CS 6140 Course Project: Fall 2023

Neural Machine Translation model for Bi-lingual conversation using LSTM and Transformers

Abstract Overview 2
Problem Analysis 2
Environment Setup 3
Prepare datasets 4
Data Statistics 4
Preprocessing 5
Creating the vocabulary 7
Creating the model 8
Training the model 9
Steps to run 10
Model Comparison and Analysis 11
LSTM vs. Pretrained Transformer 11
Conclusion 11
Future Work 12

Abstract Overview

Natural language processing (NLP) has got great development with deep learning techniques. In the sub-field of machine translation, a new approach named Neural Machine Translation (NMT) has emerged and gotten massive attention from both academia and industry. Our aim with the project has been to understand the working of various machine learning algorithms (esp Neural Networks), their architecture and impact on accuracy for translating data from one language to another. We worked with LSTM (Long Short Term Memory) and Transformers with attention. Our focus majorly has been with LSTMs due to their short training time, while we worked around working with Transformers to understand their impact on machine translation. Per our initial analysis, we realized that Transformers in general take a lot of time to train and learn while providing better accuracy while working towards machine translation, where attention is the key. This state-of-the-art algorithm is an application of deep learning in which massive datasets of translated sentences are used to train a model capable of translating between any two languages. One of the older and more established versions of NMT is the Encoder Decoder structure. This basically is two RNN models, one which encodes the source language while the other decoding the tokens(encoded vectors) to the respective target language. When coupled with the power of attention mechanisms, this architecture can achieve impressive results as we discuss through this project.

Problem Analysis

What is the problem?

The problem at hand involves developing a Neural Machine Translation (NMT) model to facilitate seamless translation between English and Hindi, and vice versa. The primary objective is to overcome language barriers and enhance communication across these two languages. The motivation stems from the increasing need for effective language translation tools, driven by globalization, multicultural interactions, and the desire for inclusive and accessible information exchange. With google translate dominating the translation market, we wanted to understand and implement a similar approach in order to understand the basic underlying math behind developing translation models.

Why is this problem interesting? Is this problem helping us solve a bigger task in some way for society? Where would we find use cases for this problem in the community?

This problem is intriguing due to its direct impact on fostering cross-cultural communication and breaking down language barriers. A successful NMT model for English-Hindi translation contributes to a more interconnected and inclusive global community. The utility extends to various domains, including business, education, healthcare, and technology, where accurate and efficient language translation is essential for effective communication and knowledge dissemination. On a larger scale, as a part of our ongoing analysis, we look towards incorporating additional languages so as to improve communication and eradicate middlewares for people to communicate.

What is the approach you propose to tackle the problem? What approaches make sense for this problem? Would they work well or not?

The proposed approach involves employing Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), for sequence-to-sequence translation. LSTMs are well-suited

for capturing sequential dependencies, making them effective in language translation tasks. The model architecture includes an encoder-decoder structure to capture relevant information during translation. While transformers are popular for NMT, LSTMs offer a computationally efficient alternative for certain language pairs. We have tried our hands at a pre-trained transformer model to understand and cross-validate our findings. Results are discussed in the upcoming sections.

What is the rationale behind the proposed approach? Did you find any reference for solving this problem previously? If there are, how does your approach differ from theirs (if any)?

The rationale for choosing LSTMs lies in their ability to capture long-range dependencies in sequential data, which is crucial for language translation. While transformer models are prevalent, especially in large-scale language models, LSTMs offer a balance between performance and computational efficiency. Previous references and research in NMT informed the choice of LSTMs, and the novelty in our approach lies in tailoring the model architecture to specifically address the nuances of English-Hindi translation.

What are the key components of the approach and results? Also, include any specific limitations.

Key components include embedding layers for token representation, LSTM layers for sequential learning, and encoder-decoder mechanisms for enhanced translation quality. The model will be trained on parallel English-Hindi corpora, and results will be evaluated using metrics like BLEU scores. However, potential limitations may include the model's sensitivity to data quality, challenges in handling idiomatic expressions, and potential difficulties in rare word translation. The model might also be prone to overfitting due to the words and translations it is trained on.

Environment Setup

The basic libraries and dependencies require to set the project up include:

```
import string
import re

from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import tensorflow as tf
import nltk

from keras.layers import Input, LSTM, Embedding, Dense
from keras.models import Model, load_model
from keras import metrics
from keras.losses import SparseCategoricalCrossentropy
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint, TerminateOnNaN

from nltk.translate.bleu_score import corpus_bleu

from datasets import load_dataset

import matplotlib.pyplot as plt
```

- nltk - for tokenizing and calculating bleu score
- keras - to set and train LSTM model
- datasets(to load datasets from Hugging Face) - To load and use dataset and pretrained model from hugging face
- numpy, pandas, re and string for data preprocessing
- Matplotlib - to chart and plot results
- Tensorflow - in order to convert our raw data into tensors for tokenizing later
- transformers - to load and check transformer pre-trained model (Helsinki-NLP/opus-mt-en-hi)

Prepare datasets

Data Statistics

Based on the raw data we received, we ran a basic statistical model to analyse and choose an appropriate length for each of the sentences. As per the data that we received, we chose 10 to be the average length since 50% of our data had this length. Choosing a higher length lead to very high training time, so keeping it at 10 was ideal.

In [16]:
```
# word count stats for source
df_eh[['src_len']].describe(percentiles = [.25, .5, .75, .95]).transpose()
```

Out[16]:

	count	mean	std	min	25%	50%	75%	95%	max
src_len	1527115.0	16.307187	16.345626	3.0	4.0	12.0	22.0	46.0	1382.0

In [17]:
```
# word count stats for target
df_eh[['trgt_len']].describe(percentiles = [.25, .5, .75, .95]).transpose()
```

Out[17]:

	count	mean	std	min	25%	50%	75%	95%	max
trgt_len	1527115.0	15.295724	14.953927	3.0	4.0	11.0	21.0	42.0	1919.0

Preprocessing

Majority of our project involved understanding, preprocessing and tokenizing our dataset. The raw dataset that we used is from Hugging face (https://huggingface.co/datasets/cfilt/iitb-english-hindi) consisting of 1.66 mn training records and 2.51K test records.

:
```
print(len(eng_sen), len(hin_sen))
print()
eng_sen[:3], hin_sen[:3]
```

```
1044136 1044136
```

:
```
(['give your application an accessibility workout',
  'accerciser accessibility explorer',
  'the default plugin layout for the bottom panel'],
 [' अपने अनुप्रयोग को पहुंचनीयता व्यायाम का लाभ दें ',
  ' एक्सेर्साइसर पहुंचनीयता अन्वेषक ',
  ' निचले पटल के लिए डिफोल्ट प्लगइन खाका '])
```

We preprocessed the data by removing extra punctuations, stripping data, and removing foreign words from our language model. Our raw data is structured as a dictionary which we extracted into lists termed eng_sen (english sentence) and hin_sen (hindi sentence) for the ease of our reference.

```
en_data = []
hi_data = []

cnt = 0

for (en, hi) in zip(eng_sen, hin_sen):
    l = min(len(en.split()), len(hi.split()))
    if l <= maxlen:
        en_data.append(en)
        hi_data.append(hi)
        cnt += 1
    if cnt == total_sentences:
        break

hi_data = ['<START> ' + hi + ' <END>' for hi in hi_data]
```

Our aim was to extract sentences with length 10 understanding the complexity of data and managing the training of the dataset. We thus filtered our data upon this metric. Post this we added the and token to each of the sentence of the target language. If the sentence length after adding these tokens was lesser than 10, we added padding at the end of each of these sentences.

Creating the vocabulary

```
en_tokenizer = Tokenizer(filters='', oov_token='', lower=False)
en_tokenizer.fit_on_texts(en_data)
en_sequences = en_tokenizer.texts_to_sequences(en_data)

hi_tokenizer = Tokenizer(filters='', oov_token='', lower=False)
hi_tokenizer.fit_on_texts(hi_data)
hi_sequences = hi_tokenizer.texts_to_sequences(hi_data)

english_vocab_size = len(en_tokenizer.word_index) + 1
hindi_vocab_size = len(hi_tokenizer.word_index) + 1
print("English Vocab Size: ", english_vocab_size)
print("Hindi Vocab Size: ", hindi_vocab_size)
```

```
English Vocab Size:  19694
Hindi Vocab Size:  19688
```

```
encoder_inputs = pad_sequences(en_sequences, maxlen=maxlen, padding='post')
```

```
# Prepare decoder data
decoder_inputs = []
decoder_outputs = []

for hi in hi_sequences:
    decoder_inputs.append(hi[:-1])
    decoder_outputs.append(hi[1:])

decoder_inputs = pad_sequences(decoder_inputs, maxlen=maxlen, padding='post')
decoder_outputs = pad_sequences(decoder_outputs, maxlen=maxlen, padding='post')
```

To create a comprehensive vocabulary from our filtered dataset, we utilize Keras' Tokenizer library. This tool breaks down the text into tokens and assigns unique numerical identifiers to each token, forming the vocabulary. Following vocabulary creation, we proceed to tokenizing them.

In the process of preparing inputs for the encoder, we apply padding at the end of each sentence, denoted by padding='post'. This choice of padding is particularly significant for neural network architectures, such as recurrent neural networks (RNNs) or transformers, where inputs must have a consistent length.

Padding plays a crucial role in ensuring uniform input lengths, facilitating the construction of batches for efficient training. It is essential for creating a structured neural network that handles inputs consistently. The structured architecture becomes particularly relevant for models with a decoder, as it enables the network to provide even inputs and receive corresponding outputs.

In summary, by incorporating padding at the end of sentences, we not only establish a consistent input structure but also contribute to the efficient training and convergence of the neural network, especially in architectures involving a decoder.

Finally after preprocessing is complete we split our dataset into training and test with 95% of the preprocessed data being used for training while 5% is used for testing and calculating BLEU score.

```
# Training and Testing split
# 95%, 5%
split = int(0.95 * total_sentences)

X_train = [encoder_inputs[:split], decoder_inputs[:split]]
y_train = decoder_outputs[:split]

# Test data to evaluate our NMT model using BLEU score
X_test = en_data[:split]
y_test = hi_data[:split]

print(X_train[0].shape, X_train[1].shape, y_train.shape)
```

```
(47500, 10) (47500, 10) (47500, 10)
```

Creating the model

We designed a Long Short-Term Memory (LSTM) model for sequence-to-sequence translation tasks, focusing on language translation from English to Hindi. The model architecture comprises of an encoder-decoder structure, a widely used technique for handling sequential data.

The encoder processes the input sequences (English sentences) and extracts meaningful representations. Key components of the encoder include:

Embedding Layer: This layer converts the input tokens into dense vectors, facilitating the neural network's understanding of word relationships.
LSTM Layer: Utilizing Long Short-Term Memory cells, the encoder captures contextual information from the input sequences. The LSTM layer returns the final hidden memory state and the carry state for subsequent use in the decoder.
The decoder generates the target sequences (Hindi translations) based on the encoded information. Notable elements of the decoder include:

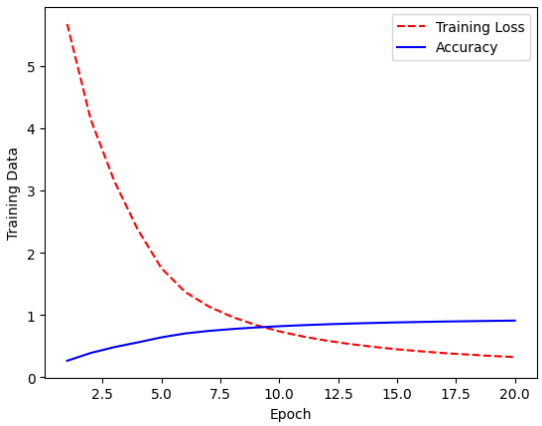
Embedding Layer: Similar to the encoder, this layer transforms the target tokens into dense vectors.
LSTM Layer: The decoder LSTM processes the embedded target sequences while considering the context provided by the encoder. It returns sequences of outputs and the final states.
A Dense layer with a softmax activation function is added to produce probability distributions over the Hindi vocabulary. This allows the model to predict the most likely Hindi token for each position in the output sequence.

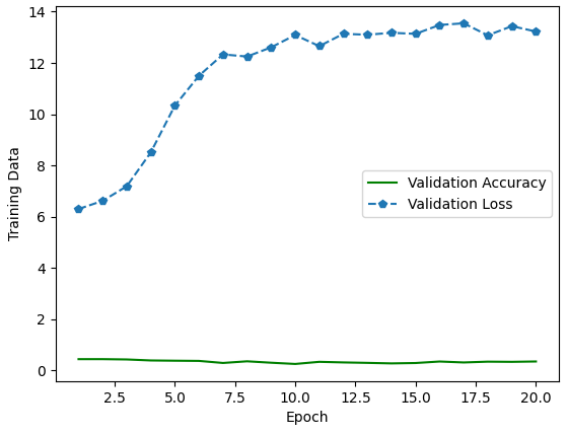
The model is compiled using the Adam optimizer and Sparse Categorical Cross Entropy as the loss function. The choice of the optimizer and loss function is motivated by their effectiveness in sequence-to-sequence tasks. The model is configured to optimize for accuracy during training.

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_5 (InputLayer)        [(None, None)]            0         []

 input_6 (InputLayer)        [(None, None)]            0         []

 embedding_2 (Embedding)     (None, None, 256)         5041664   ['input_5[0][0]']

 embedding_3 (Embedding)     (None, None, 256)         5040128   ['input_6[0][0]']

 lstm_2 (LSTM)               [(None, 256),             525312    ['embedding_2[0][0]']
                              (None, 256),
                              (None, 256)]

 lstm_3 (LSTM)               [(None, None, 256),       525312    ['embedding_3[0][0]',
                              (None, 256),                        'lstm_2[0][1]',
                              (None, 256)]                        'lstm_2[0][2]']

 dense_1 (Dense)             (None, None, 19688)       5059816   ['lstm_3[0][0]']

==================================================================================================
Total params: 16192232 (61.77 MB)
Trainable params: 16192232 (61.77 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

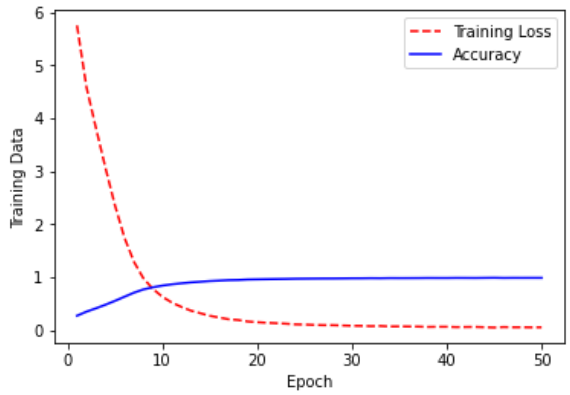
Training the model

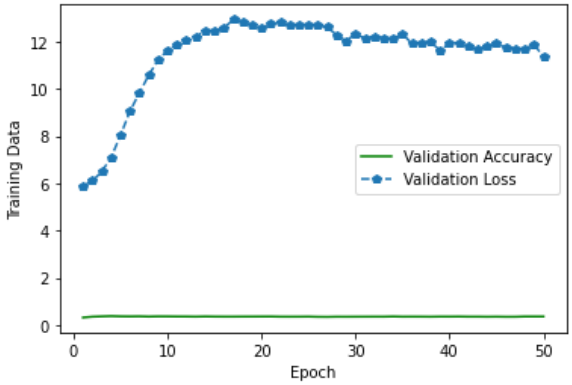
Below describes our model's training and validation sets accuracy vs loss curves over 20 epochs for model with 250k data records:





For model with 50k records and 50 epochs, below were our results:





We observe that we receive an accuracy of around 91% while the epoch loss also considerably reduces using LSTMs.

The notebook also has experimental results of training and validation accuracy vs loss for 20, and 50 epochs respectively with varying amounts of training set size for us to analyze and conclude the BLEU scores.

Steps to run

1. Since the saved models are over 100 mb, github is not allowing us to directly upload these models. Extract the folders (respective models) from given link: https://drive.google.com/file/d/1mKVcVLIhvDDMuOIyEyfvmtLxpKb0zL8z/view
2. Load the preprocessed data
3. Load the model for LSTM
4. Set up encoder and decoders for testing the model
5. Predict texts
6. Calculate and print BLEU scores in order to verify accuracy

Model Comparison and Analysis

LSTM vs. Pretrained Transformer

To benchmark the performance of our LSTM-based sequence-to-sequence model, we conducted a comparative analysis with a pretrained Transformer model (Helsinki-NLP/opus-mt-en-hi) obtained from Hugging Face, specifically trained on the Opus dataset. Given the nature of the Transformer architecture, which excels in capturing long-range dependencies through attention mechanisms, we anticipated superior results.

The BLEU (Bilingual Evaluation Understudy) score is a metric commonly used to evaluate the quality of machine-generated translations in Natural Language Processing (NLP), particularly in the context of Neural Machine Translation (NMT). BLEU was designed to align with human intuition about translation quality and has become a standard metric for comparing the

performance of different translation models. It evaluates the precision of the generated translation by comparing it to one or more reference translations. It considers n-grams (contiguous sequences of n items, typically words) in the generated translation and checks how many of these n-grams are also present in the reference translation.

Surprisingly, our findings indicate that the LSTM model performed remarkably well in comparison, demonstrating similar levels of accuracy and achieving competitive BLEU scores. Despite the architectural differences, the LSTM model proved to be robust in capturing contextual information and generating coherent translations. This suggests that for certain language translation tasks, the inherent capacity of LSTMs to capture sequential dependencies might be on par with the attention mechanisms employed by Transformers.

The pretrained Transformer model, while undoubtedly powerful and capable of capturing intricate details, did not exhibit a substantial performance advantage in our specific task. This observation underscores the importance of task-specific evaluations and the nuanced nature of model comparisons. Our results encourage further exploration into the effectiveness of traditional recurrent architectures like LSTMs, particularly in scenarios where pretrained models might not necessarily outperform their sequential counterparts.

Conclusion

This LSTM-based sequence-to-sequence model serves as a powerful tool for English-to-Hindi language translation. The detailed architecture, training configuration, and evaluation metrics establish a foundation for further experimentation and improvements. Results below indicate the outperformance of Transformers, the reason for its recent hype and the future of machine translation with Transformers.

BLEU score for Transformers (Baseline): 0.13276982988333147

Epochs	Train size	BLEU Score
50	50k records	0.5204079142754721
20	250k records	0.5446195830945151

Future Work

Our aim with this project is to take this further, training bidirectionally for hindi and english. We look towards improving our accuracy, working with transformers and achieving a consistent state in order to work with more data. We also look towards expanding this project to include a third language where the model understands the introduction of the third language and is able to translate from hindi to the third language using english as the middleware decoder. In order to

achieve this, we are building confidence on translating from hindi to english with an accuracy of 95% or higher.

References

1. https://medium.com/geekculture/english-to-hindi-text-translation-using-marianmt-models-from-huggingface-187b4719809e
2. https://towardsdatascience.com/neural-machine-translation-15ecf6b0b
3. https://huggingface.co/datasets/cfilt/iitb-english-hindi
4. https://huggingface.co/damerajee/hindi-english
5. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
   https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences
6. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
7. https://keras.io/api/layers/core_layers/dense/
8. https://www.nltk.org/api/nltk.translate.bleu_score.html