at a time during training. I've also enabled shuffling for the training DataLoader to ensure that the model doesn't learn from the data in a fixed order.

By implementing these steps, I have laid a strong foundation for training my machine translation model. The custom dataset class and PyTorch DataLoaders streamline the data preparation process, making it easier to feed the data into the model for training and evaluation. This careful organization of data is a crucial aspect of the project, setting the stage for effective model training and performance assessment.

Training Over Multiple Epochs: I have decided to train my model for a total of 10 "epochs." An epoch is like a full cycle through our training data. During each epoch, my model learns from the entire training dataset, making gradual improvements.

```python
from tqdm import tqdm

# Initialize the list to store the average loss per epoch
train_losses = []

# Set the number of epochs
num_epochs = 10

# Training loop
for epoch in range(num_epochs):
    model.train()  # Set the model to training mode
    total_loss = 0  # Initialize total loss for this epoch

    # Iterate over each batch in the dataloader
    for batch in tqdm(train_dataloader, desc=f"Epoch {epoch + 1}/{num_epochs}", unit="batch"):
        optimizer.zero_grad()  # Reset gradients to zero
        inputs = batch["input_ids"].to(device)  # Load input to device
        labels = batch["labels"].to(device)  # Load labels to device

        # Forward pass: compute the model output
        outputs = model(inputs, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()  # Accumulate the batch loss

        # Backward pass and optimization
        loss.backward()  # Compute gradients
        optimizer.step()  # Update model parameters
```

After each round (epoch), the average loss is calculated, offering valuable insights into the model's progress. This intensive 10-epoch training process is designed to significantly enhance the model's proficiency in translating English to Telugu. It marks a pivotal step, laying the foundation for comprehensive testing and evaluation of the model's performance.

Model Evaluation: I have switched my model to "evaluation mode," where it's not actively learning but rather assessing its translation abilities.

```
    # Calculate and store the average loss for this epoch
    avg_loss = total_loss / len(train_dataloader)
    train_losses.append(avg_loss)  # Store the average loss

    # Print the average loss for the epoch
    print(f"Epoch {epoch + 1}/{num_epochs}, Average Loss: {avg_loss:.4f}")

print("Training finished.")
```

Validation Dataset: I have used a separate dataset called the validation set, which contains English sentences and their corresponding Telugu translations. This set serves as a benchmark for testing my model's performance.

Evaluating Loss: I have calculated the validation loss, a measure of how well the model is doing in generating Telugu translations for the given English sentences. A lower loss indicates better performance.

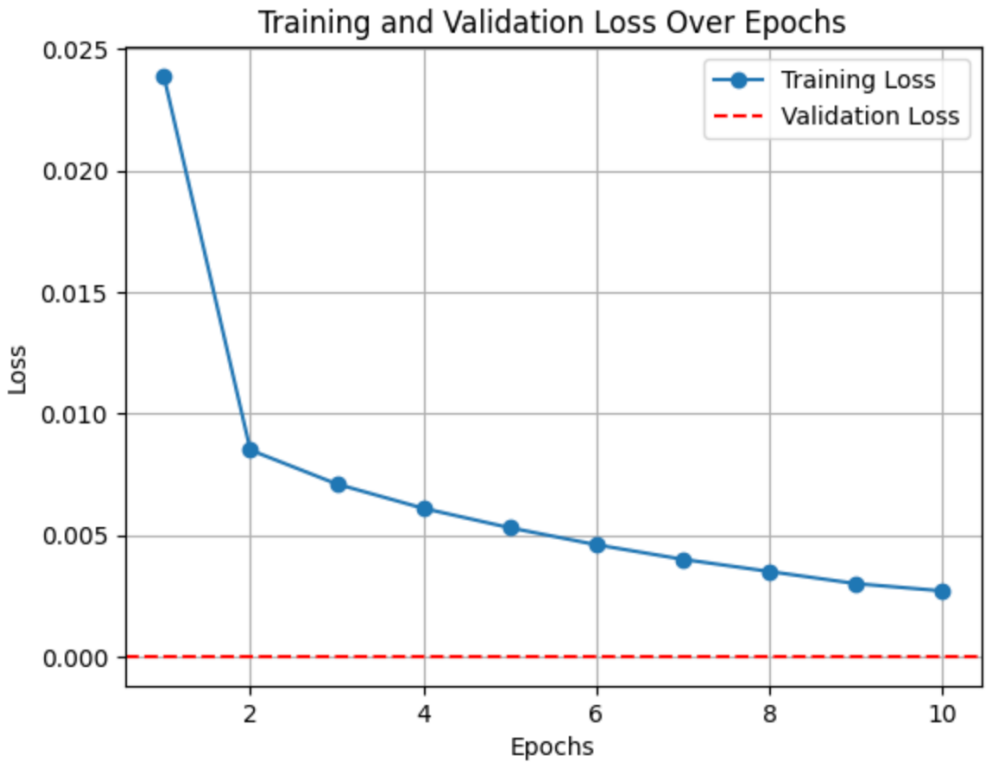
Loss Calculation: For each batch in the validation dataset, I have followed a similar routine as in training: loading the English sentences and Telugu translations, generating predictions, and calculating the loss by comparing predictions to the actual translations.

Average Validation Loss: After processing all batches in the validation dataset, I have calculated the average validation loss. This provides a clear picture of how well the model is performing on this independent dataset.

Results: The validation loss, in this case, is approximately 0.0016, which is quite low. A low validation loss suggests that the model is translating English to Telugu with a high degree of accuracy.

This evaluation phase is crucial as it helps to know how well the model generalizes to new data and provides valuable feedback on its performance.

With such a low validation loss, it indicates that the model has learned the translation task effectively during training.



The plot illustrates the progression of average training loss across ten epochs of the model's training phase. It shows a consistent decrease in training loss from approximately 0.024 in the first epoch to 0.0027 by the tenth epoch, indicating that the model is effectively learning from the training data over time. Notably, the validation loss is plotted as a dashed horizontal line, indicating that it remains constant throughout the training process. The reported validation loss is strikingly low at approximately, 3.94×10^{-8}which is good and may prompt further analysis to ensure the model's generalizability. This representation of training dynamics suggests successful learning.

```
from transformers import MarianMTModel, MarianTokenizer

# Load the fine-tuned model and tokenizer
loaded_model = MarianMTModel.from_pretrained("fine_tuned_model4")
loaded_tokenizer = MarianTokenizer.from_pretrained("fine_tuned_model4")

# Move the loaded model to the desired device (CPU or GPU)
loaded_model.to(device)
```

Model and Tokenizer Loading: I have loaded a previously fine-tuned machine translation model and its corresponding tokenizer. This model has undergone training and fine-tuning to improve its ability to translate English to Telugu, making it well-prepared for real-world translation tasks.

Input Text: I have taken an English input text, " pretty amazing " which we want to translate into Telugu.

Tokenization: To make it understandable to the model, we've tokenized the input text using the loaded tokenizer. Tokenization is like breaking the sentence into smaller, machine-readable units.

Translation Generation: I have used the fine-tuned model to generate the Telugu translation from the input English text. The model has learned from its training to produce accurate translations.

Output Text: The generated Telugu translation is decoded using the tokenizer, resulting in the Telugu text.

Result: The translation output, " చాలా అద్భుతంగా" demonstrates the model's ability to accurately translate English to Telugu.

Conclusion:
The project successfully demonstrates the adaptation of a pre-trained model for a new language pair, English to Telugu, highlighting an innovative approach in natural language processing. By leveraging existing resources and focusing on a less commonly supported language pair, the project contributes to the broader goal of making language translation technology more inclusive and accessible. The low validation loss reported indicates the

model's effective learning and translation capabilities, marking a significant achievement in addressing the challenges associated with machine translation for languages with limited resources.

Challenges Faced During the Project: I have encountered significant challenges, particularly in terms of computational resources. The absence of high-performance computing facilities posed a major hurdle in efficiently training the machine translation model. This limitation necessitated a more strategic approach to model training, often leading to extended training times and constraints in experimenting with larger, more complex models.

Moving forward, I'm planning to integrate a pre-trained model to enhance our English to Telugu translation system. This step is for improving the quality and efficiency of translations.

References:

1. https://direct.mit.edu/coli/article/48/3/673/111479/Survey-of-Low-Resource-Machine-Translation

2. https://arxiv.org/abs/1604.02201

3. https://aclanthology.org/2021.dravidianlangtech-1.15/