Project Report

Team: Sleepy Heads

Pamba Ravindra Mohith,
Ramavath Sai Srithan,
Ramaswamy Santhosh Reddy,
Thota Sri Lakshmi Teja,
Karanam Lokesh.

210050112
210050132
210050131
210050158
210050081

CS 337: AI ML

Goal of the Project:

This project focuses on building a robust machine translation model for English-to-Hindi using the state-of-the-art Helsinki-NLP/opus-mt-en-hi pre-trained model from Hugging Face Transformers. The primary objectives are to load and preprocess the CFILT IITB English-Hindi dataset, fine-tune the model, save it for future use, and demonstrate its translation capabilities.

1 Introduction

Machine translation, the automated process of converting text from one language to another, has witnessed significant advancements owing to the development of sophisticated natural language processing (NLP) models. This project centers on the task of translating English text to Hindi, utilizing the vast potential of transformer-based architectures and leveraging the CFLIT-IITB dataset, which contains parallel English-Hindi text pairs.

The primary objective of this endeavor is to build a robust and accurate machine translation system that effectively bridges the linguistic gap between English and Hindi. To achieve this, the project relies on the Helsinki-NLP model, a cutting-edge transformer-based architecture.

This introduction sets the stage for a comprehensive exploration of the methodologies employed, challenges encountered, and the outcomes attained in the pursuit of developing an efficient machine translation system utilizing the `CFILT-IITB` dataset and the Helsinki-NLP model for English to Hindi translation.

2 Objectives

The objectives of this project are outlined as follows:

1. Implement a machine translation system capable of translating English text to Hindi.
2. Utilize the CFLIT-IITB dataset for training and evaluation purposes in the development of the translation model.
3. Evaluate the performance of the translation model using various metrics, including Jaccard, Cosine Similarities and BLEU score.
4. Deploy the Helsinki-NLP model using the Hugging Face Interface effectively to create a functional translation service for English to Hindi translation.

3 Dataset

The CFILT-IITB dataset, encompassing parallel English-Hindi text pairs, was acquired using the Hugging Face **datasets** library. The dataset arrived pre-segmented into three distinct parts:

- Train set comprising 1.66 million instances.
- Validation set containing 520 instances.
- Test set consisting of 2.51 thousand instances.

Due to resource constraints, we opted to limit the number of instances in the train, validation, and test sets as follows:

- Initially, the pre-segmented data was combined into a unified dataset, followed by a random shuffling process. The objective of shuffling was to ensure that the new subset of 30,000 instances could effectively represent the entirety of the original dataset.
- From the 30,000 instances, we conducted a split into train, test, and validation sets using an 80-10-10 ratio.

This process allowed for a more manageable subset while maintaining a representative distribution of instances across the train, validation, and test sets, enabling a comprehensive and efficient exploration of the dataset within our resource constraints.

- The dataset is structured with a single column labeled "Translations," where each row consists of a dictionary. These dictionaries include pairs of language codes and their corresponding translated text.
- The data is organized in a structured manner, with each dictionary serving to provide translations for different languages. For instance, consider the following example:

$$\{"en" : "My\ name\ is\ Alex",\ "hi" : ""\}$$

4 The Model

4.1 Model Preprocessing

The tokenizer, leveraging TFAutoModelForSeq2SeqLM from the Helsinki-NLP model, transforms raw text into a sequence of token IDs, enabling the neural network model's processing.

4.1.1 Subword Tokenization

Helsinki-NLP employs subword tokenization strategies, such as Byte Pair Encoding (BPE) or SentencePiece. These techniques break down words into smaller units (subwords), facilitating the handling of rare or unknown words and enhancing translation accuracy. For example, an unseen word during training can be decomposed into smaller known subwords, allowing the model to effectively translate it. The Helsinki-NLP tokenizer, specifically designed for the "Helsinki-NLP/opus-mt-en-hi" model, operates on the principles of neural machine translation tokenization.

4.1.2 Examples

Single Sentence:

$$\text{tokenizer("Hello, this is a sentence!")}$$

$$\{'input\_ids' : [12110, 2, 90, 23, 19, 8800, 61, 0], 'attention_mask' : [1, 1, 1, 1, 1, 1, 1, 1]\}$$

4.2 Helsinki-NLP Opus-MT Model

4.2.1 Model Source

The Helsinki-NLP Opus-MT model is available here

4.2.2 Encoder Layers

The model is structured with 6 encoder layers, each employing the following mechanisms:

Self-Attention Mechanism: Each layer in the encoder utilizes self-attention to concurrently process the entire input sequence. This mechanism empowers the model to evaluate the significance of different words in the sentence in relation to each other.

Feed-Forward Neural Network: Following the attention layer, a feed-forward neural network processes the output sequentially. This network applies non-linear transformations to enhance the representation of the input.

4.2.3 Decoder Layers

Similarly, the model features 6 decoder layers, each incorporating the following components:

Masked Self-Attention: Similar to the encoder's self-attention, the masked self-attention in the decoder is 'masked' to prevent the decoder from peeping into future tokens in the sequence. This ensures that predictions for a word at a position depend solely on the known words before it.

Encoder-Decoder Attention: In this step, the decoder layers focus their attention on the encoder's output. This process is vital for aligning the input

(source language) and output (target language) sequences, playing a pivotal role in ensuring translation quality.

Feed-Forward Neural Network: Just like in the encoder, each decoder layer is equipped with a feed-forward network that further processes the sequence, contributing to the overall complexity and expressiveness of the model.

4.2.4 Capabilities

Language Pairs:
The model is specifically trained for English to Hindi translation. It is part of the larger Opus-MT project, which encompasses a wide array of language pairs.

Architecture
The model utilizes the TFAutoModelForSeq2SeqLM architecture from the Transformers library.

Dataset
It has been trained on diverse and extensive datasets, including the IITB English-Hindi dataset.

4.2.5 Key Advantages

- Demonstrates high-quality translation capabilities.
- Leverages advanced tokenization and training techniques.

4.2.6 Application Areas

The model proves effective for various natural language processing tasks, with a particular emphasis on English to Hindi translation.

4.3 Working of Helsinki-NLP Opus-MT Model

The Helsinki-NLP/opus-mt-en-hi model is purpose-built for translation between English and Hindi.

4.3.1 Input Processing

The model initiates the process by breaking down the input sentence into tokens, which can be words or sub-words.

4.3.2 Encoding

The encoder layers, consisting of 6 layers with attention mechanisms, meticulously analyze the input sentence. This analysis enables the model to comprehend the contextual intricacies and nuances of the sentence. Each layer contributes to a progressively deeper understanding.

4.3.3 Decoding

Following the encoding phase, the decoder layers take charge of generating the Hindi translation token by token. Leveraging the context provided by the encoder and its own previous outputs, the decoder predicts the subsequent word. With 6 layers and attention mechanisms, the translation is ensured to be contextually appropriate and grammatically sound.

4.3.4 Output

The model yields the translated Hindi sentence corresponding to the input sentence.

5 Hyperparameters and Fine Tuning

After utilizing grid search to tune the parameters, we transitioned to using the best parameters directly, omitting the grid search due to its time-consuming and resource-intensive nature. The hyperparameters employed are as follows:

- **Batch Size:** 16
- **Learning Rate:** 2e-5
- **Weight Decay:** 0.01
- **Number of Epochs:** 1

Additionally, we utilized the **AdamWeightDecay** optimizer to manage weight training efficiently.

AdamW Optimizer Overview

The AdamW optimizer, short for "Adam Weight Decay," is a variant of the Adam optimizer that includes an additional term for weight decay regularization. It aims to address weight decay issues observed in the original Adam optimizer.

1. **Adam Optimizer Overview:**
    - **Adam (Adaptive Moment Estimation):** Adam is an adaptive learning rate optimization algorithm that combines ideas from RMSprop and momentum. It maintains per-parameter learning rates and exponentially moving averages of gradients' squares and gradients, respectively.

- **Adaptive Learning Rates and Momentum:** Adam adjusts learning rates for each parameter and includes momentum to accelerate gradients in the right direction.

2. **AdamW - Addressing Weight Decay Issue:**
   - **Weight Decay:** Weight decay prevents overfitting by penalizing large weights in the loss function (equivalent to L2 regularization).
   - **Issue in Adam:** The original Adam optimizer doesn't correctly implement weight decay, leading to different training behaviors and potential performance issues.
3. **AdamW - Optimizing with Weight Decay:**
   - **Additional Term for Weight Decay:** AdamW explicitly adds the weight decay regularization term to address the issue, ensuring proper weight decay during optimization.
   - **Improvement and Benefits:** AdamW improves training stability and potentially enhances model generalization by correctly applying weight decay.
4. **Key Features and Benefits of AdamW:**
   - **Improved Regularization:** Ensures proper weight decay implementation, enhancing model generalization.
   - **Stable Training:** Helps maintain stable training dynamics by correctly applying weight decay.
5. **Usage:**
   - **Parameters:** Accepts parameters similar to Adam (e.g., learning rate, weight decay rate, beta parameters).
   - **Implementation:** Available in popular deep learning libraries like PyTorch and TensorFlow as an optimizer option.
6. **Considerations:**
   - **Hyperparameter Tuning:** Still requires tuning hyperparameters like learning rate, weight decay rate, etc., based on the specific problem and dataset.

Conclusion: AdamW, an extension of the Adam optimizer, addresses weight decay issues by incorporating proper weight decay regularization during optimization. Its improvements in stabilizing training dynamics and ensuring correct weight decay implementation make it a valuable optimization technique in deep learning.

6 Data Processing Steps

For the following steps, we limit the length of the sequences for both source and target to 128.
The Tokenizer used in the project is a pre-defined tokenizer which is loaded from the **Helsinki NLP** model. The Preprocessing function defined in the code is using the above tokenizer for both the source language instances as well as the target language instances.
Finally ,we use data collators with model and tokenizer from Helsinki-NLP model with return datatype as a TensorFlow tensor.
The data collator used in this project is **DataCollatorForSeq2Seq** which is a utility class typically used in Hugging Face's **transformers** library for sequence-to-sequence tasks. It helps in collating and formatting data for training sequences by handling padding, truncation, and preparing inputs and labels for the model.

7 Evaluation Metrics

The evaluation metrics used in this project are:

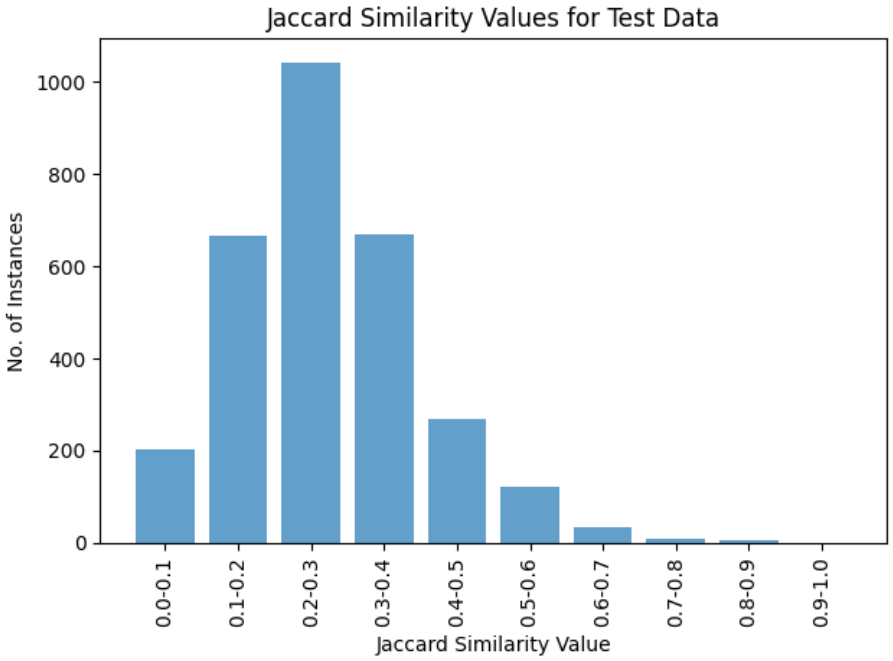
7.1 Jaccard Similarity

Jaccard similarity is a statistical measure used to compare the similarity and dissimilarity between two sets. It measures the intersection over the union of two sets and is particularly useful in cases where the presence or absence of elements matters more than their frequency or order.

The Jaccard similarity coefficient is calculated as the ratio of the size of the intersection of the sets to the size of the union of the sets:

$$\text{Jaccard Similarity}(\mathbf{A}, \mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|}$$

For our analysis, we divided the translated dataset into 10 classes based on Jaccard similarity ranges, such that the first class elements have a Jaccard similarity between 0 and 0.1, the second class between 0.1 and 0.2, and so forth. We utilized this classification to create a bar graph for easy interpretation of the data.



However, Jaccard Similarity has certain limitations:

- **Sensitive to Set Size:** Jaccard similarity might not be ideal when dealing with sets of significantly different sizes.
- **Disregard for Element Frequency:** It does not consider the frequency or importance of elements within the sets, treating all elements equally, which may not be suitable in contexts where element importance matters.
- **Limited for Text Comparison:** In text comparison, Jaccard similarity does not consider the sequence or order of words, treating text as a bag of words, disregarding grammatical structure and word order.

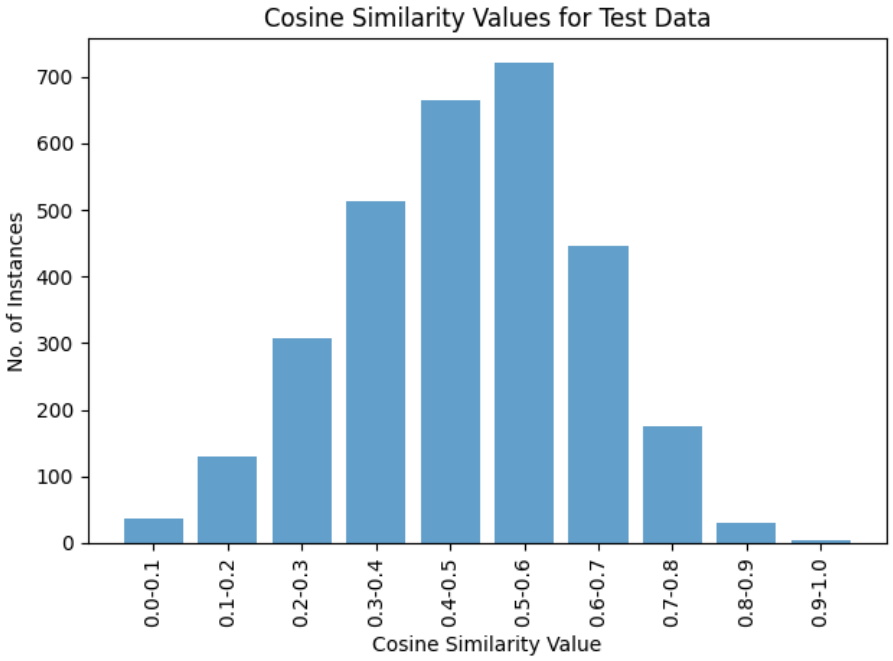
These limitations imply that Jaccard Similarity might not serve as a fully representative metric for our project.

7.2 Cosine Similarity

In the context of machine translation, Cosine Similarity measures the similarity between translated sentences represented as vectors in a high-dimensional space. It allows us to assess the quality and fidelity of translations by comparing the orientation of vectors obtained from the source and translated texts. The Cosine Similarity between the source ($\mathbf{S}$) and translated ($\mathbf{T}$) sentences or documents can be calculated using the formula:

$$\text{Cosine Similarity}(\mathbf{S}, \mathbf{T}) = \frac{\mathbf{S} \cdot \mathbf{T}}{\|\mathbf{S}\| \cdot \|\mathbf{T}\|}$$

Here, $\mathbf{S} \cdot \mathbf{T}$ represents the dot product of the source and translated sentence vectors, while $\|\mathbf{S}\|$ and $\|\mathbf{T}\|$ denote their respective Euclidean norms.
In our machine translation evaluation, we used Cosine Similarity to determine the closeness of translated sentences to their corresponding source sentences. This metric assists in gauging the level of semantic equivalence and accuracy between the original and translated texts.
To visually represent the evaluation results, we created a bar graph plotting the Cosine Similarity scores for various translated sentences against their respective source sentences. This graph aids in illustrating the degree of similarity achieved by the translation model across different sentence pairs.



However, while Cosine Similarity offers advantages such as being invariant to vector magnitude and suitable for high-dimensional spaces, it does have limitations when applied in machine translation evaluation:

- **Sensitivity to Sentence Length and Complexity:** Cosine Similarity might be sensitive to differences in sentence lengths and structural complexities, potentially affecting similarity scores.
- **Semantic Gap and Polysemy:** It might not capture nuances in meaning, especially in cases of polysemous words or when there's a semantic gap between languages.

Despite these limitations, Cosine Similarity can provide valuable insights into the degree of semantic similarity between source and translated sentences, aiding in the evaluation of machine translation systems.

7.3 BLEU Score

The BLEU (Bilingual Evaluation Understudy) score is a popular evaluation metric used to assess the quality of machine-translated text by comparing it to one or more human reference translations. It measures the similarity between the machine-generated translation and the reference translations based on n-gram precision.
The BLEU score is computed by comparing n-grams (sequences of n contiguous tokens) between the machine-generated translation and the reference(s), evaluating the precision of these n-grams in the translation. It then calculates a geometric mean of the n-gram precisions, penalizing shorter translations.
Mathematically, the BLEU score is expressed as:

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} \frac{1}{N} \log \text{p}_n\right)$$

Here, BP is the brevity penalty to account for shorter translations, and p_n represents the modified precision for n-grams up to a certain length N.
In our machine translation evaluation, we utilized the BLEU score as a key metric to quantitatively assess the quality of translated texts produced by our model.
However, while BLEU is widely used, it has certain limitations:

- **N-gram Precision Emphasis:** BLEU heavily relies on n-gram matching, which might not fully capture fluency or coherence.
- **Insensitive to Semantic Meaning:** It does not directly measure semantic equivalence, potentially rewarding literal translations that differ in meaning.
- **Subjectivity of Reference Translations:** The choice of reference translations can influence the BLEU score, and using a limited set of references may not capture all possible translations.

Nevertheless, the BLEU score remains a widely used and valuable metric for evaluating machine translation systems.
The BLEU score obtained in the project is **48.76**.
BLEU Score Range:

- **0 to 19:** Low BLEU score, indicating poor translation quality.
- **20 to 39:** Fair BLEU score, suggesting moderate similarity.
- **40 to 59:** Decent BLEU score, indicating reasonably good translation.
- **60 to 79:** Good BLEU score, representing a high degree of similarity.
- **80 to 99:** Very good BLEU score, indicating high-quality translations closely matching reference translations.
- **100:** Perfect BLEU score, although doesn't necessarily guarantee flawless translations.

8 Limitations

This project encounters several limitations, primarily in the translation of nouns, attributable to both the training data and the model architecture:

- When the source sentence disproportionately contains nouns, the model may misinterpret some nouns, leading to erroneous or random translations.
- The use of capitalization in nouns significantly impacts their translation, as the model differentiates between capitalized and lowercase nouns.

Additionally, specific limitations arise due to the implementation of **Byte Pair Encoding (BPE)**:

- **Subword Segmentation:** BPE segments words into subunits based on frequency in the training corpus. Infrequent or inconsistently represented nouns and case forms might not be adequately captured by BPE, potentially compromising the model's ability to accurately translate these elements.
- **Subword Ambiguity:** The segmentation process may yield ambiguous subunits for words with multiple meanings or grammatical cases. This ambiguity can hinder the model's capacity to differentiate between various noun forms or cases.
- **Subword Merging:** During the BPE process, continuous merging of subunits can lead to suboptimal representation of noun-specific units, especially in terms of case sensitivity, which is crucial for accurate translation.
- **Subword Representations and Case Sensitivity:** The BPE algorithm might not explicitly maintain case information in its tokens. This lack of explicit case representation can result in challenges in preserving case sensitivity during translation, affecting the accuracy of noun translations.

These limitations underscore the importance of considering both the lexical diversity and case sensitivity in training data, as well as the inherent constraints of the BPE methodology in machine translation models.