Neural Machine Translation of Hinglish to English

Anjani Ganapathy and Chini Lahoti
PLIN0072 Final Project

1 Introduction

Code-mixed languages are growing to be extremely used — especially on social media — as a byproduct of more language communities interacting with one another in this post-colonial, globalized world. As native speakers of both Hindi and English, we have especially observed the use of and have used Hinglish, a code-mixed version of both languages, a lot in our personal lives. Additionally, machine translation of code-mixed languages like Hinglish presents an underexplored challenge in the field of natural language processing. In this project, we wanted to investigate how well a neural machine translation (NMT) model can translate code-mixed languages like Hinglish. This task is important not only because of the increasing prevalence of code-mixing in multilingual communities, but also because effective translation systems can help bridge communication gaps across diverse linguistic groups. Unlike monolingual translation tasks, where there are larger, parallel corpora more readily available, translating code-mixed inputs often lack such resources, making it a compelling problem to tackle using modern machine learning techniques.

2 Related Works

As code-mixing gains prevalence, especially on internet forums and social media, work on machine translation of code-mixed languages has increased rapidly. This includes a small but growing body of work on Hinglish. Much of this body of work is dedicated to exploring and creating Hinglish datasets, as reliable and diverse data has been difficult to locate in the past. Vivek Srivastava and Mayank Singh, who are responsible for the two Hinglish datasets creation projects that comprise most of our dataset, outline six challenges with Code-Mixed NMT tasks, especially given that the data is noisier due to its source being primarily casual online platforms.

1. Ambiguity in language identification is caused by unintentional homonyms created when using the Roman alphabet to represent Hindi words.
2. The romanization of Hindi also presents a challenge with spelling variations as there is no standardization in spelling choices made.
3. The recognition of named entities is also difficult when handling code-mixed data.
4. Due to the nature of social media and other casual online forums, the style of writing tends to be very informal (i.e., using abbreviations or shorthand to express words).
5. In addition to informal writing style, punctuation is often skipped or misplaced.
6. Lastly, since the data is pulled without its context, it makes machine translation significantly more challenging.

(Srivastava and Singh 2020, p. 42-43)

Dataset	Size	Generation Type	Source: Domain
Hinglish TOP	~10k	Human generated	AI assistant conversations: navigation, events, alarm, messaging, music, reminder, timer, and weather (Chen et al., 2020, p. 5091); (Agarwal et al., 2023)
CMU Hinglish DoG	~10k	Human generated	CMU DoG (Document Grounded Conversations) dataset, containing conversations about Wikipedia articles for popular movies (Zhou et al., 2018, p.708)
HinGE	~155k	Combined human and synthetically generated	IIT-B corpus: Conversations, either human-human or human-computer (Srivastava & Singh, 2021, p.201-204)
PHINC	~14k	Human generated	Twitter, Facebook: sports, Bollywood, politics, social events (Srivastava & Singh, 2020)

Table A: Datasets aggregated in the english-to-hinglish dataset used to train and evaluate our model

While Srivastava and Singh lay the groundwork by identifying the unique challenges posed by Hinglish code-mixing, a smaller set of this work focuses on exploring neural machine translation of Hinglish to either pure Hindi or English. The work by Agarwal et al., 2021 is one of the most recent publishings in which Hinglish to English translation is explored. In this work, pre-trained multilingual models like mBART and mT5 were investigated for their ability to handle code-mixed inputs, given that none are trained on code-mixed data. These models were fine-tuned on Hinglish-English parallel corpora that are the output of previous work, such as PHINC. The authors of this study were able to achieve a much higher BLEU score than prior works, showing the power of these pre-trained multilingual models.

We benefit from having access to several established Hinglish data sources, spanning human-created and synthetically generated data. With these, we aim to provide insight into the ability of a trained-from-scratch neural network to translate Hinglish code-mixed inputs into our target language, English. Part of the goal of this work is to determine whether pre-trained multilingual models are necessary for translation if a substantially-sized corpus is available.

3 Data

For this project, we used an existing dataset hosted on HuggingFace. The dataset, called english-to-hinglish, aggregates several smaller Hinglish to English parallel corpora. The sources aggregated in this dataset are detailed in table A.

The data we used is a combination of human-annotated and synthetically generated corpora. About 20k sentences were human-annotated, and the remaining 170k were synthetically generated.
The dataset did not include pre-set training and test splits. For this task, we chose to split the data into training and test sets using HuggingFace's provided train_test_split() function. Using this function, we created a training set that included 95% of our data and a test set that included 5% of our data. This function automatically shuffles the data

it is splitting. Shuffling is important here because our dataset consists of text that comprises different domains, and it is important that our model is trained on a representative sample of this data. The size of the test set was chosen to be small (5% of our dataset) because for NMT, training a model on as much data as possible is crucial to the success of it. 5% of our dataset is still close to 10k rows of data, which should still be an adequate amount of data to evaluate the model with.

4 Model

For this task, we built a sequence-to-sequence model using a standard encoder-decoder LSTM architecture. The encoder takes tokenized sentences in Hinglish as inputs and outputs a representation of these tokens as tensors. The decoder takes the representation created by the encoder and iteratively uses this, along with its own embeddings and context from self attention, to generate predictions token-by-token. To connect the outputs of the encoder with the decoder and its self attention mechanism, we built a sequence to sequence class.

Hyperparameter	Value	Rationale
Learning rate	Run 1: 1e-2 Run 2: 1e-3	At first, we decided to use a higher learning rate in the hopes of seeing tangible results quickly. Later, we decided to decrease from 1e-2 to 1e-3 as this is more compatible with the Adam optimizer and gives the model more time to learn.
Number of layers	2	We opted to give both our encoder and decoder a smaller number of layers because as the number of layers increases, the model's ability to overfit or memorize our training data increases.
Embedding size	300	This seems to be a standard embedding dimension across use cases.
Hidden size	512	Our hidden layer size was first set to 128, but we found that this led to high computation costs because the model was trying to compress the embeddings to feed into a smaller hidden layer. To alleviate this, we updated the size of the hidden layer to 512, the closest power of 2 higher than the size of the embeddings.
Batch size	128	General guidance suggests using as large of a batch size as possible to increase computational efficiency.
Dropout	0.2	We included this dropout value because it seems to be the value used by many other Seq2Seq NMT models. Both the encoder and decoder use dropout in order to add in randomness that serves to counteract potential overfitting.
Teacher forcing rate	0.5	Adding teacher forcing helps the model train more accurately by decoding using the previous target token rather than its previous predicted token, which could be wildly incorrect, especially during early training. A value of 0.5 allows a moderate amount of target tokens to be used for training without the model becoming reliant on them
Epochs	50	Generally, 50 epochs is considered a good minimum for training. Because of computational limits, we chose to stay on the lower end of the range of epochs used for training

Table B: Hyperparameters used when training and evaluating our model

To optimize from-scratch model training, we made a few key choices. First, we did not use pre-trained embeddings because we could not find publicly-accessible Hinglish code-mixed embeddings to use. Rather, we opted for letting the model learn its own embeddings based on the training data provided. Additionally, the decoder utilizes Bahdanau attention in order to capture representations of the full sequence prior to the current token being decoded. Adding a self attention mechanism can help the model create more nuanced representations of the source language input. Bahdanau attention, though more computationally expensive, can perform better on tasks such as translation of longer sequences, and we had many long sentences in our data that we thought could benefit from this. Initially, our decoder did not use attention, but we chose to add it in when our loss score wasn't improving in a meaningful way.

5 Methods

Regarding inputs to the model, we chose to tokenize our data at a word level. Because both Hindi and English use spaces as word boundaries, it was easy to tokenize at a word level. Additionally, any pre-trained tokenizers we found did not seem to perform well for Hinglish, where sentences contain words in two languages, and words are themselves sometimes mixed.

We ran training on the model twice and used the hyperparameter values outlined in **Table B**. The first run was our first attempt at establishing a baseline for performance, but our final loss was quite high. In the second run, we fixed a few things with our model (see Analysis) and updated the learning rate to a more reasonable rate for the Adam optimizer.

After training the model, we chose to evaluate the model using the BLEU (Bilingual Evaluation Understudy) score, a standard metric for machine translation tasks. This score calculates the similarity between the model's predicted translation and the target translation by calculating the amount of overlapping n-grams between them. The BLEU score takes on a value between 0 and 1, with 1 meaning the translations are identical and 0 meaning they have no overlap. In order to calculate this score, we used the Natural Language Toolkit (NLTK)'s in-built corpus_bleu() function.

6 Results

Due to limitations on compute resources, we were only able to train the model twice in its entirety.

We first ran the model for 50 epochs using the set of hyperparameters discussed in the Methods section, including a learning rate of 1e-2. The initial loss score for the model was 3.3774. Losses, then, decreased steadily until about epoch 30, after which they plateaued around 2.26 (see figure 1). The final loss score at epoch 50 was 2.2651. We evaluated the model on the full set of test data and the averaged BLEU score across all test sentences was very close to 0, at 4.313e-232. This indicates that our model was not generating sentences that were close at all to the target sentences. We thought that this result could have been due to our model producing semantically similar but overtly different translations, but when investigated, this did not seem to be the case. Rather, our model was generating nonsensical translations that had no apparent tie to the source or target sentences.

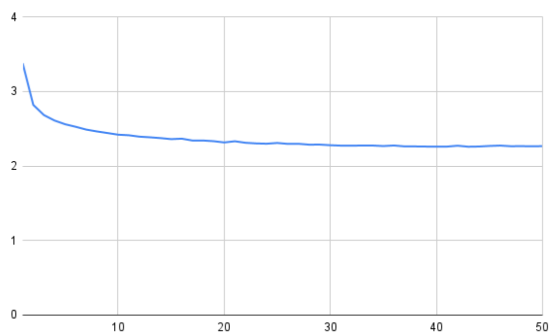

Figure 1: Loss vs Epoch Plot from Run 1

We ran the model again after making infrastructural improvements (see Analysis) and one hyperparameter change, namely, changing the learning rate to 1e-3. The initial loss score for this model was 4.2799. Unlike our first model, losses decreased steadily for the full 50 epochs, though the rate of decrease was quite slow by epoch 15 (see figure 2). The final loss for this model at epoch 50 was 0.821, which is much lower than that of the first model. We evaluated the model again on the full set of test data and the averaged BLEU score was 1.579e-4. Though still quite close to 0, it is a much better score than that of the previous model, though predicted translations are still nonsensical.

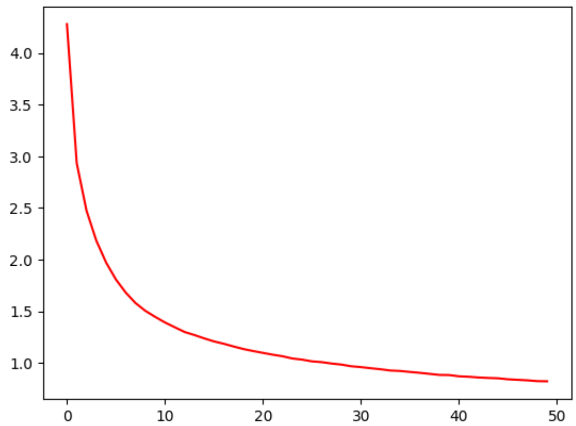

Figure 2: Loss vs Epoch Plot from Run 2

7 Analysis

Observing our first model's poor performance, both in training and in evaluation, inspired us to reevaluate our code on an infrastructural level. This was partially caused by not including start- and end- of sequence tokens in our output. Additionally, we were not properly masking for padding tokens in our encoder-decoder architecture and the cross-entropy loss, which created more noise for the model than needed. We also were not using dropout in the encoder, which was an oversight. Next, we updated our decoder to use beam search for decoding rather than greedy decoding, which can help with improving translation quality by selecting for more probable sequences of tokens rather than one token at a time. To make sure we were only passing in salient parts of a target sequence when evaluating, we cleaned the sequences of the padding, start- and end- of sequence tokens. Lastly, we added the Smoothing Function from the NLTK corpus BLEU package to accommodate for the shorter sequences.

After making these improvements, we trained the model in its entirety again, which did result in a reduced loss score. However, given that our BLEU score was still quite low, it appears that there are still improvements to be made.

For one, a larger dataset could lead to better results — machine translation is an ML task that benefits greatly from having robust training data. Our dataset, at just below 200k rows, is closer in size to the lower threshold of an ideal dataset for this task. Second, tokenizing at a sub-word level for Hinglish could be beneficial. Finding a code-mixed tokenizer that worked well for Hinglish was challenging, even though there are plenty of monolingual tokenizers available for both Hindi and English. Furthermore, ideally, given more time and computational resources, we would have used a hyperparameter optimization process to better tune our model's hyperparameters. Doing this requires running the model iteratively, and because NMT is inherently computationally expensive, this was out of scope for the current timeline. Lastly, in evaluating our model, calculating perplexity as well within the validation loop would be not only useful for hyperparameter tuning but also making sure we're not overfitting the model.

8 Conclusion

In this project, we explored the feasibility of training a neural machine translation model from scratch to translate Hinglish code-mixed text into English. Using an aggregated dataset of both human-annotated and synthetic data, we implemented a Seq2Seq LSTM model with Bahdanau attention.

We can see future work on this topic taking a multitude of different paths. One thing that we would have greatly benefited from is a meaningful and sizable parallel corpus. This could be generated either manually with annotations done by hand or synthetically with human revision, building further on the works of PHINC and HinGE. Another avenue that could be worth exploring is code-mixed speech data since code-mixing tends to occur more in an oral environment than a written one. Lastly, expanding the scope of the task to other code-mixed languages could be an interesting way to compare against other multilingual models, traditionally trained on a plethora of monolingual translation corpora.

References

Anmol Agarwal, Jigar Gupta, Rahul Goel, Shyam Upadhyay, Pankaj Joshi, and Rengarajan Aravamudhan. 2023. CST5: Data Augmentation for Code-Switched Semantic Parsing. In Proceedings of the 1st Workshop on Taming Large Language Models: Controllability in the era of Interactive Assistants!, pages 1–10, Prague, Czech Republic. Association for Computational Linguistics.

Kangyan Zhou, Shrimai Prabhumoye, and Alan W Black. 2018. A Dataset for Document Grounded Conversations. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 708–713, Brussels, Belgium. Association for Computational Linguistics.

Vibhav Agarwal, Pooja Rao, and Dinesh Babu Jayagopi. 2021. Hinglish to English Machine Translation using Multilingual Transformers. In Proceedings of the Student Research Workshop Associated with RANLP 2021, pages 16–21, Online. INCOMA Ltd..

Vivek Srivastava and Mayank Singh. 2020. PHINC: A Parallel Hinglish Social Media Code-Mixed Corpus for Machine Translation. In Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020), pages 41–49, Online. Association for Computational Linguistics.

Vivek Srivastava and Mayank Singh. 2021. HinGE: A Dataset for Generation and Evaluation of Code-Mixed Hinglish Text. In Proceedings of the 2nd Workshop on Evaluation and Comparison of NLP Systems, pages 200–208, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-Resource Domain Adaptation for Compositional Task-Oriented Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5090–5100, Online. Association for Computational Linguistics.

Appendix

Link to Colab Notebook