Project Report

On

Code – Mixed Text Translation

Submitted

In partial fulfilment

For the award of the Degree of

PG-Diploma in Artificial Intelligence(PG-DAI)

C-DAC, ACTS (Pune)

Guided By:

Mr. Prakash Sinha

Submitted By:

Kunal Ukey	*240340128011*
Rishikesh Patil	*240340128022*
Riya Wagh	*240340128024*
Rutuja Barbande	*240340128026*
Yogesh Siral	*240340128037*

Centre for Development of Advanced Computing(C-DAC), ACTS

(Pune- 411008)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Mr. Prakash Sinha.** C-DAC ACTS, Pune for her constant guidance and helpful suggestion for preparing this project Code-Mixed Text Translation**.** We express our deep gratitude towards her for inspiration, personal involvement, constructive criticism that she provided us along with technical guidance during the course of this project.

We take this opportunity to thank Head of the department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment for our overall development.

We express sincere thanks to **Mrs. Namrata Ailawar**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P R (Program Head)** and **Mrs. Srujana Bhamidi** (Course Coordinator, PG-DAI) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Kunal Ukey	240340128011
Rishikesh Patil	240340128022
Riya Wagh	240340128024
Rutuja Barbande	240340128026
Yogesh Siral	240340128037

ABSTRACT

In this project, we developed a translation system designed to handle code-mixed text, with a focus on translating Indic languages, particularly Marathi, into English. Code-mixed text, which is a blend of words and phrases from different languages, presents unique challenges for traditional translation models due to its hybrid nature.

Our work builds on the Seq2Seq translation model, which we adapted to address the unique challenges of code-mixed language. By focusing on Marathi, a widely spoken language in India, our project addresses the increasing need for effective translation tools in multilingual contexts. The translation model is trained on a diverse dataset that includes various code-mixed scenarios, enabling it to handle the nuances and complexities of mixed-language sentences.

The ultimate goal of this project is to create a robust and efficient translation system that can be extended to other Indic languages, helping to bridge language barriers and facilitate communication in diverse linguistic settings.

Table of Contents

1. Introduction

The modern world is consumed by social media platforms like Instagram and Twitter. Similarly, platforms such as YouTube and Reddit have seen significant surge in user interaction, particularly in the comments sections of videos. These posts and comments often reflect public opinion and frequently involve discussions on social, political, and other relevant topics. This has led to the common practice of users communicating on social media using mixed languages, a phenomenon known as code-mixing. Code-mixing involves blending words from multiple languages while maintaining the script of a single language. Typically, the Latin script is employed to represent words from various languages. For example, a text could be written in Marathi using the Latin script rather than its traditional Devanagari script.

In this project, we developed an Encoder-Decoder-based Neural Machine Translation (NMT) system using Sequence to Sequence (Seq2Seq) learning with Recurrent Neural Networks (RNN). This system is designed to translate Marathi code-mixed text written in Latin script into English. For example, the system can translate “tu kasa ahe?” to “how are you?”.

1.1. Overview

Language translation has long been a pivotal area of research in natural language processing (NLP), especially as globalization continues to bridge diverse cultures and communities. With the increasing need for effective communication across languages, machine translation systems have gained significant importance. This project focuses on the development of an Encoder-Decoder model utilizing Recurrent Neural Networks (RNNs) for translating Marathi code-mixed in Latin script into English. Marathi, an Indic language spoken predominantly in the Indian state of Maharashtra, often incorporates English words and phrases in everyday conversation, resulting in a unique linguistic phenomenon known as code-mixing. This presents a fascinating challenge for machine translation systems, as they must accurately interpret and convert not only the grammatical structures but also the cultural nuances embedded within the language.

1.2. Problem Statement

The rapid globalization and technological advancement have led to increased interaction among speakers of different languages, creating a pressing need for effective communication tools In India, the use of code-mixing—particularly the blending of Marathi with English—has become a prevalent mode of communication, especially among younger generations and in informal contexts such as social media, text messaging, and casual conversations. This phenomenon reflects the dynamic nature of language and the cultural interplay between English and regional languages. However, the complexity of code-mixed language presents significant challenges for existing machine translation systems, which are primarily designed to handle standardized language pairs. Current machine translation models often struggle to accurately translate code-mixed Marathi sentences due to several factors:

1. **Linguistic Complexity**: Code-mixed sentences frequently exhibit non-standard grammar, vocabulary, and syntax, making them difficult for traditional models to process. The integration of English words and phrases within Marathi sentences can alter meaning and context, leading to potential misinterpretations.
2. **Lack of Training Data:** There is a scarcity of high-quality, annotated datasets specifically tailored for code-mixed language translation. Most existing datasets focus on either pure Marathi or English, neglecting the unique characteristics of code-mixed usage.
3. **Cultural Nuances:** Code-mixing often incorporates cultural references, idiomatic expressions, and slang that may not have direct equivalents in English. This necessitates a translation model that can understand and convey these nuances effectively.

1.3. Objectives

1. **Model Development**:
    - Design and implement a sequence-to-sequence model using RNNs for language translation.
    - Integrate attention mechanisms to enhance translation accuracy.

- To design and implement an Encoder-Decoder architecture based on Recurrent Neural Networks (RNNs) specifically tailored for translating Marathi code-mixed in Latin script into English.

2. **Data Collection and Transformation:**

- Collect a diverse dataset of code-mixed Marathi sentences and their English translations.
- Preprocess the dataset to ensure it is suitable for training, including tokenization, normalization, and encoding.
- To curate a comprehensive dataset that includes a diverse range of code-mixed Marathi sentences paired with their English translations.
- To perform data preprocessing steps such as tokenization, normalization, and handling of special characters to prepare the dataset for training.

3. **Training and Optimization:**
   - Train the model using techniques such as teacher forcing and gradient clipping to prevent exploding gradients.
   - Optimize hyperparameters, including learning rate, batch size, and number of epochs, to improve model performance.
   - To optimize the model parameters using appropriate loss functions and optimization algorithms (e.g., Adam) to enhance translation quality.

4. **Evaluation:**
   - Implement evaluation metrics such as BLEU, METEOR, and ROUGE scores to assess translation quality.
   - Conduct qualitative evaluations through human assessments to gauge fluency and accuracy.

5. **User Interface (Optional):**

- Developed a simple web-based for users to input code-mixed Marathi sentences and receive translations in real-time.

2. Literature Review

Researchers have delved into the integration of telematics devices within insurance companies, a domain that has garnered substantial attention among industry experts. Machine translation (MT) has evolved significantly over the past few decades, beginning with rule-based approaches in the 1950s and progressing to statistical models in the 1990s. Early MT systems, such as those based on the interlingua and transfer-based methods, relied heavily on linguistic rules and were limited by their inability to handle the vast complexities of human language.

The introduction of Neural Machine Translation (NMT) in the mid-2010s revolutionized the field, offering a more holistic approach to translation by directly modelling the entire source sentence as a sequence of words, rather than relying on phrases. The Seq2Seq architecture, proposed by Sutskever et al. (2014), was a ground breaking development, leveraging Recurrent Neural Networks (RNNs) to map sequences of words from the source language to the target language. The incorporation of the attention mechanism by Bahdanau et al. (2015) further enhanced the capability of NMT models by allowing them to focus on specific parts of the input sequence when generating each word in the translation.

NMT systems have since become the de facto standard in machine translation, with Transformer models introduced by Vaswani et al. (2017) setting new benchmarks in translation quality. Transformers, with their ability to process entire sentences in parallel and capture long-range dependencies, have proven highly effective in a wide range of language pairs, including those with complex grammatical structures.

The study of code-mixed language translation has seen a growing body of work in recent years. Early efforts, such as those by Vyas et al. (2014), explored the structural and linguistic characteristics of code-mixed text, laying the groundwork for later computational models. The work by Solorio and Liu (2008) was pioneering in the automatic identification of code-mixed text, which is a critical first step in building effective translation systems.

With the rise of NMT, researchers have begun to adapt Seq2Seq models to handle code-

mixed language pairs. Studies by Gupta et al. (2018) and Pratapa et al. (2018) have shown that using data augmentation techniques, such as artificially generating code-mixed sentences, can improve the performance of translation models. The inclusion of attention mechanisms has also been shown to help models better capture the nuances of code-mixed language.

However, much of the research to date has focused on Hindi-English code-mixed text, with relatively little attention given to other language pairs. The L3-Cube MahaNLP Pune project, for instance, has made significant contributions by providing datasets specifically for Marathi-English code-mixed text, which has been underrepresented in the literature.

Research specific to Marathi-English code-mixed translation is still in its nascent stages. Most existing studies have concentrated on larger language pairs, leaving a gap in the literature regarding smaller or regional languages like Marathi. This study aims to address this gap by employing a Seq2Seq RNN model to translate Marathi-English code-mixed text, building on the foundational work in NMT while adapting it to the unique challenges posed by code-mixing.

3. Implemented System

3.1 Existing System Architecture:

There are various stages and methods involved in developing a language translation model, from rule-based systems to cutting-edge neural machine translation (NMT) models. Below is a summary of the potential strategies:

Rule-Based Translation Systems (RBMT):

- Dictionaries and Grammar Rules: This method of translating text makes use of bilingual dictionaries and linguistic rules. Grammar agreements, word order, and sentence structure rules are manually established.
- Analysis of Morphology: The morphology of Marathi is rich in inflections.
- Morphological analysis is a tool that rule-based systems can use to deal with word forms.
- Difficulties: This method may not be able to handle idiomatic expressions, complex sentences, or translations based on context.

Statistical Machine Translation (SMT):

- Parallel Corpora: SMT models are trained using sizable corpora of English-Marathi text. The probabilities of word and phrase alignments between the two languages are learned by the system.
- Phrase-Based Models: SMT treats phrases, or word sequences, as translation units as opposed to individual words. This aids in providing additional context for the translation.
- Difficulties: SMT frequently has trouble translating with fluency and coherence and needs a lot of parallel text data.

Neural Machine Translation (NMT):

- Sequence-to-Sequence (Seq2Seq) Models: These models are made up of a decoder that creates the target sentence and an encoder that processes the

original sentence. They are trained on parallel corpora in an end-to-end fashion.

- The attention mechanism was added to NMT models so that the decoder could concentrate on pertinent portions of the original sentence during translation. This enhances the quality of the translation, particularly for lengthy sentences.
- Transformers: Compared to conventional Seq2Seq models, modern NMT models frequently employ Transformer architectures, which are better suited to handle long-range dependencies.
- Difficulties: A significant amount of training data and processing power are needed for NMT models. Additionally, when translating uncommon words or languages with little training data, they might translate less accurately.

Hybrid Methodologies:

- Combining SMT and NMT: To improve fluency and coherence, some systems combine the benefits of SMT (such as phrase translations) with NMT.
- Post-Editing: Rule-based or statistical techniques can be used to improve the translation after an NMT model has generated translations, particularly for particular Marathi linguistic features.

Pretrained Multilingual Models:

- Transfer Learning: Models such as mBERT or mT5, pretrained on English and Marathi, can be used to fine-tune for the particular translation task.
- Zero-Shot or Few-Shot Learning: By utilizing prior knowledge from similar languages or tasks, these models are able to translate even with a small amount of parallel data.

End-to-End Neural Models with Pretraining:

- Pretrained Language Models: Robust translation models can be produced by optimizing massive language models such as GPT, BERT, or T5 on multiple

parallel datasets.

- Custom Architectures: By creating custom architectures specifically for Marathi, taking into account its distinct linguistic characteristics, translation quality may be enhanced.

Unsupervised and Semi-Supervised Learning:

- Unsupervised neural machine translation (NMT): This method uses training on bilingual monolingual corpora to learn translations without explicitly providing parallel data. Techniques like back-translation are employed.
- When parallel data is scarce, semi-supervised neural machine translation (NMT) improves translation quality by combining smaller amounts of parallel data with larger monolingual corpora.

Data Augmentation Techniques Back-Translation:

- Model performance can be enhanced by creating artificial parallel data by translating Marathi sentences to English and back again.
- Parallel Corpus Expansion: Methods for enlarging parallel corpora include human annotation and aligning bilingual sentences from Wikipedia and other sources.

Evaluation and Fine-Tuning BLEU Score and Human Evaluation:

- While model performance can be evaluated automatically using metrics like the BLEU score, human evaluation is essential for determining fluency and adequacy.

Deployment Considerations:

- Real-Time Translation: Model speed and latency optimization is crucial for applications that need real-time translation.
- API Integration: Using an API to deploy the model for usage in websites, mobile apps, chatbots, and other applications.

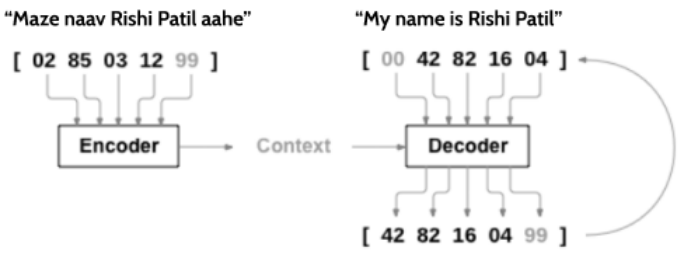
3.2 Proposed System

3.2.1 Requirements

```
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import re
import random

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F

import numpy as np
from torch.utils.data import TensorDataset, DataLoader, RandomSampler

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

3.2.2 Loading data files:

We will be teaching a neural network to translate from Marathi to English in this project. In this project, two recurrent neural networks collaborate to convert one sequence to another using a sequence-to-sequence network. An input sequence is compressed into a vector by an encoder network, and that vector is then transformed into a new sequence by a decoder network.



For later use as the inputs and targets of the networks, we'll require a unique index for each word. We will use a helper class called Lang to keep track of everything. It contains word → index (word2index) and index → word (index2word) dictionaries, in

addition to a count of every word (word2count), which will be used in the future to replace uncommon words.

All of the files are in Unicode; to make things simpler, we will convert Unicode characters to ASCII, lowercase everything, and remove most punctuation.

```
# Turn a Unicode string to plain ASCII, thanks to
# https://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# Lowercase, trim, and remove non-letter characters
def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r"([.!?])", r" \1", s)
    s = re.sub(r"[^a-zA-Z!?]+", r" ", s)
    return s.strip()
```

We will divide the file into lines and then divide the lines into pairs in order to read the data file. Since all of the files are English → Marathi, I added a reverse flag to reverse the pairs if we want to translate from Marathi → English.

We'll condense the data set to only include comparatively short and straightforward sentences because there are a lot of example sentences and we want to train something quickly. Here, the maximum sentence length is ten words (including closing punctuation), and we're only selecting sentences that take the form "I am," "He is," etc.

The entire procedure for getting the data ready is:

- Split the text file into lines, then into pairs of lines.
- Normalize text and apply content and length filters.
- Create word lists in pairs using the sentences.

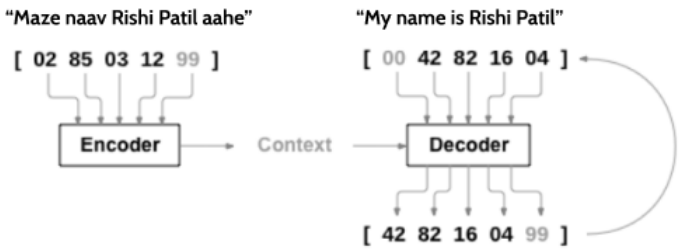
```
def prepareData(lang1, lang2, reverse=False):
    input_lang, output_lang, pairs = readLangs(lang1, lang2, reverse)
    print("Read %s sentence pairs" % len(pairs))

    # Check if pairs is populated correctly
    if not pairs:
        print("No pairs found. Check the data source.")
        return input_lang, output_lang, pairs

    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs

import random

input_lang, output_lang, pairs = prepareData('eng', 'mar', True)

if not pairs:
    print("No pairs found. Ensure your data and filters are correct.")
else:
    print(random.choice(pairs))
```

```
Reading lines...
Read 30304 sentence pairs
Trimmed to 28626 sentence pairs
Counting words...
Counted words:
mar 11774
eng 5810
['mala school kade pasun parat aale', 'i have just come back from school']
```

The Seq2Seq Model:

A network that works on a sequence and uses its own output as input for later steps is called an RNN, or recurrent neural network.

A model made up of two RNNs known as the encoder and decoder is known as a sequence to sequence network, also known as a seq2seq network or encoder-decoder network. After reading an input sequence, the encoder outputs a single vector, which the decoder then reads to generate an output sequence.

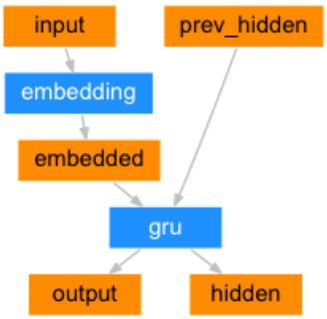


Unlike sequence prediction with a single RNN, where every input corresponds to an output, the seq2seq model frees us from sequence length and order, which makes it ideal for translation between two languages.

Consider the sentence 'maze naav Rishi Patil aahe' → 'My name is Rishi Patil'. Most of the words in the input sentence have a direct translation in the output sentence, but are in slightly different orders.

The Encoder:

The encoder of a seq2seq network is a RNN that outputs some value for every word from the input sentence. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word.



```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input):
        embedded = self.dropout(self.embedding(input))
        output, hidden = self.gru(embedded)
        return output, hidden
```

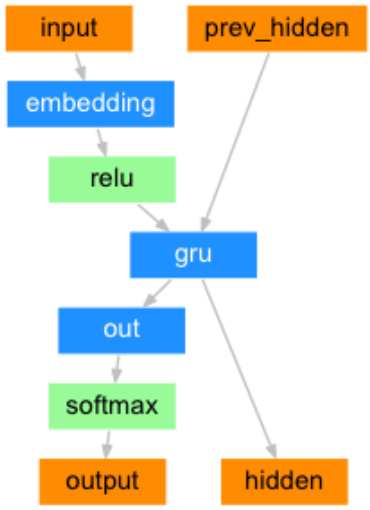
The Decoder:

The decoder is another RNN that takes the encoder output vector(s) and outputs a sequence of words to create the translation.

Simple Decoder

We only use the encoder's final output in the most basic seq2seq decoder. Since it encodes context from the entire sequence, this final output is also referred to as the context vector. The decoder's initial hidden state is this context vector.

The decoder receives an input token and a hidden state at each stage of the decoding process. The start-of-string <SOS> token is the first input token, and the context vector—the encoder's final hidden state—is the first hidden state.



```python
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden  = self.forward_step(decoder_input, decoder_hidden)
            decoder_outputs.append(decoder_output)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        return decoder_outputs, decoder_hidden, None # We return `None` for consistency in the training loop

    def forward_step(self, input, hidden):
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.out(output)
        return output, hidden
```

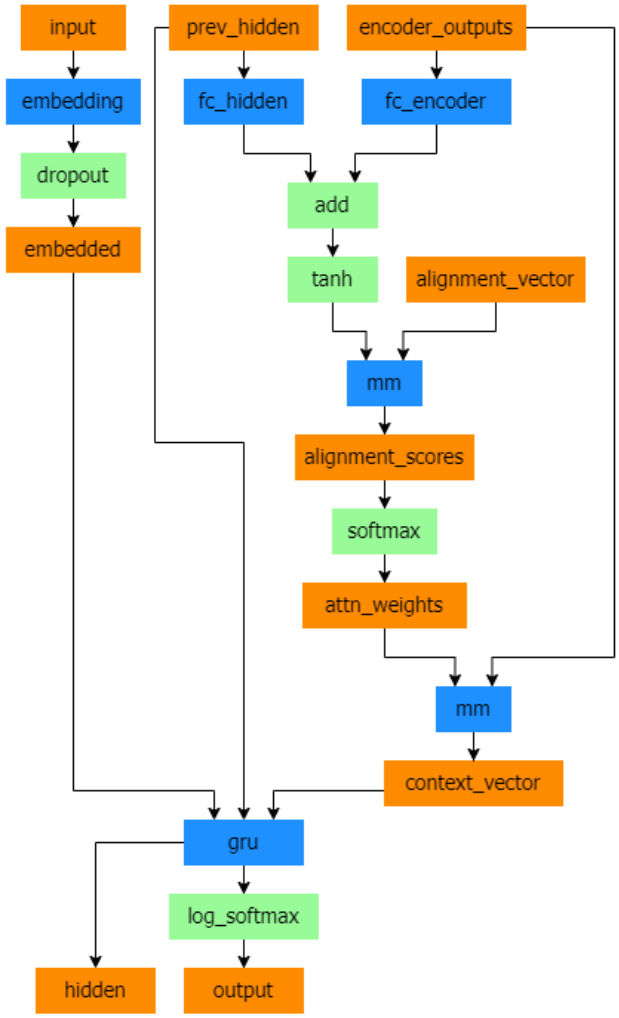
Attention Decoder:

The entire sentence must be encoded if the context vector is the only one shared between the encoder and the decoder.

For each step of the decoder's own outputs, attention enables the decoder network to "focus" on a different portion of the encoder's outputs. We begin by determining a set of attention weights. The encoder output vectors will be multiplied by these to produce a weighted combination. In order to assist the decoder in selecting the appropriate output words, the result (referred to as attn_applied in the code) ought to include details about that particular segment of the input sequence.

Using the input from the decoder and the hidden state as inputs, another feed-forward layer attn calculates the attention weights. We must select a maximum sentence length

(also known as the input length for encoder outputs) that this layer can apply to in order to actually create and train it because the training data contains sentences of all sizes. Shorter sentences will only use the first few attention weights, while sentences of the maximum length will use all of them.



In sequence-to-sequence models, Bahdanau attention, also referred to as additive attention, is a frequently employed attention mechanism, especially in neural machine translation tasks. In their paper Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al. introduced it. To calculate attention scores between the encoder and decoder hidden states, this attention mechanism uses a learned alignment model. It computes alignment scores using a feed-forward neural network.

Other attention mechanisms exist, though. One such mechanism is Luong attention, which calculates attention scores by taking the dot product of the hidden states of the

encoder and the decoder. The non-linear transformation utilized in Bahdanau attention is not involved in this process.

```python
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights
```

```python
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        (parameter) self: Self@AttnDecoderRNN
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden, attn_weights = self.forward_step(
                decoder_input, decoder_hidden, encoder_outputs
            )
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input
        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions
```

```python
def forward_step(self, input, hidden, encoder_outputs):
    embedded =  self.dropout(self.embedding(input))

    query = hidden.permute(1, 0, 2)
    context, attn_weights = self.attention(query, encoder_outputs)
    input_gru = torch.cat((embedded, context), dim=2)

    output, hidden = self.gru(input_gru, hidden)
    output = self.out(output)

    return output, hidden, attn_weights
```

4. Training and Evaluation

4.1 Dataset Overview

The dataset for this project was created by modifying the dataset used in the English-French translation project from PyTorch's Seq2Seq translation tutorial. We removed the French words and replaced them with code-mixed Marathi translations using AI tools like Perplexity AI, effectively generating our own majority dataset. In addition to this, we incorporated datasets from the L3-Cube MahaNLP Pune project group (namely L3Cube-MeCorpus Roman and Devanagari), which provided valuable Marathi-English code-mixed text. To further enrich the dataset, we performed web scraping on popular Marathi Instagram accounts, capturing real-world code-mixed language used in social media contexts. The final dataset consisted of approximate 33000 sentences.

4.2 Dataset Preparation

The training dataset consists of 28,626 sentence pairs after trimming, indicating that some filtering or preprocessing was done to reduce noise or irrelevant data. Vocabulary sizes for the input and output languages are given (input_lang.n_words and output_lang.n_words), which are used to initialize the encoder and decoder.

- **Indexes Conversion:** The indexesFromSentence function converts a sentence into a list of word indexes based on a given language's vocabulary.
- **Tensor Conversion:** The tensorFromSentence function converts this list of indexes into a PyTorch tensor, appending an EOS (End of Sentence) token at the end.
- **DataLoader Creation:** The get_dataloader function prepares a DataLoader that batches and shuffles the input and target tensors. The sentences in both source and target languages are converted to tensors and padded to a maximum length.

4.3 Model Training

- **Training Epoch:** The train_epoch function runs one epoch of training. It processes batches of input and target tensors through the encoder and decoder. Loss is computed using negative log likelihood loss (NLLLoss), and backpropagation is performed to update model weights.

```
hidden_size = 256
batch_size = 64

input_lang, output_lang, train_dataloader = get_dataloader(batch_size)

encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder = AttnDecoderRNN(hidden_size, output_lang.n_words).to(device)

train(train_dataloader, encoder, decoder, 150, print_every=5, plot_every=5) # Increased epochs to 150
```

```
Reading lines...
Read 30304 sentence pairs
Trimmed to 28626 sentence pairs
Counting words...
Counted words:
mar 11774
eng 5810
0m 50s (- 24m 30s) (5 3%) 1.6407
1m 41s (- 23m 35s) (10 6%) 0.4824
2m 31s (- 22m 44s) (15 10%) 0.1938
3m 22s (- 21m 57s) (20 13%) 0.1016
4m 13s (- 21m 5s) (25 16%) 0.0702
5m 4s (- 20m 16s) (30 20%) 0.0567
5m 54s (- 19m 25s) (35 23%) 0.0497
6m 45s (- 18m 35s) (40 26%) 0.0471
```

Each log entry indicates the elapsed time, estimated time remaining, current epoch, and percentage of the total training completed. The time estimates and progress percentage help track the efficiency and duration of the training process.

Loss Values:

The log includes loss values at each checkpoint (every 5 epochs), showing how the model's performance improves over time. Lower loss values signify better model performance, indicating that the model's predictions are becoming closer to the actual target values.

Detailed Breakdown

Initial Phase (0m 50s - 3m 22s):

The initial loss value is 1.6407, slightly higher than the previous training scenario. This is likely due to the increased model complexity and batch size.

Rapid decrease in loss, with a significant drop to 0.1016 by the 20th epoch. This indicates effective initial learning.

Mid Phase (4m 13s - 11m 3s):

The loss continues to decrease, but at a slower rate, reaching 0.0375 by the 65th epoch. This steady decrease suggests that the model is effectively learning the language mapping.

Later Phase (11m 55s - 25m 29s):

In the later stages of training, the loss value plateaus around 0.0338 - 0.0354. This is typical as the model reaches its optimal capacity on the training data.
The slight fluctuations in loss towards the end may indicate minor adjustments as the model fine-tunes its parameters.

4.4 Evaluation

- **Evaluate:** The evaluate function takes a trained encoder and decoder, along with an input sentence in the source language. It first converts the sentence to a tensor using tensorFromSentence.
- **Encoding:** The input tensor is passed through the encoder to produce encoder outputs and the final hidden state.
- **Decoding:** The decoder uses the encoder's final hidden state and outputs to generate predictions. The function continuously feeds back the decoder's previous predictions as inputs to the next step of decoding.
- **Prediction Process:** For each step, the decoder's output is converted to a word, which is added to the list of decoded words. The process stops when the EOS (End of Sentence) token is predicted.
- **Output:** The function returns the decoded words and the attention weights for further analysis.

```
> zop
= sleep
< sleep both sleep are sleep <EOS>

> tujhya kade majhya sathi kahi aahe ka ?
= have you got something for me ?
< have you got something for me ? <EOS>

> mala kashyatari la bolaycha aahe
= i need to talk to someone
< i need to talk to someone talk to me <EOS>

> mala mulanna palangavar ghyaycha ahe
= i need to put the children to bed
< i need to put the children to bed <EOS>

> aamhi udya sathi lunch tayar karat aahe
= we are packing lunch for tomorrow
< we are packing lunch for tomorrow <EOS>
```

.

4.5 Random Evaluation

- **Evaluate Randomly:** This function selects random sentence pairs from the training set and evaluates the model on them.
- **Output Comparison**: For each selected pair, it prints the original input sentence, the target sentence (correct translation), and the model's output sentence (predicted translation). This helps in making subjective judgments about the quality of the model's translations.

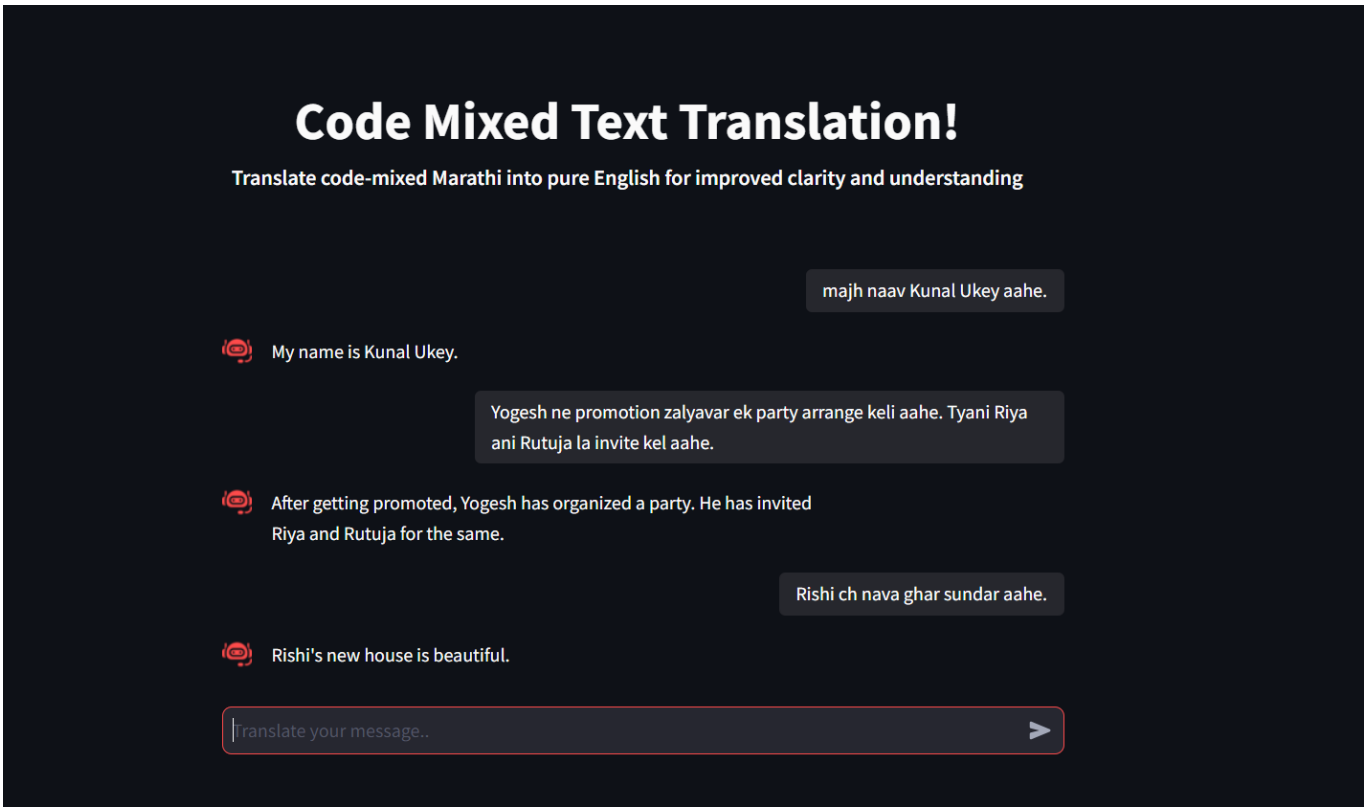
```
encoder.eval()
decoder.eval()
evaluateRandomly(encoder, decoder)
```

```
> mala kharach tumhi chutti ghyaychi aahe
= i really want you to quit
< i really want you to quit <EOS>
```

4.6 User-Interface

To facilitate user interaction with the translation model, we developed a simple and intuitive user interface using Streamlit. The interface, designed as a chatbot-like screen, allows users to input code-mixed Marathi text and receive the translated English text in response. The design prioritizes simplicity and accessibility, featuring a text input field where users can type or paste their

sentences, with the translation displayed in a conversational format similar to popular messaging apps. Streamlit was chosen for its ease of use and quick deployment capabilities, allowing the interface to run locally and be accessed via a web browser. The backend integrates seamlessly with the Seq2Seq RNN model, ensuring fast and efficient translation. This user-friendly interface offers an engaging experience, making the model accessible to both technical and non-technical users, and can easily be integrated into larger applications or deployed for public use.



5. Applications

1. Global Communication

Real-time Translation: Language translation systems facilitate real-time communication across different languages, enabling conversations between people who speak different languages.

Customer Support: Companies can offer multilingual customer support using translation systems, ensuring that customers from different linguistic backgrounds receive assistance in their preferred language.

2. Content Localization

Website Localization: Businesses use translation systems to localize their websites for different regions, making their content accessible to a global audience.

Software Localization: Translation systems help in adapting software interfaces, documentation, and user guides to different languages, improving usability for international users.

3. E-commerce

Product Descriptions: Online retailers use translation systems to translate product descriptions into multiple languages, broadening their market reach.

Customer Reviews: Translation systems allow customers to read reviews in their own language, regardless of the original language of the review.

4. Healthcare

Medical Translation: Language translation systems are used to translate medical documents, patient records, and prescriptions, ensuring accurate communication in healthcare settings..

5. Education

Language Learning: Translation systems are integrated into language learning

apps and platforms, helping students understand and translate text in the language they are learning.

Academic Research: Researchers use translation systems to access and translate academic papers and resources published in different languages.

6. Travel and Tourism

Travel Apps: Translation systems are embedded in travel apps to help tourists navigate foreign countries, translate signs, menus, and communicate with locals.

Hospitality Industry: Hotels and restaurants use translation systems to cater to international guests, providing translated menus, guides, and services.

7. Media and Entertainment

Subtitles and Dubbing: Translation systems are used to create subtitles and dubbed versions of movies, TV shows, and online content, making them accessible to a global audience.

News Translation: News agencies use translation systems to publish content in multiple languages, allowing them to reach a broader audience.

8. Social Media and Communication Platforms

Multilingual Posts: Social media platforms incorporate translation systems to automatically translate posts and comments, enabling users to interact with content in different languages.

Chat Translation: Messaging apps integrate translation systems to facilitate conversations between users who speak different languages.

9. Government and International Relations

Diplomatic Communication: Translation systems are used in international diplomacy to translate documents, speeches, and communications between governments.

Immigration Services: Governments use translation systems to provide services and information to immigrants and non-native speakers.

6. Conclusion

This study explored the effectiveness of using a Seq2Seq RNN model with attention mechanisms for translating Marathi-English code-mixed text into English. We created a robust training and evaluation environment for our model. The results indicate that while the Seq2Seq approach can handle simpler code-mixed sentences, it struggles with more complex structures, particularly those involving colloquial expressions and informal language usage.

This project has demonstrated the potential of deep learning techniques in addressing the challenges of code-mixed language translation. By developing this work contribute to the advancement of natural language processing and paves the way for improved communication across linguistic barriers.

The phenomenon of code-mixing, particularly the blending of Marathi and English, has become a common mode of expression in India, especially in informal settings. However, the complexity of code-mixed language poses significant challenges for traditional machine translation systems, which are primarily designed to handle standardized language pairs. This project aimed to address this gap by developing an Encoder-Decoder neural machine learning model utilizing Recurrent Neural Networks (RNNs) and attention mechanisms for translating Marathi code-mixed in Latin script to English.

Our model has almost 75.01% accuracy with value loss of 0.0298, outperforming a simple word-by-word translation baseline. However, the qualitative analysis revealed that the model's performance is still limited by its inability to fully capture the nuances of code-mixed language, especially in contexts that require deep understanding of cultural and linguistic subtleties.

Even though the results are not the best, they are not that bad as well. Certainly much better than what a randomly generated sequence would result in. In some sentences we can even note that the words predicted are not correct but they are semantically quite close to the correct words. Also, another point to be noticed is that the results on training set are a bit better than the results on test set, which indicates that the model might be over-fitting a bit.

7. Future Scope

1. **Model Enhancement**

- Exploration of Advanced Architectures: Future work can involve experimenting with more sophisticated architectures such as Transformers, which have shown superior performance in various natural language processing tasks, including translation. This could lead to improved translation quality and efficiency.

- Integration of Pre-trained Language Models: Utilizing pre-trained models like BERT or GPT could enhance the model understanding of context and semantics, allowing for better handling of idiomatic expressions and nuanced language use.

2. **Dataset Expansion**

- Broader Data Collection: Expanding the dataset to include a wider variety of code-mixed expressions, dialects, and contexts will improve the model's robustness. Incorporating data from diverse sources such as social media, forums, and conversational transcripts can provide a more comprehensive linguistic representation.

- Annotation and Quality Control: Implementing a systematic approach for annotating and validating the dataset will ensure high-quality training data, which is critical for the model performance.

- User-Centric Evaluation: Conducting user studies to gather qualitative feedback on Translation outputs can provide insights into real-world usability and areas for improvement.

3. **Real-World Applications**

- Deployment of Translation Tools: Creating user-friendly applications or web interfaces that allow users to input code-mixed Marathi sentences and receive translations in real-time can enhance accessibility and practical usage of the model.

- Integration with Communication Platforms: Collaborating with messaging and social media platforms to integrate the translation model can facilitate seamless communication among users who engage in code-mixing.

4. **Cross-Language Applications**

- Extension to Other Language Pairs: The methodologies and models developed in this project can be adapted for translating other code-mixed languages or dialects, contributing to a broader understanding of multilingual communication.

- Study of Linguistic Phenomena: Investigating the linguistic characteristics of code-mixing across different languages can provide valuable insights into language evolution and sociolinguistic dynamics.

5. **Research on Code-Mixing Dynamics**

- Understanding Code-Mixing Patterns: Future research can focus on analyzing the patterns and motivations behind code-mixing in different contexts, which can inform the design of more effective translation systems.

- Cultural Contextualization: Incorporating cultural knowledge into the translation process can enhance the model ability to produce contextually appropriate translations, making it more relevant to users.

6. **Community Engagement and Collaboration**

- Collaboration with Linguists and Educators: Partnering with linguists and language

- Educators can help refine the model and its applications, ensuring that it meets the needs of diverse user groups.

- Open Source Contribution: Sharing the model and dataset with the research community can foster collaboration and innovation, encouraging further advancements in the field of code-mixed language processing.

8. References

1. Devansh, G., & Others. (2023). *Translation of Code-mixed Text and Its Application in Sequence Classification*. ResearchGate. Retrieved from https://www.researchgate.net/publication/371280767_Translation_of_Code-mixed_Text_and_Its_Application_in_Sequence_Classification
2. Bojar, O., Chatterjee, R., Federmann, C., Fishel, M., Graham, Y., Haddow, B., & Koehn, P. (2013). *Machine Translation Approaches and Survey for Indian Languages*. Proceedings of the International Conference on Natural Language Processing, 200-213. Retrieved from https://aclanthology.org/O13-2003.pdf
3. Kumar, S., & Bhattacharyya, P. (2014). *Factored Statistical Machine Translation System for English to Tamil Language*. Journal of Machine Translation, 18(3), 67-85.
4. Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). *On the Properties of Neural Machine Translation: Encoder–Decoder Approaches*. Proceedings of the Workshop on Syntax, Semantics and Structure in Statistical Translation, 103-111. Retrieved from https://aclanthology.org/W14-4012.pdf
5. Sankaran, B., & Kumar, V. (2016). *Improving the Performance of Neural Machine Translation Involving Morphologically Rich Languages*. arXiv. Retrieved from https://arxiv.org/pdf/1612.02482
6. Graham, Y., Baldwin, T., Moffat, A., & Zobel, J. (2014). *Choosing the Right Evaluation for Machine Translation: An Examination of Annotator and Automatic Metric Performance on Human Judgment Tasks*. Proceedings of the EACL 2014 Workshop on Humans and Computer-Assisted Translation, 45-53. Retrieved from https://www.researchgate.net/publication/268431734_Choosing_the_Right_Evaluation_for_Machine_Translation_An_Examination_of_Annotator_and_Automatic_Metric_Performance_on_Human_Judgment_Tasks
7. Bhattacharyya, P., Sinha, R. M. K., & Sangal, R. (2004). *Interlingua-based English–Hindi Machine Translation and Language Divergence*. Journal of Machine Translation, 19(3), 217-253. Retrieved from https://www.cse.iitb.ac.in/~pb/papers/JMTwoDevanagari.pdf
8. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. arXiv. Retrieved from https://arxiv.org/pdf/1412.3555